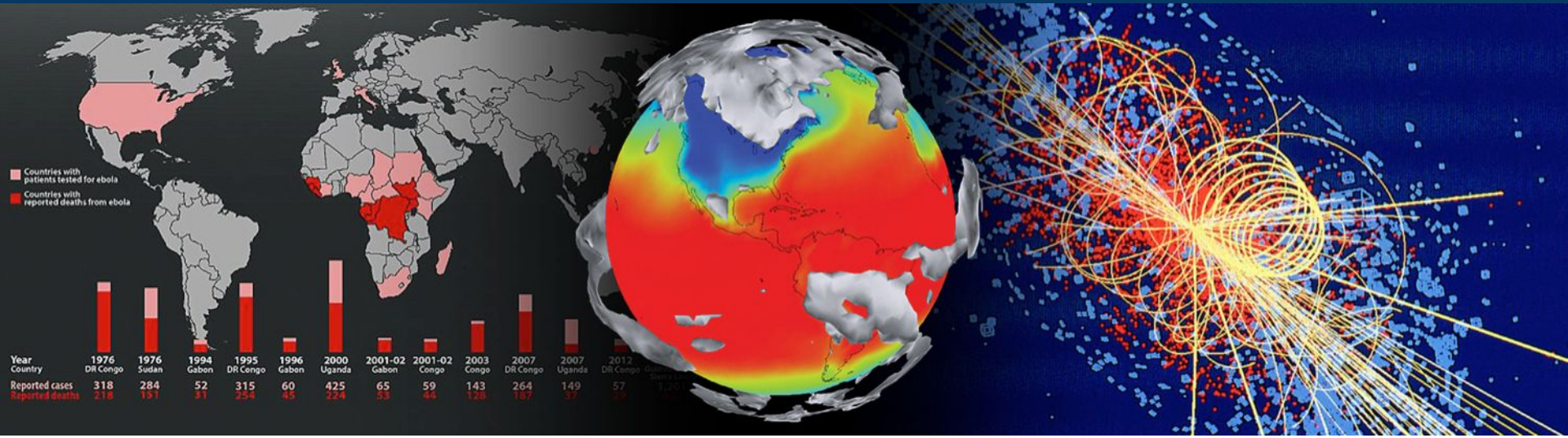
EPSRC and MRC Systems Approaches to Biomedical Science CDT

