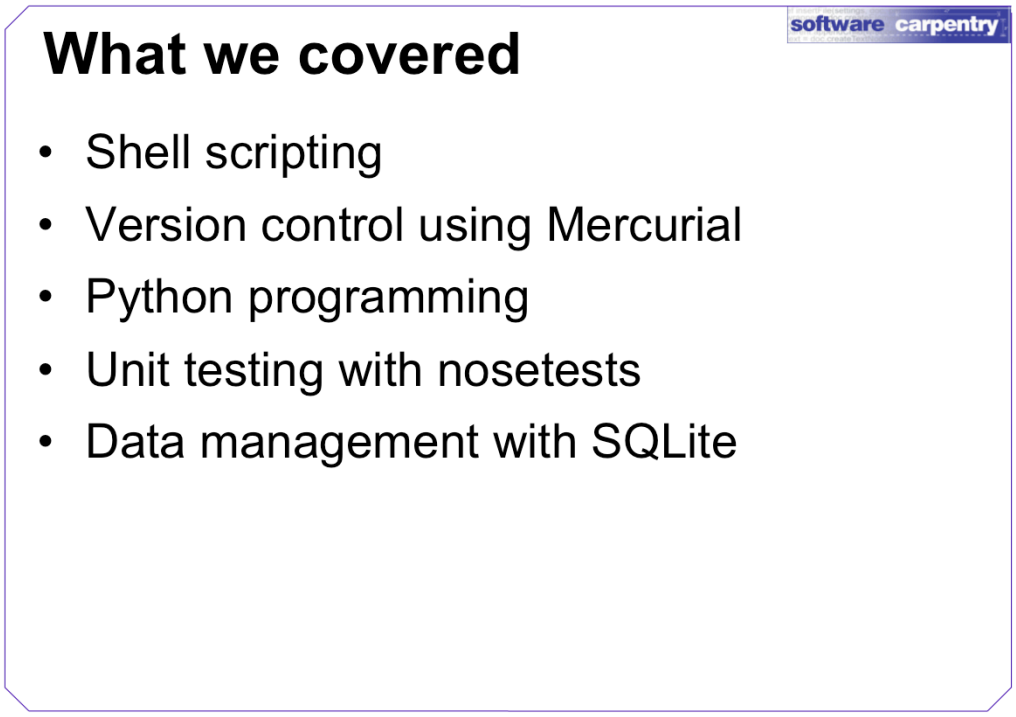
Putting it all together

Greg Wilson
Software Carpentry
Mike Jackson
The Software Sustainability Institute



On the surface, this was what we covered.

But our aim was not to teach these tools, but to show how tools such as these deliver powerful benefits which make a positive contribution to your research.

Shell

- Reduce errors by re-using tried-and-tested components
- String together components into powerful computational and data processing pipelines
- Avoid re-inventing the wheel and free up time to do research

Version control

- Record the provenance of your code, documents or configuration files
- Track which version of your software produced the data you used in a paper, presentation, poster or thesis
- Back up your work
- Productively collaborate

Version control gives you who changed what and when.
Remember that you need to provide the why!

Python

- Make it easier for colleagues to spot bugs in your code
- Help others to validate what you did by peer review
- Help others replicate or reproduce what you did
- Encourage others to reuse what you did
- Reduce your “truck factor”

Modular code.

