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

And how long will it take?

Paul Wilson

Inspired by Greg Wilson
Software Carpentry

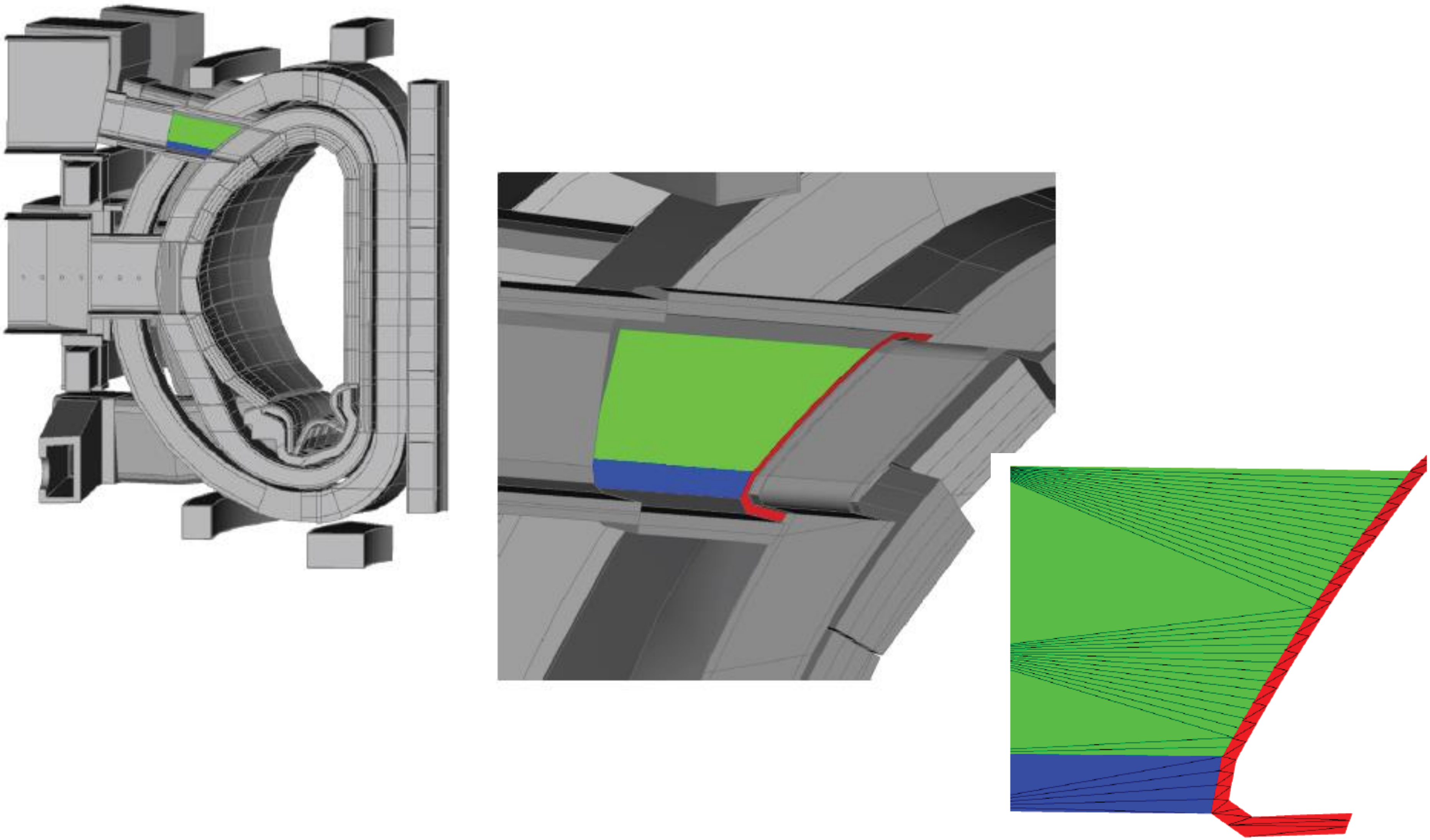
Reality of Research Computing

- Many scientists spend most of their time developing, maintaining, or running software
  - Most don't consider themselves software engineers
  - Few have ever been taught how
    - Learned on-the-job
    - Tribal knowledge