Introduction to Software Engineering Concepts

Steve Crouch
Software Sustainability Institute
s.crouch@software.ac.uk

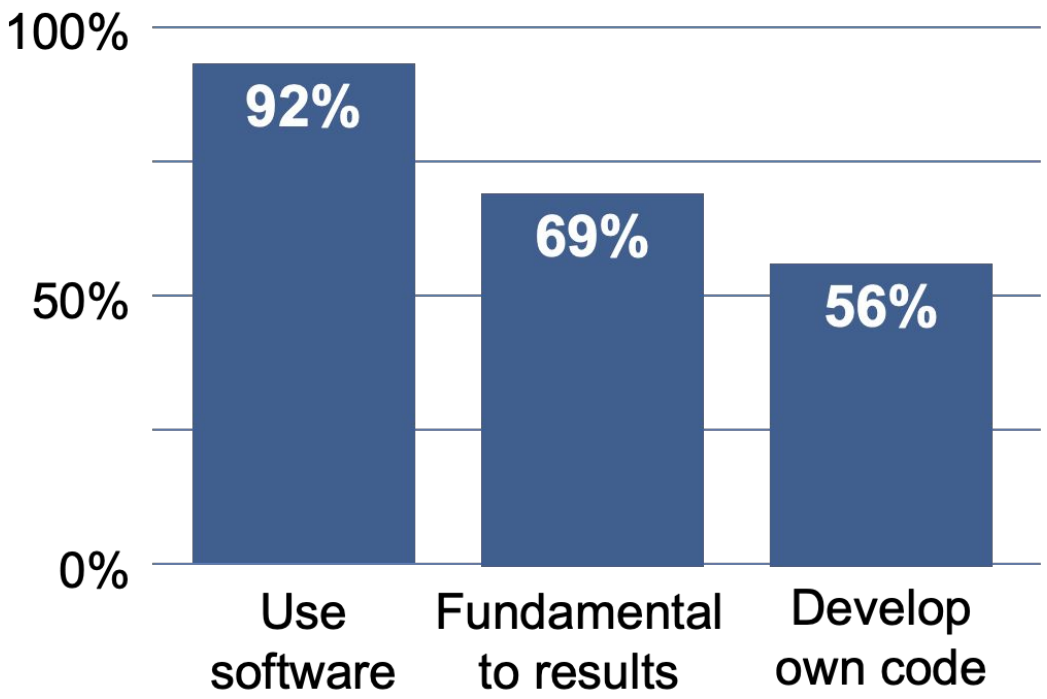
12th October 2020

Modern research is impossible without software



From thrown-together scripts, through an abundance of complex spreadsheets, to the millions of lines of code behind large-scale infrastructure, there are few areas where software does not play a fundamental part in research

Why should we care about software?



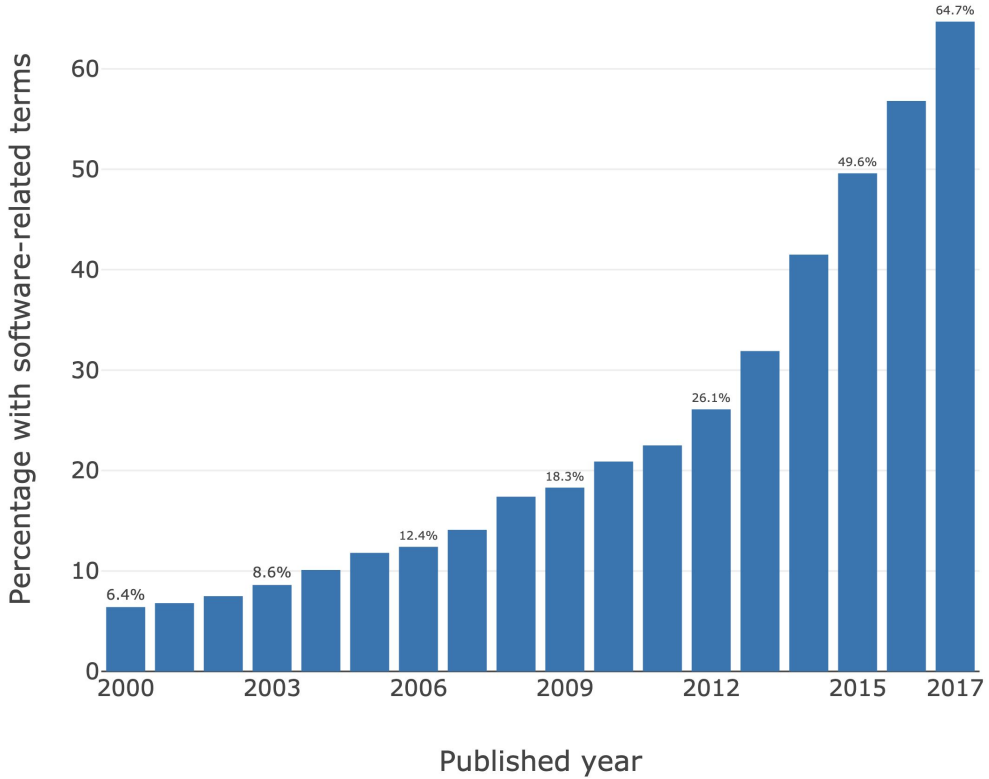
SSI survey of researchers, 2014[1]