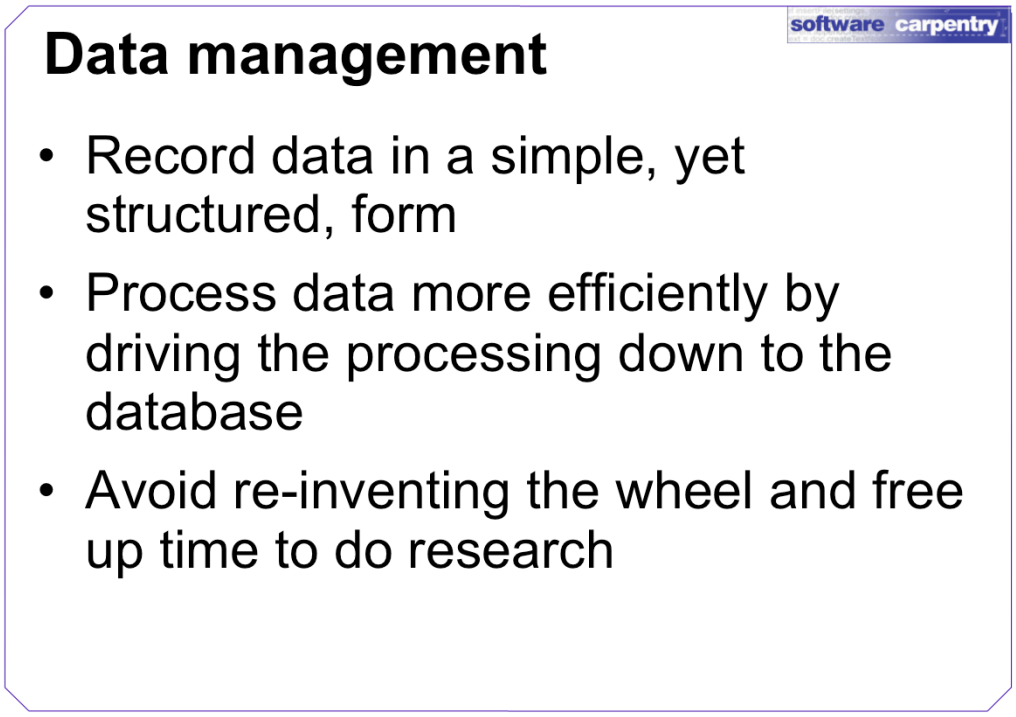
Meaningful names.

Indentation.

Unit testing

- Ensure that your algorithms, implementation and data are correct and so that your research is correct
- Make changes without introducing bugs
- Fix bugs without introducing bugs

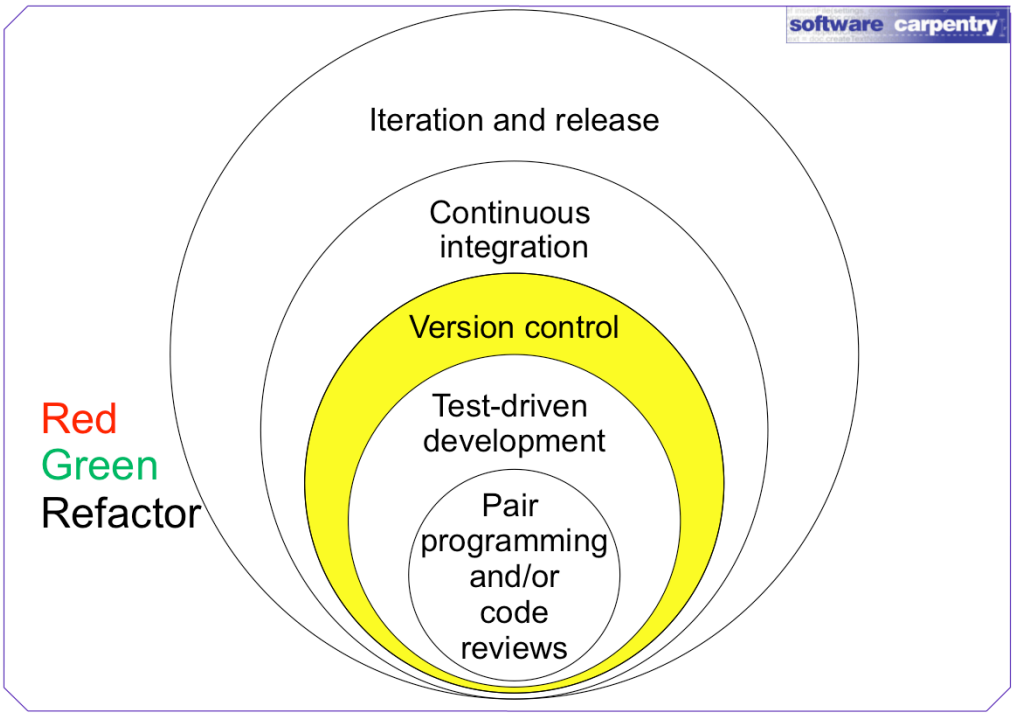
- Remember Geoffrey Chang!



