Essential Software Skills
For Research Scientists

Dr. Gregory V. Wilson
gywilson@cs.utoronto.ca
“Software engineering for science has to address three fundamental issues: (i) dealing with datasets that are large in size, number, and variations; (ii) construction of new algorithms to perform novel analyses and syntheses; and (iii) sharing of assets across wide and diverse communities.”

— Emmott et al, *Towards 2020 Science*

● It’s a shame “getting the right answer” didn’t make the list...
The State of Play

- Most computational “science” isn’t
  - Not reproducible
  - Of unknown quality
- Most scientists don’t care
  - Because journal reviewers and tenure committees don’t
- Is “computational thalidomide” inevitable?

2006-09-06  http://www.swc.scipy.org  00010
Many scientists spend most of their time developing, maintaining, or running software

- But few have ever been taught how to do this efficiently

Most computational results take longer to produce than they need to
Productivity? Tell Me More...

• The best way to improve productivity is to improve quality
  - RUP: prevent bugs through over-design
  - XP: catch bugs right after writing them

• Will scientists adopt quality if it’s disguised as productivity?
Self-Assessment

- What are your chances 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
...Self-Assessment

1. Do you use version control?
2. Can you rebuild with one command?
3. Do you write your tests before your code?
4. Do you run your tests before checking in?
5. Do you know how much of your code they cover?
6. Do you have a bug database?
7. Do you use assertions and other defensive programming techniques?
8. Do you use a symbolic debugger?
Can you trace everything you release back to its origins?
Do you document as you program?
Can you set up a development environment without heroic effort?
Is there a searchable archive of discussions about the project?
Do you use a style checker to maintain code quality?
Do you write small tools to automate recurring tasks?

2006-09-06  http://www.swc.scipy.org  00111
And Your Score Is?

Negative: Whatever you’re doing, it isn’t science
0-5: Low probability of success
6-10: Able to bootstrap on your own
11+: You should be up here talking...
Our Claim

The things that will turn computational science into real science will also make computational scientists more productive.
Amdahl’s Law

Let:
- $t_1$ be a program’s runtime on one CPU
- $t_p$ be its runtime on $P$ CPUs
- $\beta$ 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

\[
T_p = \beta T_1 + (1 - \beta) T_1 / p + D \\
s_\infty = 1 / (\beta + D/T_1)
\]
Why We’re Here

- Which development practices are most relevant to scientists?
- How much benefit is it reasonable to expect?
  - What’s the likelihood of success?
  - What will adoption cost?
- What obstacles must be overcome?
  - How best to overcome them?

2006-09-06  http://www.swc.scipy.org  01100
The Core

- Version control
- Test-driven development
- Continuous integration
- Issue tracking
- Debugging aids
- Enforcing style
- Traceability
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
Test-Driven Development

- Write the test, then write the code
  - Not as new an idea as XP’s advocates would have you believe
- Forces you to create testable code
  - And to be explicit about what “correct” actually means
- Provably leads to more reliable code per unit of effort

2006-09-06  http://www.swc.scipy.org  01111
Grand Challenges

- No one knows how to write unit tests for floating-point code
  - “Relative error < 10^{-6}” isn’t science
- Full analysis impractical
- The real Grand Challenge facing computational science
  - Along with getting people to care...
Continuous Integration

- Automatically rebuild and retest every time a change is made
- Early feedback keeps costs low
  - And ensures everything *is* checked in

![Graph showing cost increasing over time](http://www.swc.scipy.org)
Issue Tracking

- The project’s collective to-do list
  - Version control’s forward-looking counterpart
  - The primary reality check on planning and scheduling
- *Not* just for bugs
  - “Are we done yet?”
### Ticket Query

Filter:  
- Status: [ ] Open, [ ] Completed, [ ] Reopened, [ ] Rejected  
- Owner: [ ] In, [ ] gwilson

Group results by: [ ] Milestone, [ ] Descending, [ ] Show full description under each result

#### someday

<table>
<thead>
<tr>
<th>Ticket</th>
<th>Summary</th>
<th>Priority</th>
<th>Reporter</th>
<th>Type</th>
<th>Created</th>
<th>Last_modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>fix sizes of graphic images</td>
<td>high</td>
<td>gwilson</td>
<td>enhancement</td>
<td>2005-01-30</td>
<td>2005-01-30</td>
</tr>
<tr>
<td>96</td>
<td>Make sure all Python code conforms to Python style guidelines</td>
<td>medium</td>
<td>gwilson</td>
<td>defect</td>
<td>2006-02-01</td>
<td>2006-02-01</td>
</tr>
<tr>
<td>97</td>
<td>Check nifty.stanford.edu for ideas for Python projects</td>
<td>medium</td>
<td>gwilson</td>
<td>enhancement</td>
<td>2006-02-03</td>
<td>2006-02-03</td>
</tr>
<tr>
<td>98</td>
<td>Get PDF of crunch mode paper into repository</td>
<td>medium</td>
<td>gwilson</td>
<td>enhancement</td>
<td>2005-02-04</td>
<td>2005-02-04</td>
</tr>
<tr>
<td>100</td>
<td>Implement 'head' and 'tail' filters in inclusions</td>
<td>medium</td>
<td>gwilson</td>
<td>enhancement</td>
<td>2005-02-04</td>
<td>2005-02-04</td>
</tr>
<tr>
<td>1</td>
<td>readstyle - Describe PyDoc</td>
<td>medium</td>
<td>sdawson</td>
<td>defect</td>
<td>2005-01-29</td>
<td>2005-01-29</td>
</tr>
<tr>
<td>3</td>
<td>version - More focus on revision history as motivation for Subversion</td>
<td>medium</td>
<td>sdawson</td>
<td>defect</td>
<td>2006-01-29</td>
<td>2006-01-29</td>
</tr>
<tr>
<td>4</td>
<td>sql - Rework SQL examples to be scientific data</td>
<td>medium</td>
<td>sdawson</td>
<td>defect</td>
<td>2006-01-29</td>
<td>2006-01-29</td>
</tr>
</tbody>
</table>

2006-09-06  http://www.swc.scipy.org  10011
Debuggers

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

- Your brain thinks that differences must be significant
  - And $7 \pm 2$ 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.
Traceability

- Programs (and data) should record their lineage
  - Version numbers for source files
  - Processing steps (and settings) for data
- Greatly reduces management and maintenance costs
  - And is very comforting in a crisis

2006-09-06  http://www.swc.scipy.org  10111
Automation

- Theme running through all of these is automation
  - “Anything worth doing again is worth automating.”

- Lots of little tools are a sign of a healthy project
  - Have to be designed and maintained like any other piece of software
How To Get There

- **Educational carrots**
  - A one-term grad course reduces time spent wrestling with software by 20%
  - While improving quality

- **Institutional sticks**
  - Question the trustworthiness and reproducibility of computational results
  - And the cost-effectiveness of their producers

2006-09-06  http://www.swc.scipy.org  11001
Questions?

FROM NOW ON, I WILL NOT TRY TO REASON WITH THE IDIOTS I ENCOUNTER. I WILL DISMISS THEM BY WAVING MY PAW AND SAYING “BAH.”

JUST BECAUSE SOMEONE THINKS DIFFERENTLY FROM YOU DOESN’T MEAN HE’S AN IDIOT, DOGBERT.

Copyright © 1996 United Feature Syndicate, Inc.
Redistribution in whole or in part prohibited
Further Reading

- Berkun: *The Art of Project Management*
- 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 and Code Quality*