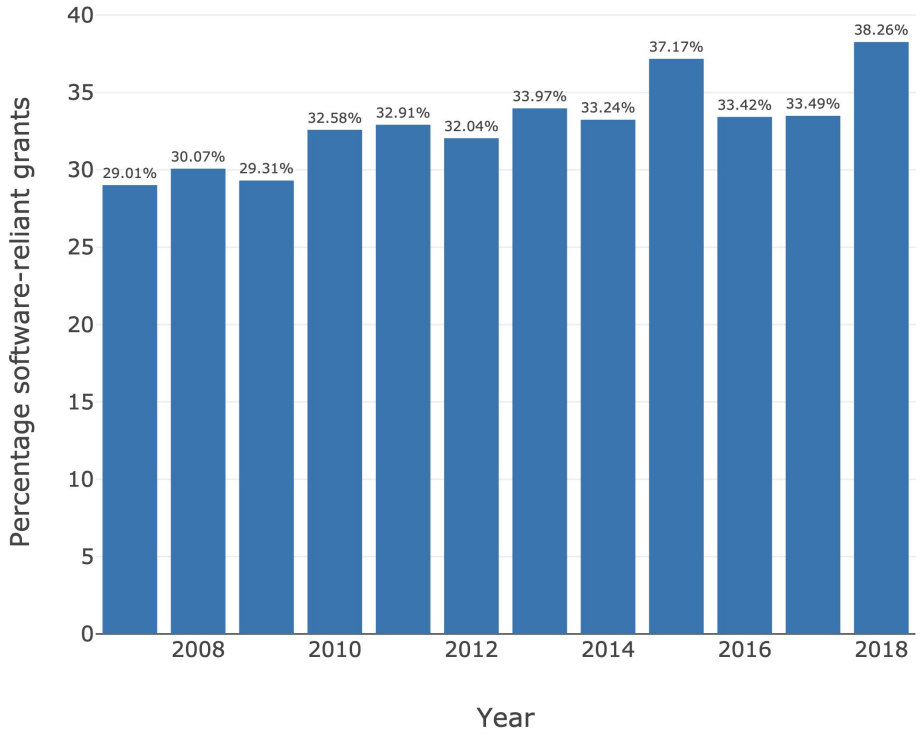
15 Russell Group Universities

Their software use and background

417 respondents

Why should we care about software?

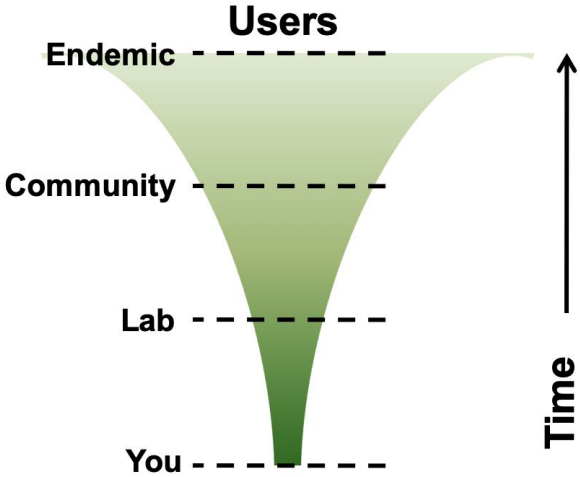




[2]