Putting it all together

Databases are designed to be highly efficient and support highly powerful data access, filtering, and combination.

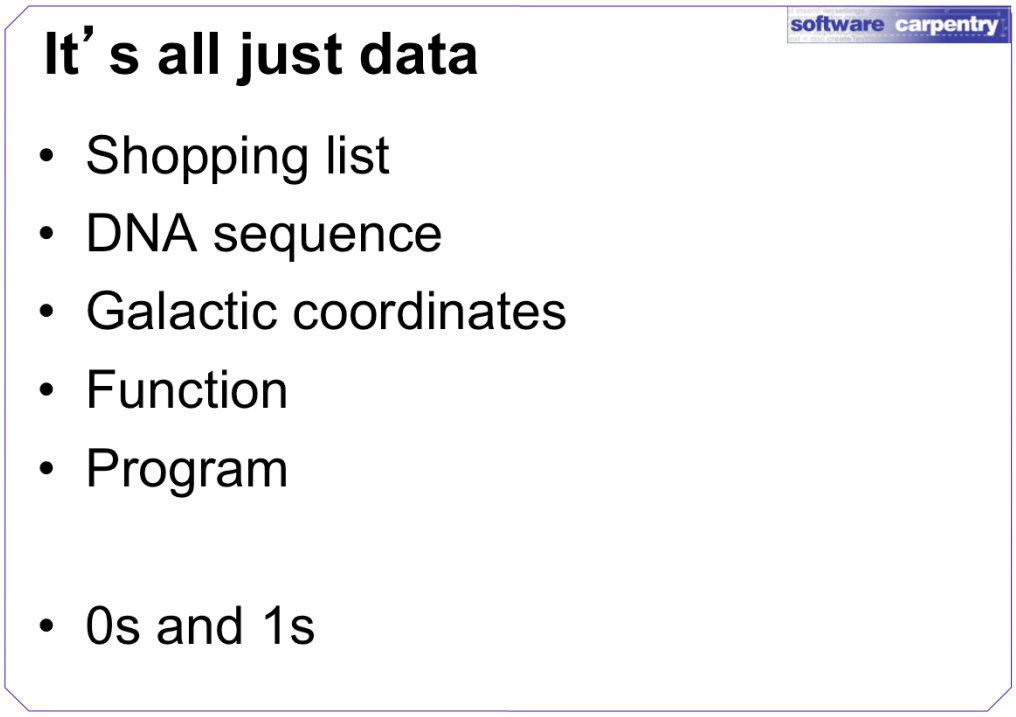
They've served their time in and been driven by the requirements of business!



Putting it all together

Pair programming. Driver and navigator. Swop after an hour. Peer review. Share ideas and expertise. Share knowledge of code. Collective ownership. Induct new team members. Less Facebook-ing!.
Code reviews. Share ideas and expertise. Fagan and Cohen.
TDD. Red (fail). Green (pass). Refactor (clean). Focus on what code should do rather than what it does. Code always tested. Promotes working in small steps.