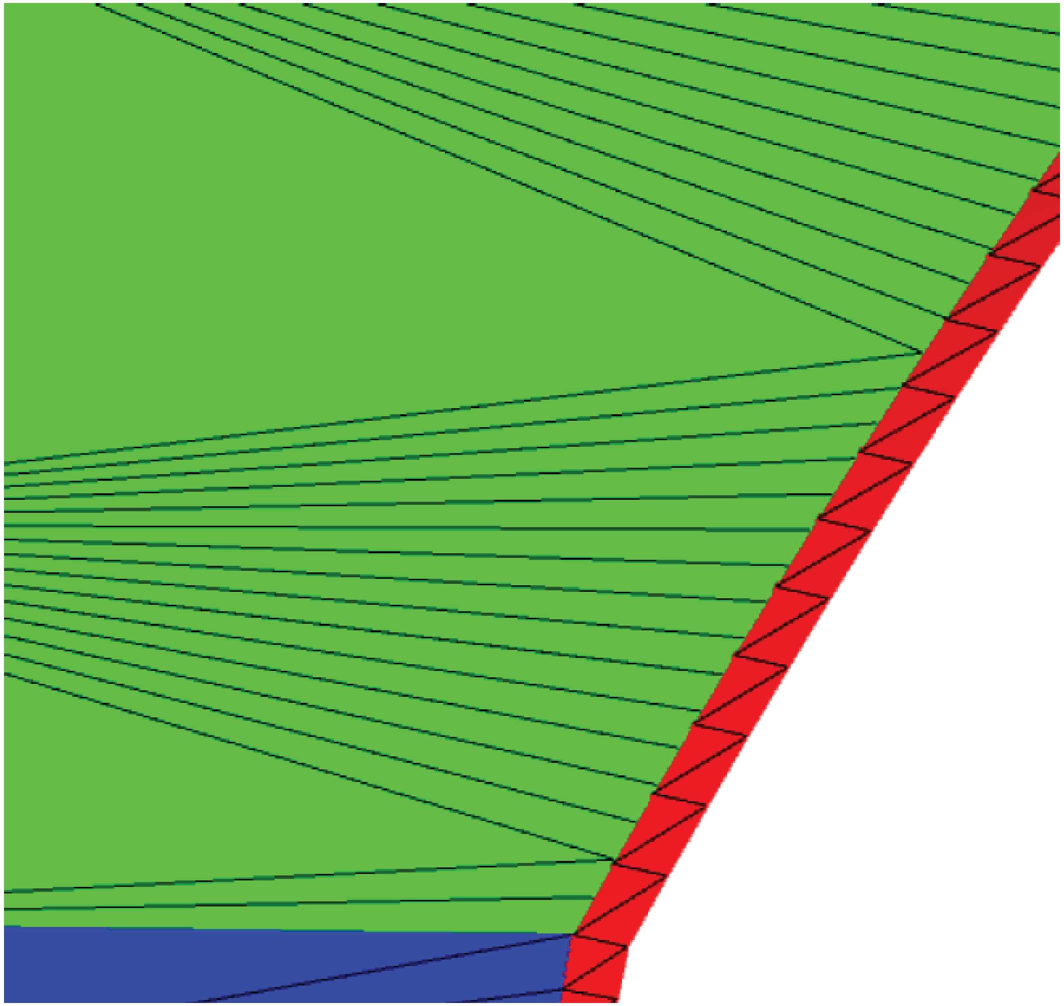
So What…

- Most results take longer to produce than they need to
  - Not because of a lack of computers
- Difficult to assess quality
  - Often measured by reproducibility
  - “System” doesn’t care