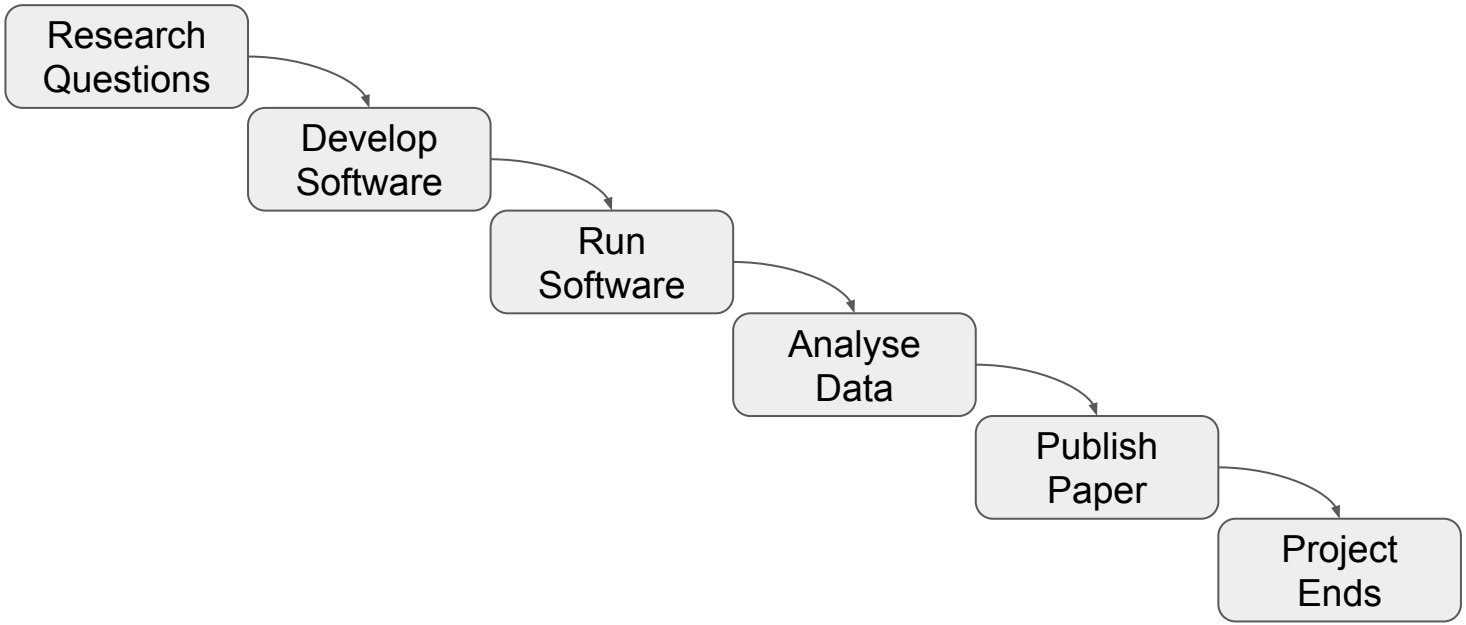
The software you write is important!

- Software inherently contains *value*
  - Produces results, contains lessons learnt, effort
- Difficult to gauge to what extent it might be used in the future
  - By who?
  - Which parts?
  - Which projects?
  - Reproducibility – from publications!