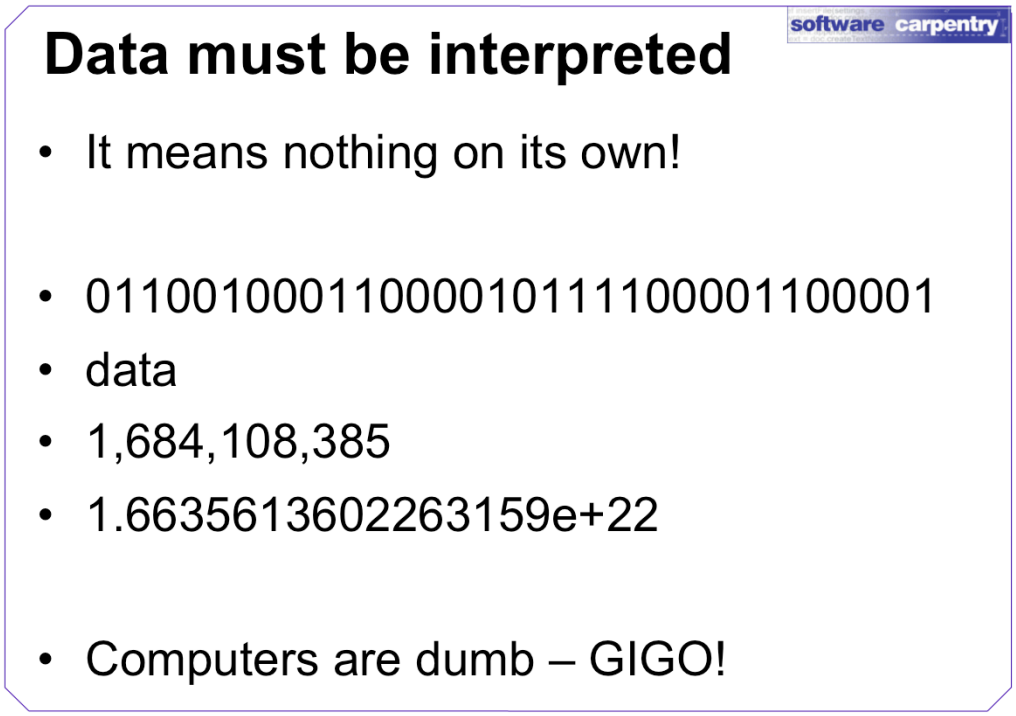
Version control. Provenance. Commit small, complete units of work, ideally a reviewable unit.
Continuous integration (CI). Checkout/build/test/publish. Fail-fast. Name and shame. Always runnable. Always releasable. On which note…
Release early, release often – so long as it's built and tested cf. MPA. Users often don't know what they want until they have something they can use. A tool can change how users do their work. Short iterations avoid spending too much time building what turns out to be the wrong thing. Well-suited to research software. Requires less coordination as people can track what they do for a few days themselves. Easier to find bugs – 2 weeks worth of code, not 2 months or 2 years worth. Always have something that is working, tested and ready for release. MAUS and bi-monthly releases.
Concentric circles focusing on continuous feedback, collaborative working and small development cycles. Agile development methods, frequent feedback for developer and customer. Not an excuse for cowboy coding, or no process. Even if you are a solo these still benefit you!



Putting it all together

Some principles of computational thinking….

Anything can be considered as data.
Functions can be assigned to variables in Python.
To a compiler, source code is input data and binary code is output data.



These 32 bits represent the word data, the integer and the float, as well as a bluish-gray pixel, a medium-loud sound sample, a machine code instruction, and many more things.

Computer does not know or care e.g. if a variable is called temperature or t or Frankenstein or abc.

The computer does what it is told.

software carpentry

Programming is about creating and composing abstractions

- Meaningful names
- Comprehensible units
- Separate…
  - "what I do" – interface
  - "how I do it" – implementation
- Clarity over cleverness

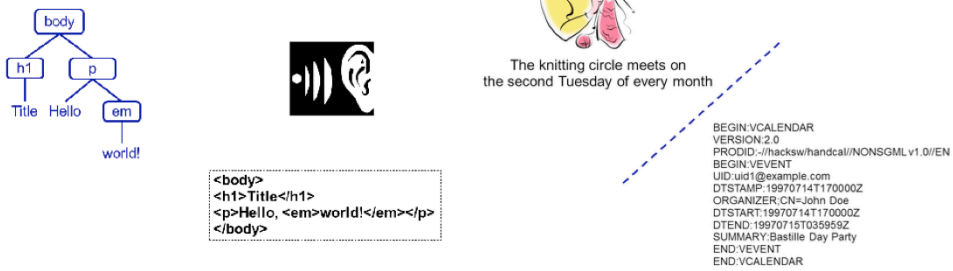
Putting it all together

To use things you shouldn't need to care about the implementation (unless you're concerned with efficiency, for example).

Brian Kernighan, "…debugging is twice as hard as writing a program in the first place. So if you are as clever as you can be when you write it, how will you ever debug it?"

Models are for computers

- …they're good at structured data so help them out!
- Views are for us…



Putting it all together

Structured data is better than unstructured data.

Models are representations that are easy for computers to work on.

Data are presentations of the model that we can understand.

HTML model and browser or audio-described views that make the content accessible.

Easy to go from model to view.

Very difficult to go from view to model e.g. parse audio, OCR of browser image.

