software carpentry

Basic Software Skills for Scientists and Engineers

Greg Wilson
gvwilson@cs.utoronto.ca

Double Standards

- Experimentalists check the purity of samples, calibrate equipment, and write everything down
  - That's how you know they're scientists
- Computationalists...don't
- *Most computational results are of unknown quality, and irreproducible*

Inefficiency

- Many scientists spend much of their lives developing software
- Few do this productively
  - They've never been taught how
- Half a dozen tools and techniques can improve productivity by 20-30%
  - That's one less semester in grad school

Amdahl's Law

- Let:
  - t_1 be a program's runtime on one CPU
  - t_p be its runtime on P CPUs
  - β be the algorithm's serial fraction

$$t_p = \beta t_1 + (1 - \beta) t_1 / p$$

$$s_\infty = 1/\beta$$

Wilson's Amendment

- But software doesn't "just happen"
  - Let T and S be the program lifetime equivalents of t and s
  - Let D be development time

$$T_p = \beta T_1 + (1 - \beta)T_1/p + D$$

$$s_\infty = 1/(\beta + D/T_1)$$

Therefore

Development practices affect time to result as much as algorithms and hardware.

…But Nobody Wants to Die

- "It takes me a week just to fall further behind."
- "I'll learn how to do your job when you learn some field theory."
- "God wishes he was a better mathematician."
- "I don't have to outrun the bear."

Twenty Questions

- What are your changes of building something that works, *without* heroic effort?
- Give yourself:
  - +1 for "yes"
  - 0 for "no" or "not applicable"
  - -1 if you don't know

…Twenty Questions

1. Does your project have an elevator pitch?
2. Do you use version control?
3. Can you rebuild everything with one command?
4. Do you build the software from scratch daily?
5. Do you have an automated test suite?
6. Do you run it before checking in?
7. Do you know how much code your tests cover?

…Twenty Questions

8. Do you have a bug database?
9. Do you use a symbolic debugger?
10. Do you use assertions and other defensive programming techniques?
11. Can you trace everything you release (not just software) back to its origins?
12. Do you document as you program?
13. Do you keep your documentation in your source files?

…Twenty Questions

14. Can you set up a development environment without heroic effort?
15. Do you have a schedule with small binary milestones?
16. Do you estimate how long tasks will take before you start, and compare that with how long they actually took?
17. Is there a searchable archive of discussions about the project?

…Twenty Questions

18. Do you use a style checker to ensure your code is written in a uniform, readable way?
19. Do you write small tools for automating common tasks?
20. Does your schedule allow for infrastructure development, training, sick time, etc.?

And Your Score Is?

Negative:	Low probability of success
0-5:	Everything takes longer and costs more
6-10:	Able to bootstrap on your own
11+:	You should be up here talking…

Let's Start Over…

- "Good programmers are 20 times more productive than bad ones."
- But many of them were self-taught
  - Studies were *actually* measuring cultural differences
- A few basic skills (and tools) can make a dramatic difference

Let the Bandwagons Roll

- Formalism! Agility! Formalism! Agility!
  - Both are improvements in practice
- Either:
  - The way most programmers do things is the worst way possible
  - Thinking about process is what makes the difference
  - The "common core" is what matters

What We Actually Know

- Focusing on quality reduces total development time
  - No, your case is *not* special
- Quality is:
  - Designed in
  - Monitored
  - Maintained
  - Through the whole software lifecycle

My Claim

The things that will turn computational science into real science will also make computational scientists more productive.

There's More Than One Way

- Every good process is quality-oriented
  - Sturdy ones use up-front design to prevent mistakes
  - Agile ones rely on lots of short cycles to permit early correction
- Which works best for you depends on your problem domain
  - And personality

The Core

- Thinking before coding
- Version control
- Automation
- Test-driven development
- Issue tracking
- Debugging aids
- Enforcing Style

Thinking Before Coding

- "A week of hard work can sometimes save you an hour of thought."
- Public description and discussion:
  - Stops you from building the wrong thing
  - Uncovers cognitive dissonance
  - Helps the next person get up to speed
- Primarily a social skill
  - Technology can only be an aid

Version Control

- A great big "undo" button
- Unavoidable coordination
  - Highlights changes
  - Manages collisions
  - Supports independent development
- A key predictor of success
  - Teams that *don't* use version control usually fail

Automation

- "Anything worth doing again is worth automating."
  - make; make test; make install
- Automate as much testing as possible
  - Everyone runs tests before checking in
  - Build and test code continuously
- Lots of little tools are a sign of a healthy project

Test-Driven Development

- Write the test, then write the code
  - Not as new an idea as XP's advocates would have you believe
- Biggest benefit is that it forces you to create testable code
- Also forces you to be explicit, early, about what "correct" actually means

Issue Tracking

- The project's collective to-do list
  - *Not* just for bugs
- Version control's forward-looking counterpart
- The primary reality check on planning and scheduling
  - "A schedule's purpose is to tell you when to start cutting corners."

Debugging Aids

- Interactive debuggers dramatically increase productivity
  - Edit/compile/run/page is a stupid waste of 50% of your life
- Systems that don't support interactive debugging are much harder to use
  - The real Grand Challenge of our times

Enforcing Style

- Your brain thinks that differences must be significant
  - And 7±2 is built in
- Any convention is better than none
  - Use tools to check conformance
- Today's tools can also find memory leaks, race conditions, etc.

Conclusion

- Productivity is a personal reason to improve the way we work
- But there are collective reasons:
  - Reasonable confidence in results
  - Possibility of verification
- The only question is whether we will jump or be pushed

Further Reading

- Doar: *Practical Development Environments*
- Feathers: *Working Effectively with Legacy Code*
- Fogel: *Producing Open Source Software*
- Glass: *Facts & Fallacies of Software Engineering*
- Hunt & Thomas: *The Pragmatic Programmer*
- Spinellis: *Code Reading*
- Zeller: *Why Programs Fail*