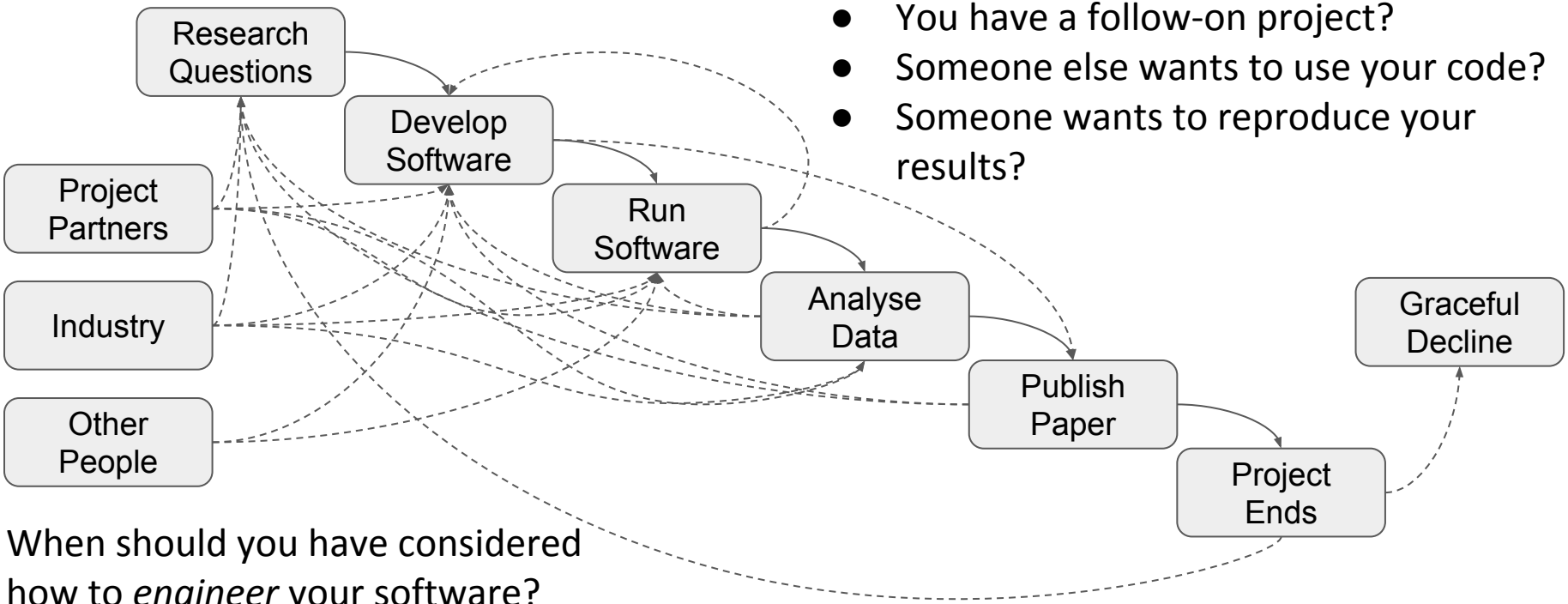


Can it/should it be reusable by others?
...including yourself?

A typical research software lifecycle



In reality...



What happens when…

- You have a follow-on project?
- Someone else wants to use your code?
- Someone wants to reproduce your results?

When should you have considered how to *engineer* your software?

"The best time to plant a tree is 20 years ago.

The second best time is now."

What could go wrong?

- Ariane 5
- $7B
  $500
- Used Ari

EXCEPTION HANDLER DISABLED

BOOM!

de info → **64-bit FP converted to 16-bit signed integer**

Programming vs Engineering

Programming / Coding

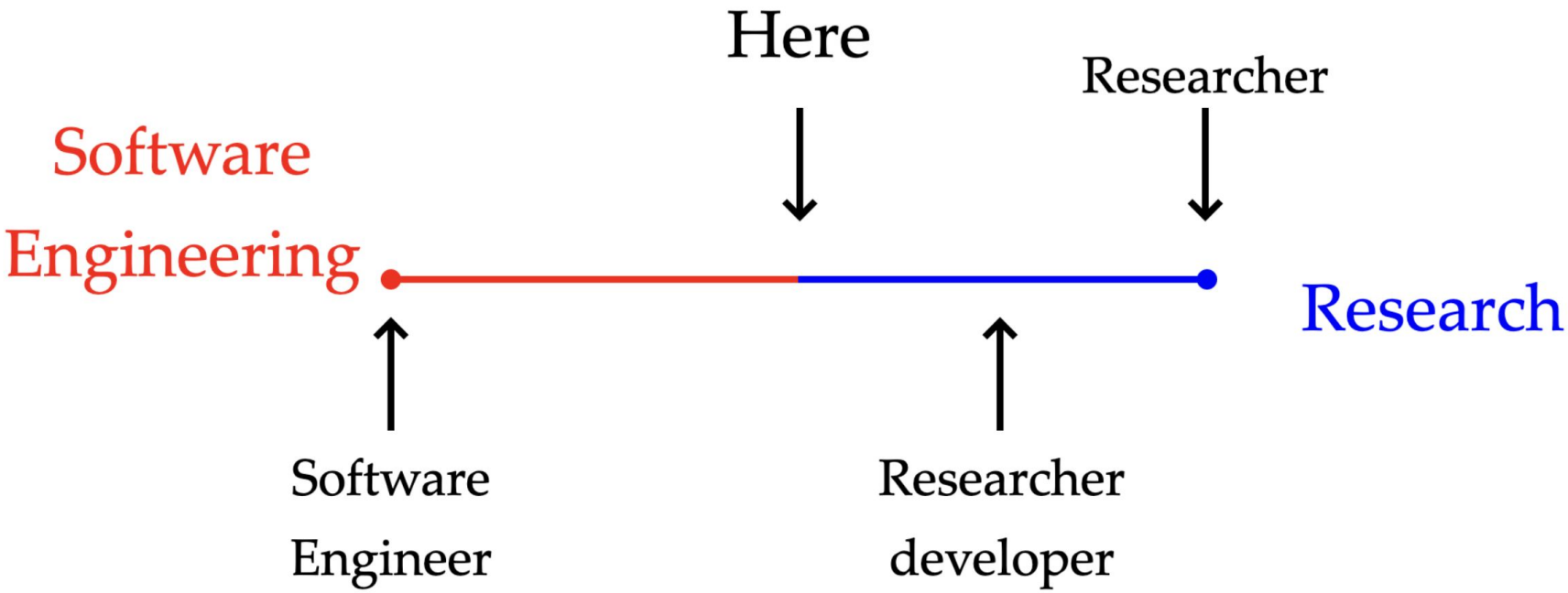
- Focus is on one aspect of software development
- Writes software for themselves
- Mostly an individual activity
- Writes software to fulfil research goals (ideally from a design)

