

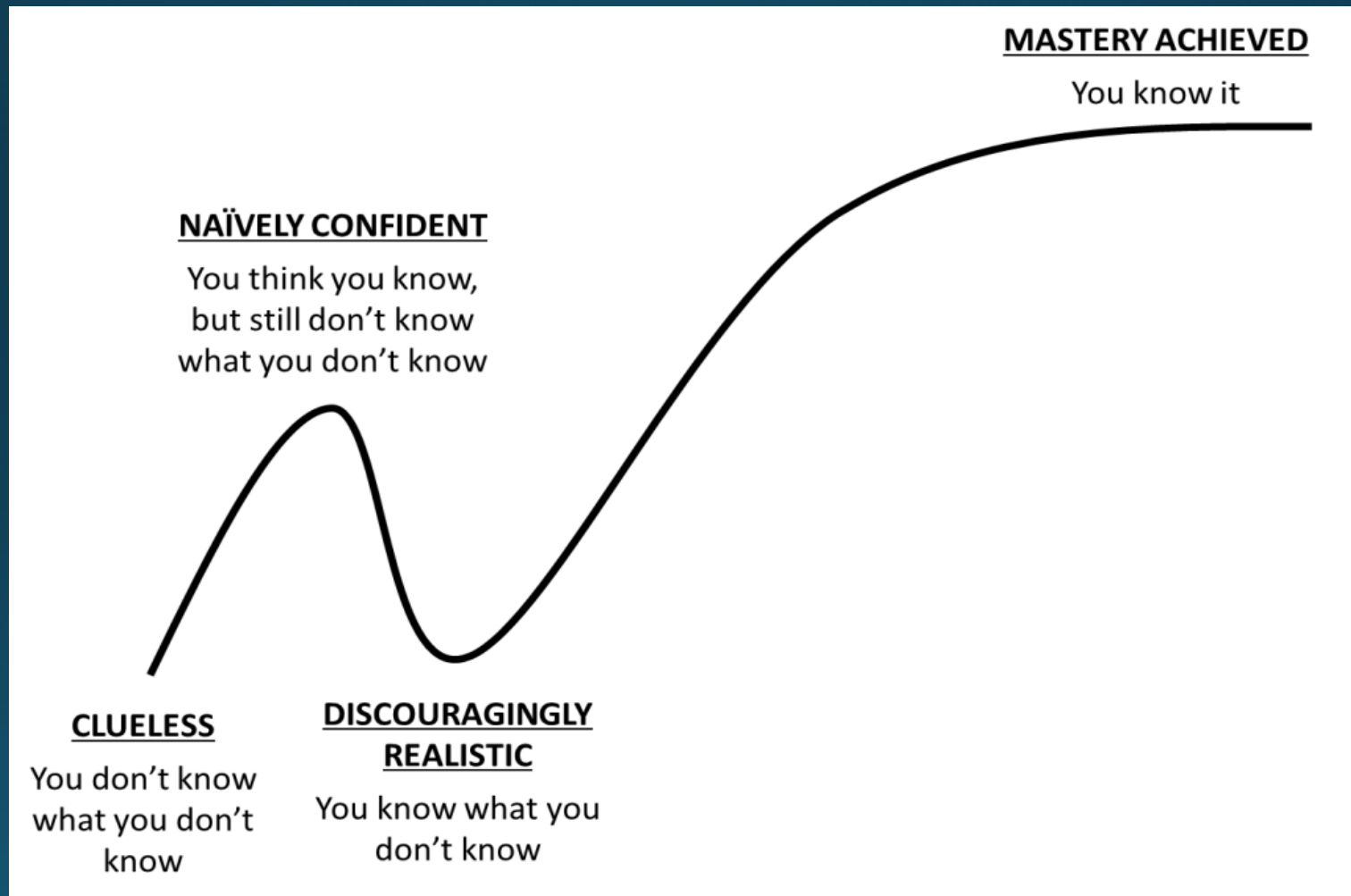
Today's Agenda

- Last Time in CSE 112
- Peter's Perspective
 - What he roughly wanted last time
 - His interpretation of the process
 - What he really wants this time
- Logistical announcements
 - Groups
 - Participation Approaches
- Reviewing the Homework
 - Why did we do such a homework?
 - Space Shuttle discussion
- The Lecture