A Recent Story:
Sealing a Faceted Geometry

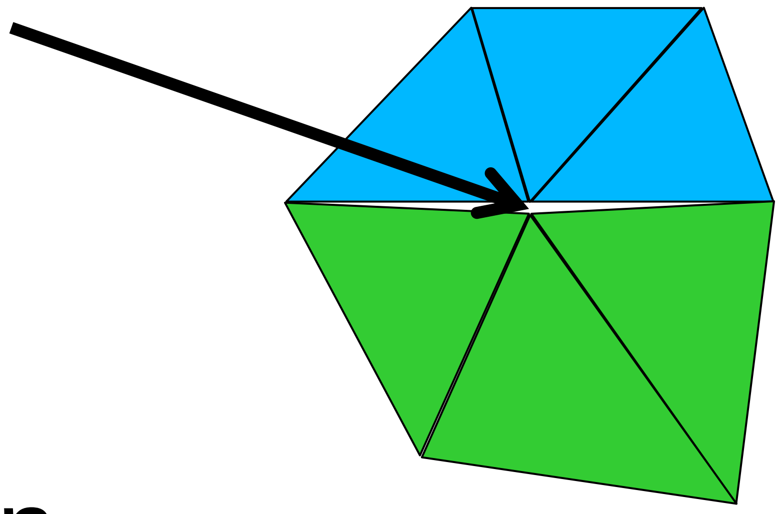


A Recent Story: Sealing a Faceted Geometry



A Recent Story: Sealing a Faceted Geometry

- Lost particles through “leaks”



- Reduce confidence in solution

A Recent Story: Sealing a Faceted Geometry

Model	Particles Simulated [millions]	Lost Particles	
		Original	Robust
UW Nuclear Reactor	41	5649 ± 178	0
Advanced Test Reactor	74	141 ± 32	0
40° ITER Benchmark	225	67 ± 39	0
ITER TBM	205	665 ± 184	0
ITER Module 4	59	59 ± 19	0
ITER Module 13	79	450 ± 60	0
FNG Benchmark	1310	31273 ± 989	0
ARIES First Wall	4070	25 ± 18	0
HAPL IFE	286	65 ± 19	0
Z-Pinch Fusion	409	2454 ± 317	0

Software Carpentry to the Rescue

- Best practices used by the best software engineers whose business is development of quality software
  - They don’t always have formal training
  - They don’t always follow all the practices
  - Growing evidence supported by empirical studies

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

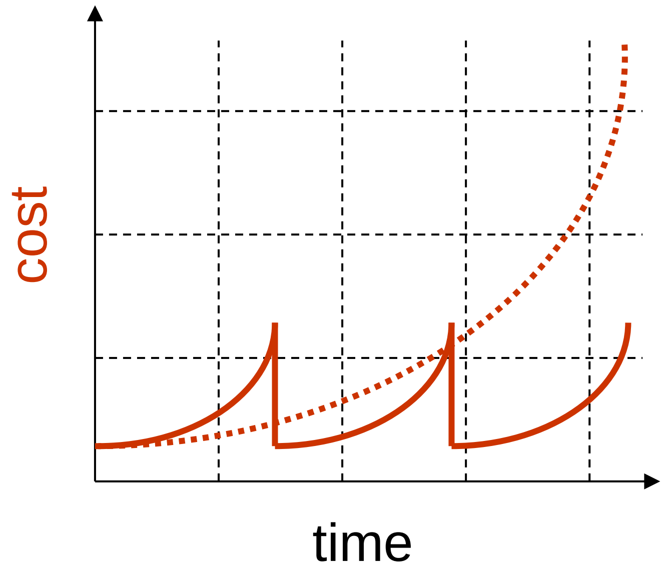
- Careful record keeping is fundamental to science
  - A manual log book works for experiments occurring at a “traditional” pace
  - What happens when you can perform 100 experiments/day? 1000? 10,000?

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?