Engineering

- Considers the *lifecycle* of software
- Writes software for *stakeholders*
- Takes *team ethic* into account
- Applies a *process* to understanding, designing, building, releasing, and maintaining software

"Programmers tend to start coding right away. Sometimes this works."
- Eric Larsen, 2018

Where are you?



Beyond building a 'sequence of instructions'

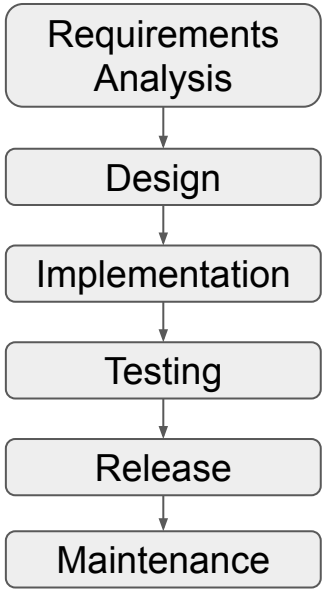
Software is far more than that…

- **Outcome of a *development process***

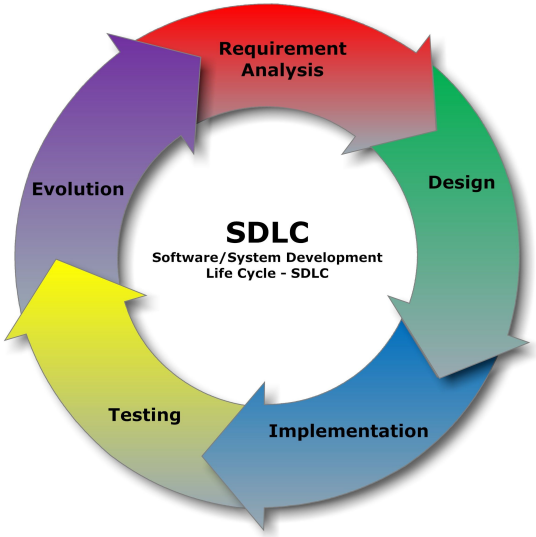
But also...

- Architecture
- Implementation of algorithms
- Data model
- Programming paradigm
- Documentation
- *Best practices* and *conventions ...*

Waterfall Model



Agile Model

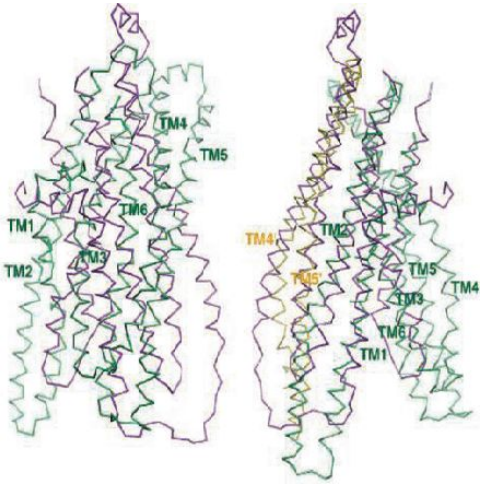


Testing

- Humans are fallible! Our software will contain defects
  - In requirements, design, as well as code
  - 1-10-150 hours to fix in design/development/production

- Validation: are we building the right product?
- Verification: are we building the product right?
  - Manual testing, unit testing, automated testing, code reviews

