



# Essential Software Skills For Research Scientists



*Dr. Gregory V. Wilson*  
*gvwilson@cs.utoronto.ca*

---

Copyright © 2006 Gregory V. Wilson.  
This presentation may be used and distributed freely with attribution.

# Plus Ça Change...

---

“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?

# ...The State of Play|

---

- 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!


---

 Do you use version control?

 Can you rebuild with one command?

 Do you write your tests before your code?

 Do you run your tests before checking in?

 Do you know how much of your code they cover?

 Do you have a bug database?

 Do you use assertions and other defensive programming techniques?

 Do you use a symbolic debugger?


# ...Self-Assessment!


---


 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?

# 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?

# 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

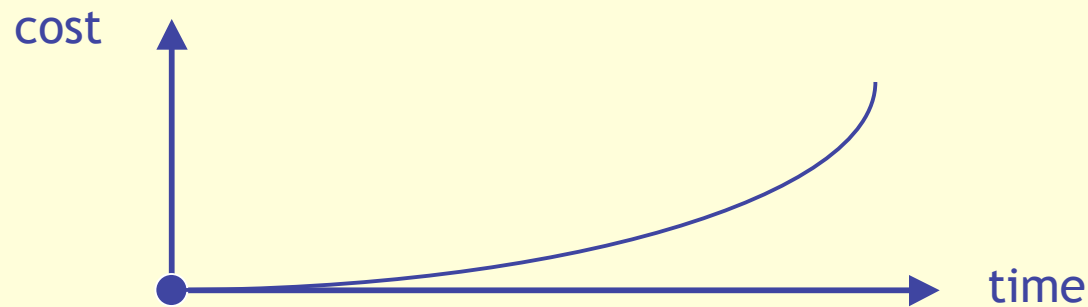
# Grand Challenges

---

- 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



# 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?”

- Wiki
- Event Log
- Roadmap
- Browse Source
- View Tickets**
- New Ticket
- Mail
- Help/Guide
- Search:
- 



**Ticket Query** (79 matches)

Filters

Status:  open  completed  reopened  rejected

Owner: is

Group results by:   descending  Show full description under each result

Ticket	Summary	Priority	Reporter	Type	Created	Last_modified
93	fix sizes of graphic images	high	gvwilson	enhancement	2006-01-30 07:30:23	2006-01-30 07:30:23
96	Make sure all Python code conforms to Python style guidelines	medium	gvwilson	defect	2006-02-01 17:18:51	2006-02-01 17:18:51
97	Check nifty.stanford.edu for ideas for Python projects	medium	gvwilson	enhancement	2006-02-03 18:33:50	2006-02-03 18:33:50
98	Get PDF of crunch mode paper into repository	medium	gvwilson	enhancement	2006-02-04 12:58:17	2006-02-04 12:58:17
100	Implement 'head' and 'tail' filters in inclusions.	medium	gvwilson	enhancement	2006-02-04 13:01:10	2006-02-08 09:59:07
1	readstyle - Describe PyDoc	medium	sdawson	defect	2006-01-29 22:03:36	2006-01-29 22:03:36
2	lec - Add algorithmics lecture	medium	sdawson	defect	2006-01-29 22:03:36	2006-01-29 22:03:36
3	version - More focus on revision history as motivation for Subversion	medium	sdawson	defect	2006-01-29 22:03:36	2006-01-29 22:03:36
4	sql - Rework SQL examples to be scientific data	medium	sdawson	defect	2006-01-29 22:03:36	2006-01-29 22:03:36

# 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

The screenshot shows the Wing IDE interface with a Python script named `invperc.py` open. The script contains two functions: `printGrid` and `isBoundary`. The `printGrid` function has a loop that iterates over the grid, and the `isBoundary` function checks for boundary conditions. A red highlight is on the `print` statement in the `printGrid` function. The Exception window shows a `TypeError: list indices must be integers` with a traceback pointing to line 28 in `printGrid`. The Source Assistant window shows a list of symbols, including `FILLED`, `arg`, `extras`, `findNextCell(grid)`, `grid`, `gridSize`, `isAdjacent(grid, x, y)`, `isBoundary(gridSize, x, y)`, `makeGrid(gridSize, valRange)`, `numFilled`, `opt`, `options`, `printGrid(grid, out)`, and `randSeed`.

```
def printGrid(grid, out=sys.stdout):
    '''Print values in grid for debugging purposes.'''
    for x in grid:
        for y in range(len(grid)-1, -1, -1):
            print '%2d' % grid[x][y],
            print ''

# -----

def isBoundary(gridSize, x, y):
    '''True if (x, y) is on the boundary of the grid, False otherwise.'''
    return (x == 0) or (x == gridSize) or (y == 0) or (y == gridSize)
```

Exception: `TypeError: list indices must be integers`

Traceback (innermost last):

```
File "f:\home\swc\src\debugging\invperc.py", line 1, in ?
#usr/bin/env python
File "f:\home\swc\src\debugging\invperc.py", line 86, in ?
printGrid(grid)
File "f:\home\swc\src\debugging\invperc.py", line 28, in
printGrid
print '%2d' % grid[x][y]
```

Symbol: *Not Python code*

# 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

# Automation

---

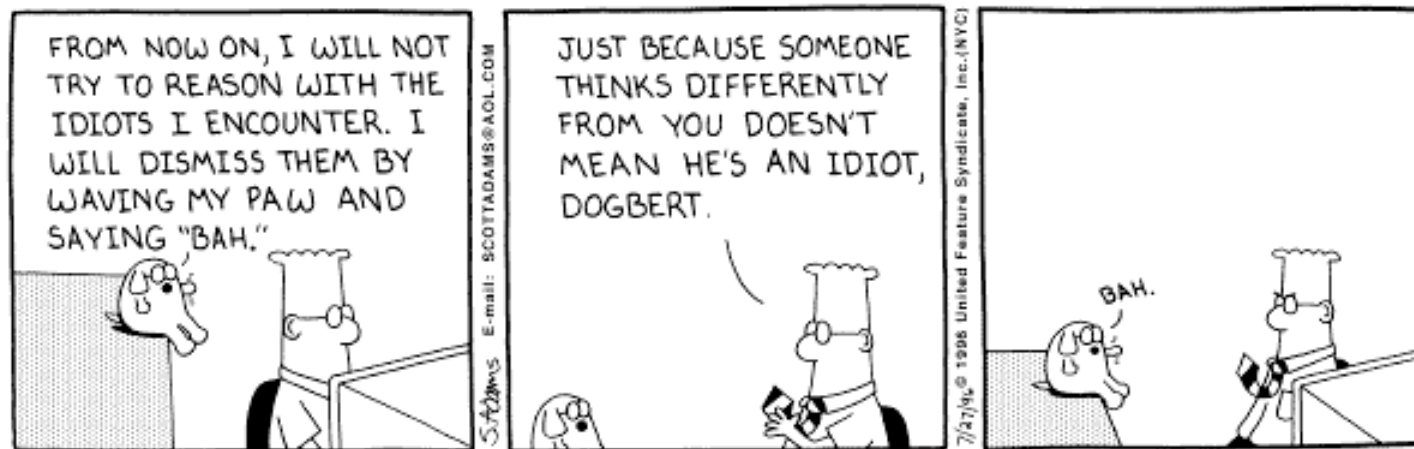
- Theme running through all of these is automation
  - “Anything worth doing again is worth automating.”
- Lots of little tools are a sign of a healthy project
  - Have to be designed and maintained like any other piece of software

# 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

# Questions?



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*
- <http://www.third-bit.com/reading.html>