Paranoia makes us productive

software carpentry

- Improve quality => improve productivity
- Think before you code
- Validate inputs wherever they originate
- Pair program and code review
- Automate your build-and-test

Putting it all together

Why? Because higher quality => less errors => less recovery!

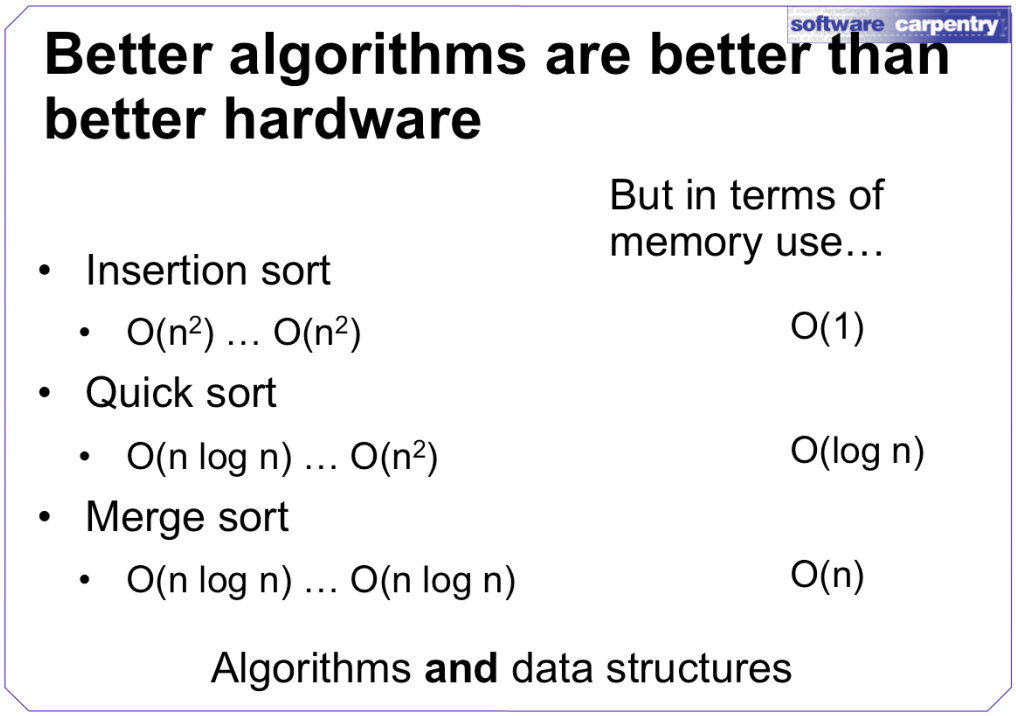
Think before writing code. What does “I want to count all the stars in this photograph” mean? Ask yourself What is a star? How many pixels does a star occupy?

Validate. Remember, GIGO. Devise data formats that support validation.

Automate. Continuous integration.

Reduce chances of getting it wrong next time.

Focus on things that can’t be automated until we write the code to automate them too!



Putting it all together

Algorithmic complexity. Estimate how many operations an algorithm will do, or how much memory it requires, as a function of the size of the problem we're trying to solve. Some algorithms slow down gently as their inputs get larger, while others slow down so much that even if the whole universe was one large computer, it couldn't solve any problem big enough to be interesting.

Big-O – worst-case order of growth (speed or space) as size of data set increases. Not efficiency but scalability.

O(1) – independent of data set size.

O(n) – directly proportional to data set size e.g. traverse an array is O(n)

O(n log n) – typical for data that is partitioned e.g. a tree, or quick and merge sorts. Data partitioned, partitions partitioned etc.

$O(n^2)$ – proportional to square of data set e.g. if algorithm processes every list element and processing requires another list traversal or other inner loops.

$O(2^n)$ – exponential growth. Growth doubles for each new element in the data set. Not very scalable without masses of hardware!

Scalability is worse as O increases.

In terms of execution time, merge sort win. In terms of memory insertion is 1, quick is log n and merge can be n.