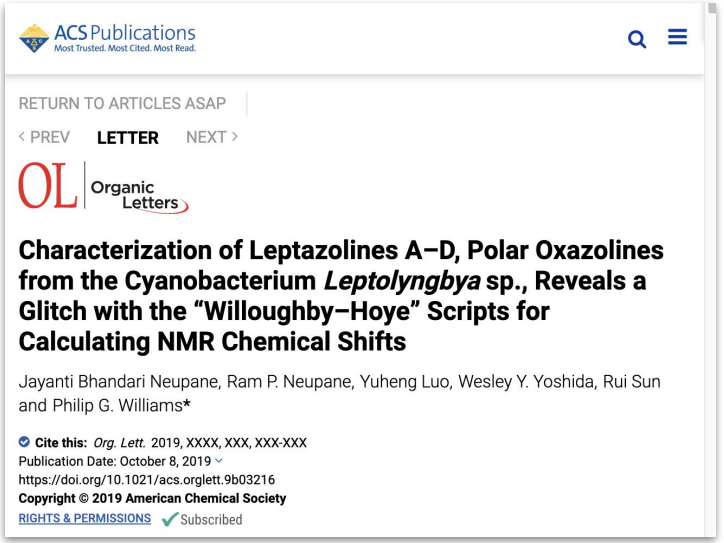
- Highly-cited papers published on multidrug resistance transporters between 2001 - 2010
- Results couldn't be reproduced - 5 retractions
- Caused by error in an *internal software utility*
  - Flipped two columns of data, inverting electron-density map used to derive protein structure



"I didn't question it then. Obviously now I check it all the time."
- Geoffrey Chang[3]

Platform support?

… Density functional theory nuclear magnetic resonance calculations established the relative configurations of 1 and 2 and revealed that **the calculated shifts depended on the operating system when using the "Willoughby–Hoye" Python scripts to streamline the processing of the output files, a previously unrecognized flaw that could lead to incorrect conclusions**.



- Due to *different sorting of file names* on different operating systems

Organic Letters, October 8 2019
https://doi.org/10.1021/acs.orglett.9b03216

Optimisation

*"Three orders of magnitude in **machine speed** and three orders of magnitude in **algorithmic speed** add up to six orders of magnitude in solving power. A model that might have taken a year to solve 10 years ago can now solve in less than 30 seconds."*

– Robert Bixby, review of linear programming solvers from 1987-2002

- Faster code, faster results!
- Understanding trade-offs
  - Maintainability, accuracy
- When & where to optimise?
  - 80/20 rule, code profiling

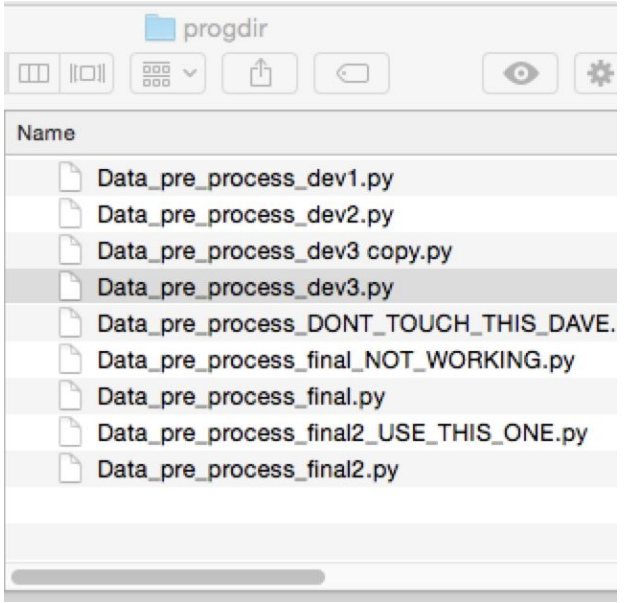
Amdahl's Law:

Time to result = Develop Time (D) + Time to Run (R)

As effort is put into reducing R, overall time required to get new result is dominated by writing, testing, maintaining, installing, configuring software.

Code management & collaboration

- *Version control* provides a full history of your project's software and other assets
- Makes for easy:
  - Backups
  - Collaboration
  - Recovering from dead-ends
- What should be in version control?
  - Code, documentation, tests, test data, analysis scripts
  - Reports, papers, etc.
- Packaging and deployment



"If you're not using version control, whatever else you might be doing with a computer, it's not science."