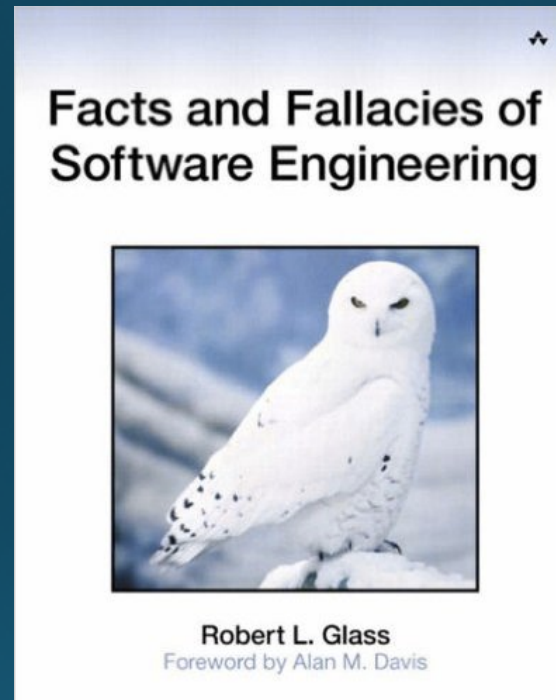
Lecture 2

Making a Software Engineer

Where are you?



Fact #1



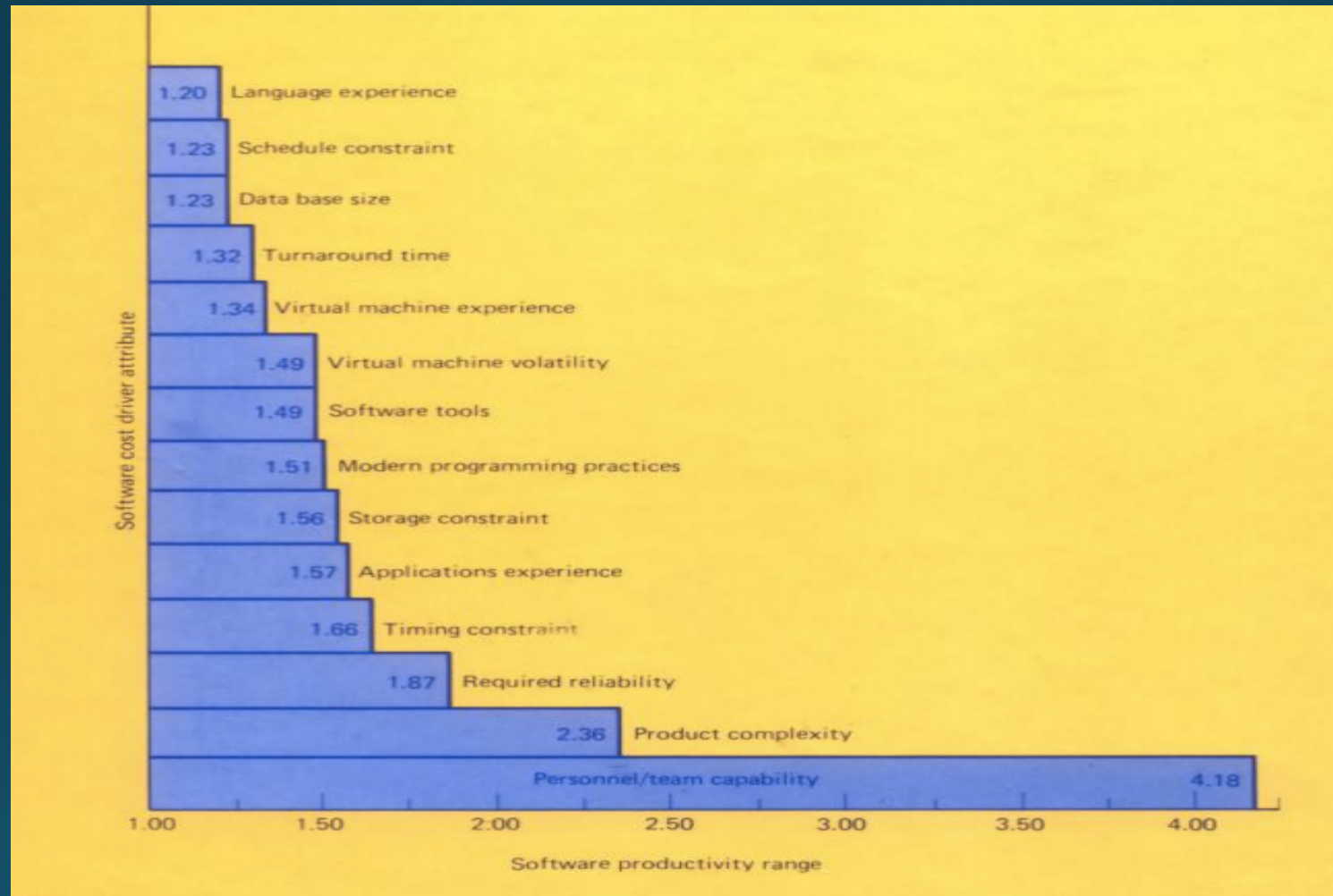
The most important factor in improving software development is not the tools and techniques used by programmers, but the quality of the programmers themselves