A Recent Story:
Sealing a Faceted Geometry

- Why did this disruption happen?
  - Inadequate testing
  - Inadequate reporting of version numbers
  - Lack of automation

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

Two Days ≠ Ten Practices

- Automation (requires Shell)
- Writing Code for People
- Don’t Repeat Yourself (or Others)
- Version Control
- Testing
- Collaboration

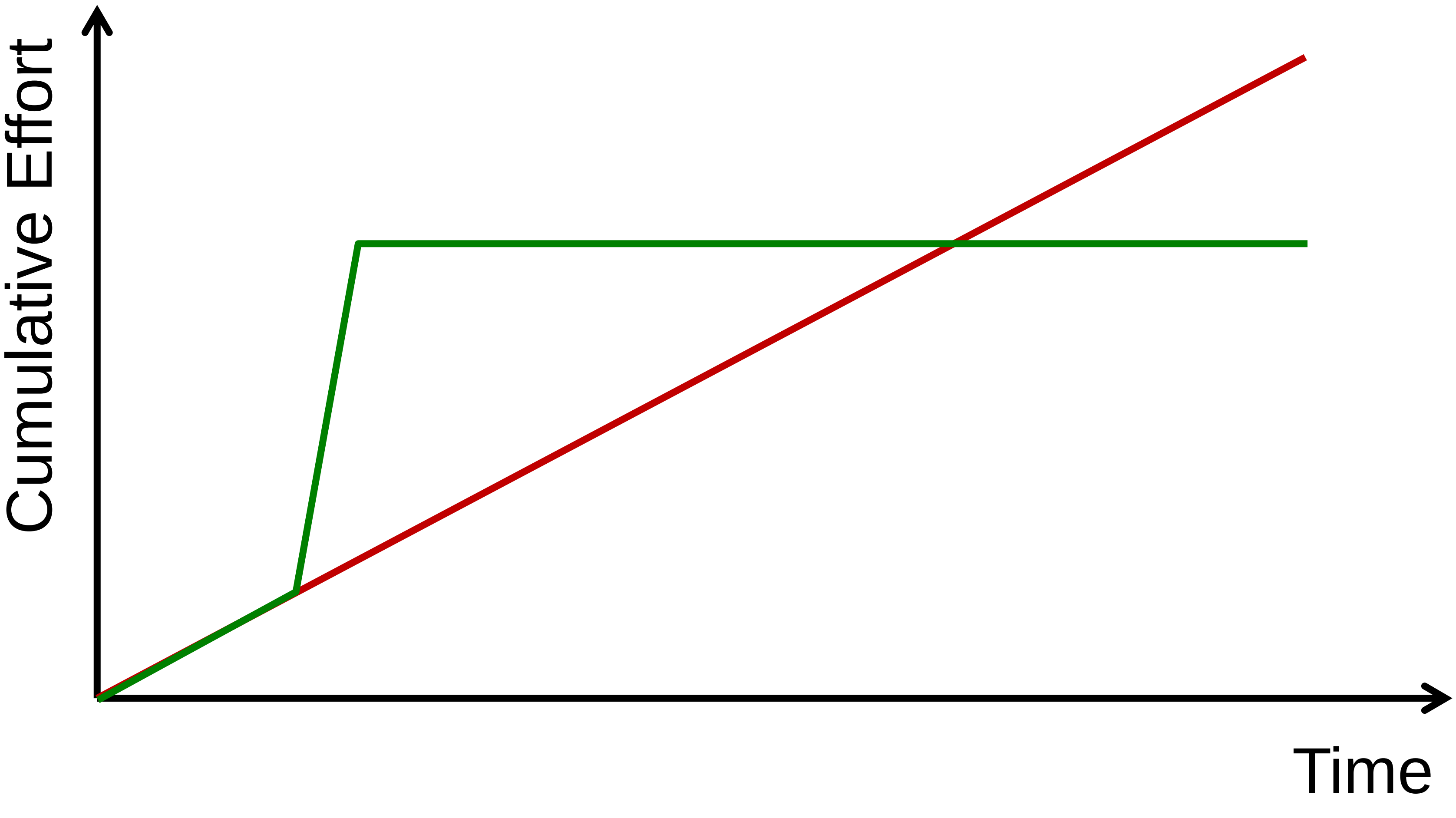
Make Incremental Changes Redux

- This applies to HOW you work
- Choose one practice
  - Implement it in your work
  - Share it with your lab group
  - Allow it to sink in
- Repeat