- Greg Wilson, SWC

Other key points

- These skills will **save you time**
- Always assume others will use and develop your software
- Be clear on requirements and assume they will change
- Funders are increasingly expecting software outputs to be sustainable and reusable

More on software engineering

Robert L Glass, Addison-Wesley Professional

Group mini-project

For week 2, in same groups as those for Zoom...
Design and implement Python library to specify & solve a Pharmokinetic model, which should ideally have the following functionality:

- pip installable
- github repository, with issues + PRs that fully document development process
- unit testing with good test coverage
- fully documented, e.g. README, API documentation, OS license
- continuous integration for automated testing/doc generation
- Ability to specify form of the PK model
- Users can specify protocol independently from the model
- Ability to solve for drug quantity in each compartment over time
- Ability to visualise the solution of a model, compare two different solutions
- Something else? Feel free to suggest alternative features!

Group projects: SABS students

1. Extensible Clinical Imaging QC Tool [GE Healthcare]
    - Existing tools are typically proprietary or focussed on specific modality/therapeutic area
    - Envision in-stream device to receive DICOM data & accept/reject based on configurable criteria
    - Develop open, extensible software for checking, updating, tracking schema-d metadata

2. Automated Data Extraction to Future-Proof Therapeutic and Natural Antibody Databases [UCB, Roche, GSK]
    - Antibody informatics databases manually curated & can't harvest literature/other sources automatically
    - Goal to increase usability and automation of their updating systems, possibly leveraging NLP
    - Develop frameworks to maximally automate updating process for each database

Group projects: SABS students

3. Development of Open Source Epidemiology Modelling Software [Roche]
    - COVID-19 has highlighted crucial role and pitfalls of math/computational modelling in epidemiology
    - Goal to develop robust software platform for modelling of infectious disease transmission
    - A pedagogical tool for universities, schools, industry to illustrate importance of robust SE principles
    - Ultimate goal to reproduce robust, sustainable versions of key current codes used to model pandemic

4. "Drug Discovery Game" SABS R^3 software project [Roche]
    - Develop game software to help learn how medicinal chemists make compound progression decisions
    - Inspired by "Drug Discovery Game", similar to Mastermind game; successful guesses cost play money
    - Medicinal variant: basis of possible substituents/functional groups attached to molecular scaffold

Group project: DTP, NERC students

Tree Generation Web Application

- Apply software engineering best practices, packaging, testing, CI skills to new tech domain
- Develop a single-page web application using JavaScript, Vue.js and Nuxt framework
- Goal to allow users to generate visualise branching tree structures using L-Systems
- To be deployed to a static web-page hosted on GitHub pages

General daily teaching structure

09:30-10:00 Welcome and Q&A	***Main Zoom room***
10:00-12:30 Self-learning session 1	***Pre-assigned groups, separate Zoom room***
12:30-13:30 Lunch (advisory only!)	***Advisory (instructors' may be unavailable)***
13:30-14:00 Q&A session	***Main Zoom room***
14:00-16:30 Self-learning session 2	***Pre-assigned groups, separate Zoom room***
16:30 Finish	***Advisory (can keep working if you like)***

Each Zoom room will have 'roving' demonstrators
Also - there will be a "common room" for breaks!

A few infrastructure things...

- Ensure you have a name set in the *Participants* list
  - *Participants* -> hover over your entry, select *More* then *Rename*
- Please mute when not talking
- Need help?
  - In first instance, ask demonstrators for help by raising hand in Zoom room
    - Particularly if there are a lot of people asking questions - demonstrators can answer them in order
    - Or if quiet, just ask them (or your Zoom buddies!)
  - If demonstrator not in room, ask on Slack
- Move Zoom rooms (i.e. to Common Room) at break times if you like)
- Be sure to tick off the exercises you complete in Canvas as you go!

Need help?

Say hi to your Zoom demonstrators & neighbours!

References

[1] "It's impossible to conduct research without software, say 7 out of 10 UK researchers”, http://www.software.ac.uk/blog/2014-12-04-its-impossible-conduct-research-without-software-say-7-out-10-uk-researchers

[2] "An investigation of the funding invested into software-reliant research", https://github.com/softwaresaved/software_in_grants_GTR

[3] "Retractions unsettle structural bio", https://www.the-scientist.com/daily-news/retractions-unsettle-structural-bio-46891