NO MATTER WHAT THE PROBLEM IS



IT'S ALWAYS A PEOPLE PROBLEM

Example #1



Credit: *Software Engineering Economics* by Barry Boehm

Example #2: 10X Programmers

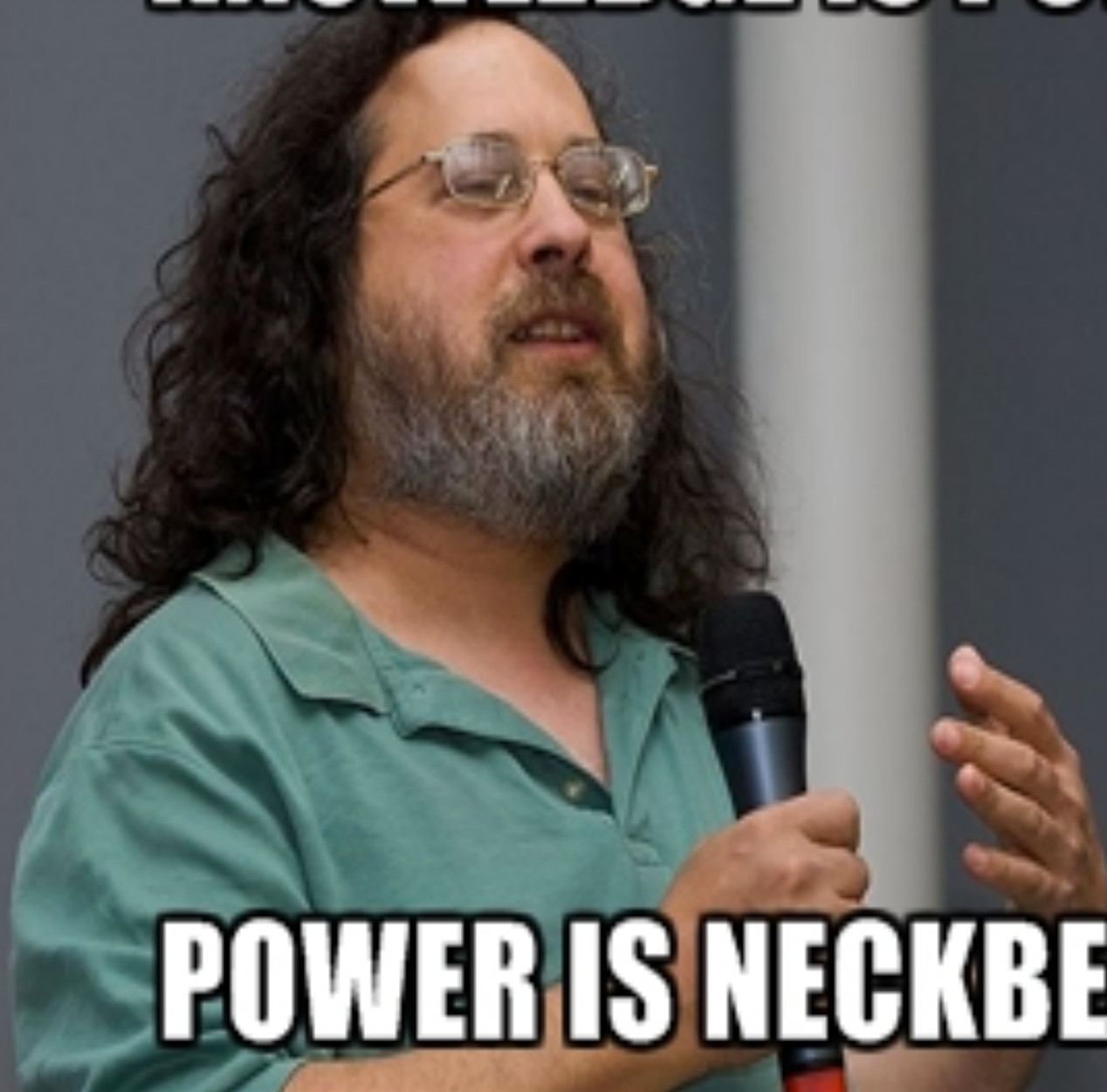
“...researchers have found 10-fold differences in productivity and quality between different programmers with the same levels of experience”