Algorithms are nothing without data structures to operate on and the performance of the former often derives from the latter e.g. arrays/lists and loops, trees and recursion. Faster hardware helps, but efficient implementations help more! Trade-off of complexity versus

The tool shapes the hand

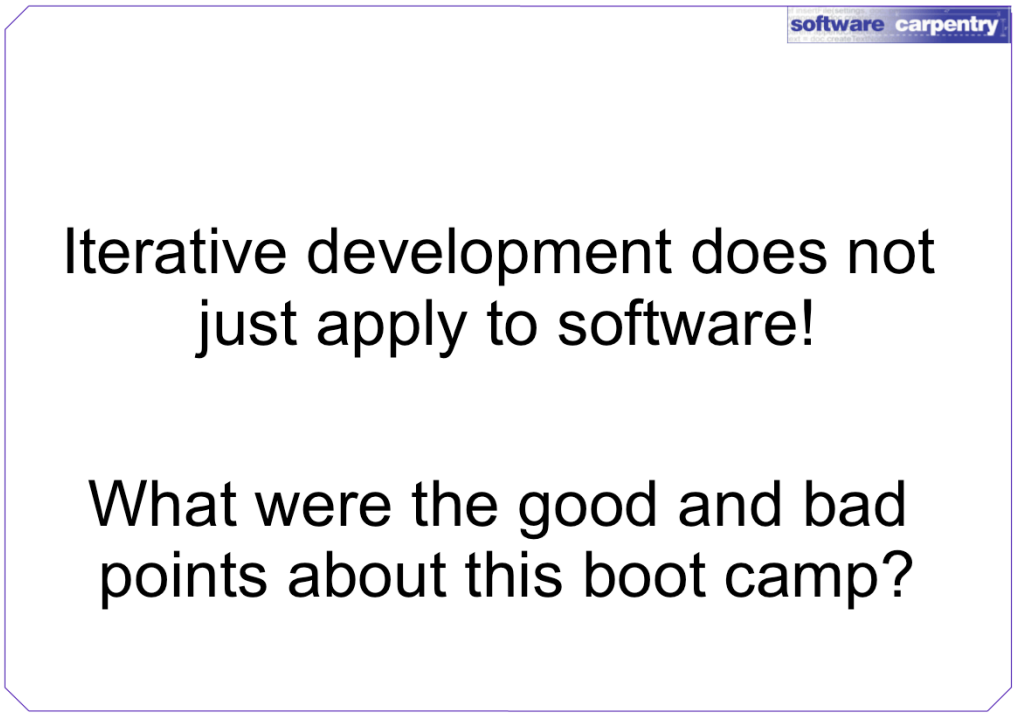
- Using a tool changes how you do the work the tool helps you to do
- Writing software changes your understanding of what computers/ software can do

All work and no play…

- “Why Crunch Modes Doesn't Work: Six Lessons”, Evan Robinson
  - http://www.igda.org/why-crunch-modes-doesnt-work-six-lessons
- Continuous work reduces cognitive function 25% for every 24 hours!
- Our greatest productivity occurs in the first four to six hours

Take regular breaks away from the screen. Go for a walk. Avoid RSI!

And, on that note…



Just ask Ridley Scott after he releases Blade Runner version 4 or Prometheus version 2!
Get a helper to scribe.
Go round each attendee in turn – one suggests a good point, the next a bad point etc.

Want to get involved?

- Write online lectures and practicals
  - Scientific python libraries e.g. PANDAS or SciPy image processing
  - Version control in Git
  - Something of use to your discipline
- Want to be a helper…
  - …or instructor?
  - Edinburgh, December 4-5th, 2012
  - Munich and Tuebingen, January 22-23rd, 2013
  - Kiel, March 2013

Putting it all together

Looks good on a CV!

A chance to get teaching experience, a networking opportunity, trips to nice places!

Edinburgh can cover travel expenses.

And there are other boot camps always coming on-stream.

Want to get involved?

- North American 3 week teaching tour
  - MIT and Harvard => across the US => Stanford and Monterey
  - Travel and accommodation covered
  - Currently seeking funding
- Get in touch!
  - info@software.ac.uk
  - info@software-carpentry.org
  - Speak to Steve or Mike

Putting it all together

Should funding go through the tour would be open to those who have contributed to Software Carpentry (see previous slide) and want to get more experience as instructors.

We can add you to the UK software carpentry list so you're aware of helper/instructor opportunities – no obligation!