If You Can’t Reproduce It,
Is It Still Science?

And how long will it take?

Paul Wilson

Inspired by Greg Wilson
Software Carpentry

- Write software for people, not computers
- Automate repetitive tasks
- Use the Computer to Record History
- Make Incremental Changes
- Use Version Control
- Don’t Repeat Yourself
- Plan for Mistakes
- First make it correct, then make it fast
- Document Design & Purpose
- Conduct Code Reviews

Write Programs for People, Not Computers

- Most researchers will spend more time reading code than writing code
  - It's the primary way to learn what it does and how
- Recognize realities of human cognition
  - Working memory is limited
  - Pattern matching abilities are finely tuned
  - Attention span is short

Automate Repetitive Tasks

- This is why we invented computers!!
  - It’s not why we invented graduate students
- Saves time & avoids errors
- Can track dependencies
- Unambiguous record of workflow
- Motivates command-line interfaces

Use the Computer to Record History

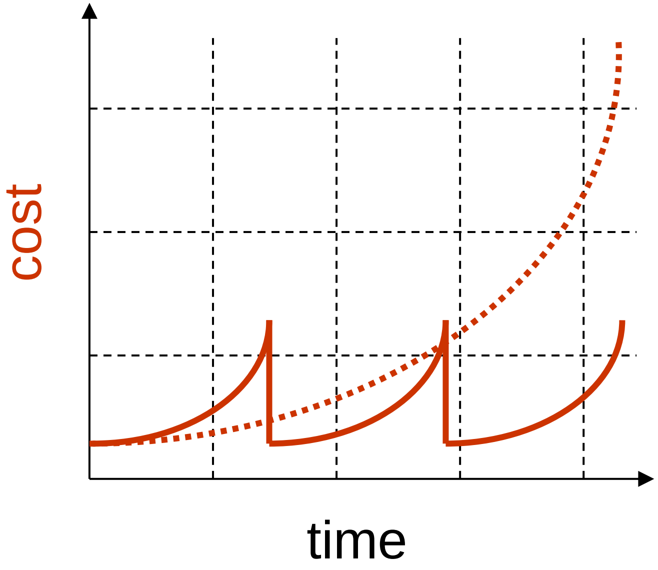
- Careful record keeping is fundamental to science
  - A manual log book works for experiments occurring at a “traditional” pace
  - What happens when you can perform 100 experiments/day? 1000? 10,000?

Use the Computer to Record History

- Use software tools to track computational work
  - Unique identifiers/versions for data
  - Unique identifiers/versions for software
  - All input parameters
- Embed this information in output

Make Incremental Changes

- Long development cycles have many disadvantages
  - Human attention span
  - Delayed identification of bugs
  - Adapt to changes in requirements
- “Agile” development



Use Version Control

- Two big challenges
  - Tracking all the changes to code over time
  - Synchronizing changes during collaboration
- Bleeds back to provenance
  - How do you know exactly which version you used?

Use Version Control

- Ad-hoc solutions:
  - Make separate copies for different versions
  - Dropbox, email for sharing
- All subject to human error
- Why not “Use the Computer to Record [this] History”, too?

Use Version Control

- A great big “undo” button
- Focus on changes

Don’t Repeat Yourself (or Others)

- Anything repeated in 2 or more places is difficult to maintain
  - Increases chance of errors and inconsistencies
- Modularize the code you write
- Don’t reinvent the wheel

Plan for Mistakes

- Bugs are guaranteed!
- Finding bugs is hard!
- No single practice will catch all defects – use in combination
  - Defensive programming
  - Testing
  - Debuggers

Optimize Software Only After it Works Correctly

- Correct is more important than fast
- Complexity of modern hardware & software make it difficult to predict bottlenecks
- Profile and test performance after it works to identify need for improvement

Optimize Software Only After it Works Correctly

- Corollary: Use high level languages!
- Fixed: number of lines of code per day, independent of the language
- Get more done with high-level languages, even if slower
- Profile, measure and improve

Document the Design and Purpose of Code Rather than its Mechanics

- Most research software will be handed off at least once
  - Large cost for “forensic” analysis
- Documentation is critical
  - … but only if it’s good documentation

Document the Design and Purpose of Code Rather than its Mechanics

- Document interfaces
  - How to use something
  - What behavior to expect & why
- Do not document implementation
  - Well-written implementation should be self explanatory
  - If not, refactor it until it is
  - May need to document reasons for specific implementation decision

Conduct Code Reviews

- Peer review is a cornerstone of modern research
  - Reduces errors
  - Improves communication/ understandability
- Why review publications based on software and not the software itself?

Combining Best Practices

- Continuous Integration
  - Automatically rebuild and retest every time a test is made
    - Automation of repetitive task
    - Supports agile development
    - Relies on testing