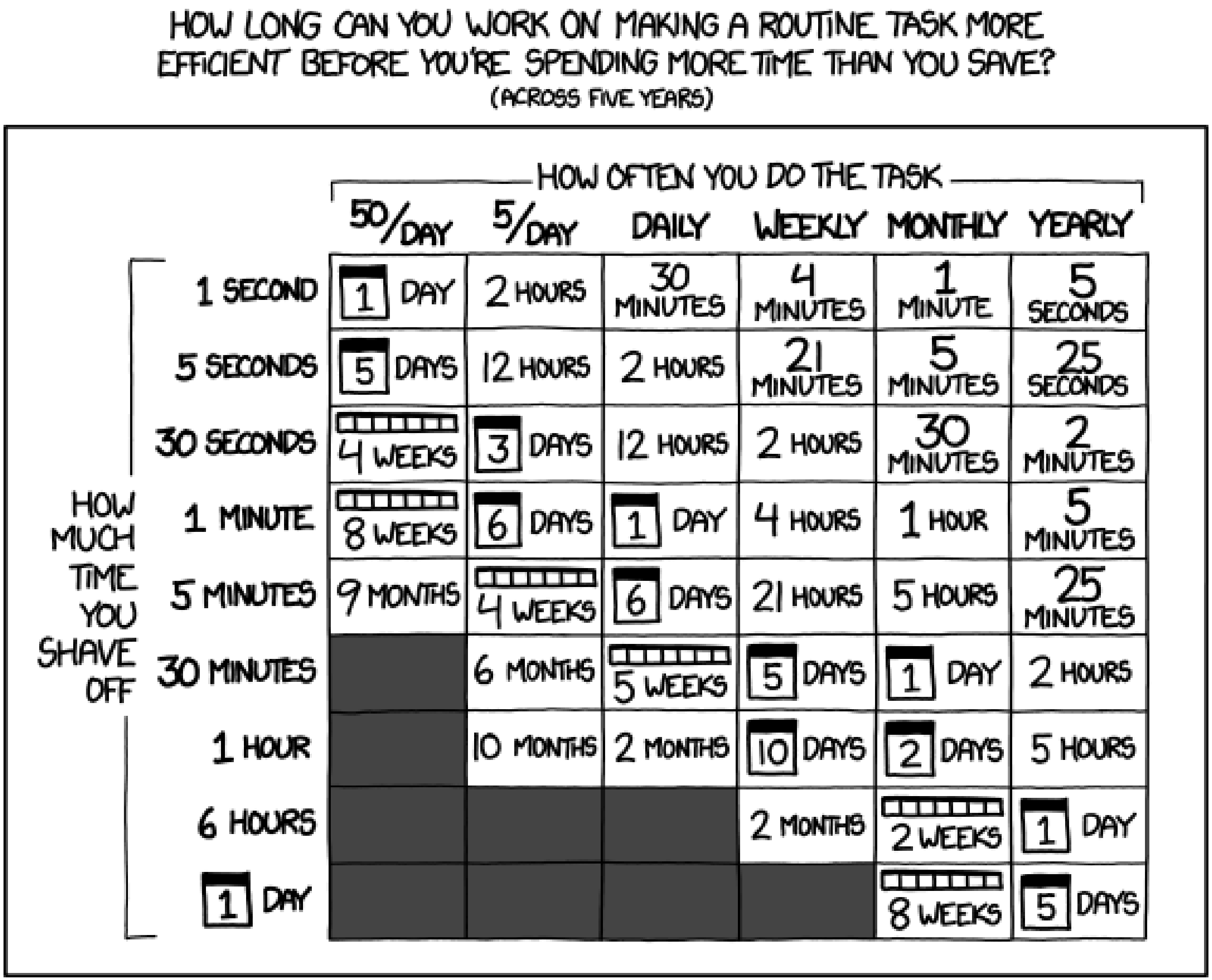
How to Choose Where to Start?

- It will depend on the nature of your work
- Consider the purpose:
  - Improve productivity
  - Improve quality

Thoughts on Productivity and Automation



Thoughts on Productivity and Automation



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF / HOW OFTEN YOU DO THE TASK	50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY					8 WEEKS	5 DAYS