http://www.construx.com/10x_Software_Development/Productivity_Variations_Among_Software_Developers_and_Teams_The_Origin_of_10x/

The romantic image of an über-programmer is someone who fires up Emacs, types like a machine gun, and delivers a flawless final product from scratch. A more accurate image would be someone who stares quietly into space for a few minutes and then says “Hmm. I think I’ve seen something like this before.”

<http://www.johndcook.com/blog/2009/12/23/why-programmers-are-not-paid-in-proportion-to-their-productivity/>

KNOWLEDGE IS POWER



POWER IS NECKBEARD

memegenerator.net

2X, 10X or 28X?

- Best programmers are X times better than average/worst programmers what X is
 - X generally isn't small per the experts it seems to range from 2, 10, to 28x or more depending on the pundits degree of hyperbole.
 - Let's just agree it is better to have better programmers
 - TAP: "I rather have a coding SEAL team than 50 'hacks' in nearly every situation"
 - What about a brute force job?
 - Think about how the coding SEAL would address this?
 - "Code generator, etc."

Why fight Fact #1?

- We embrace the fallacy because what we do is hard as it is and we want it easier
 - Tools and languages are predictable, people often seem arbitrary and thus are much harder
- People are not computers with emotions, but instead are emotional beings who try to be logical or rational. This includes us.

Fact #5 Supports Fact #1

Software tool and technique improvements account for at most 5-35% percent increase in productivity or quality, not the order of magnitude benefits promised

- Careful: Hypesters tend to have underlying reasons for the hype
 - Consulting, books, \$ attraction (VC), naiveté and wishful thinking
 - Sometimes it is a hyperbolic means to an end
 - Common marketing approach to fixing a market is to make a new market
 - Can't beat the competition or already lost to them
 - Rename/rebrand existing idea and claim it is all new
- We like to believe the fantasy, we wish it were true
 - Less work more gain! 6 min abs, how about 3 min abs

Fact #6 isn't acknowledge enough

Learning a new tool, language, library, technique, etc. actually lowers productivity and quality initially. The gains come only later

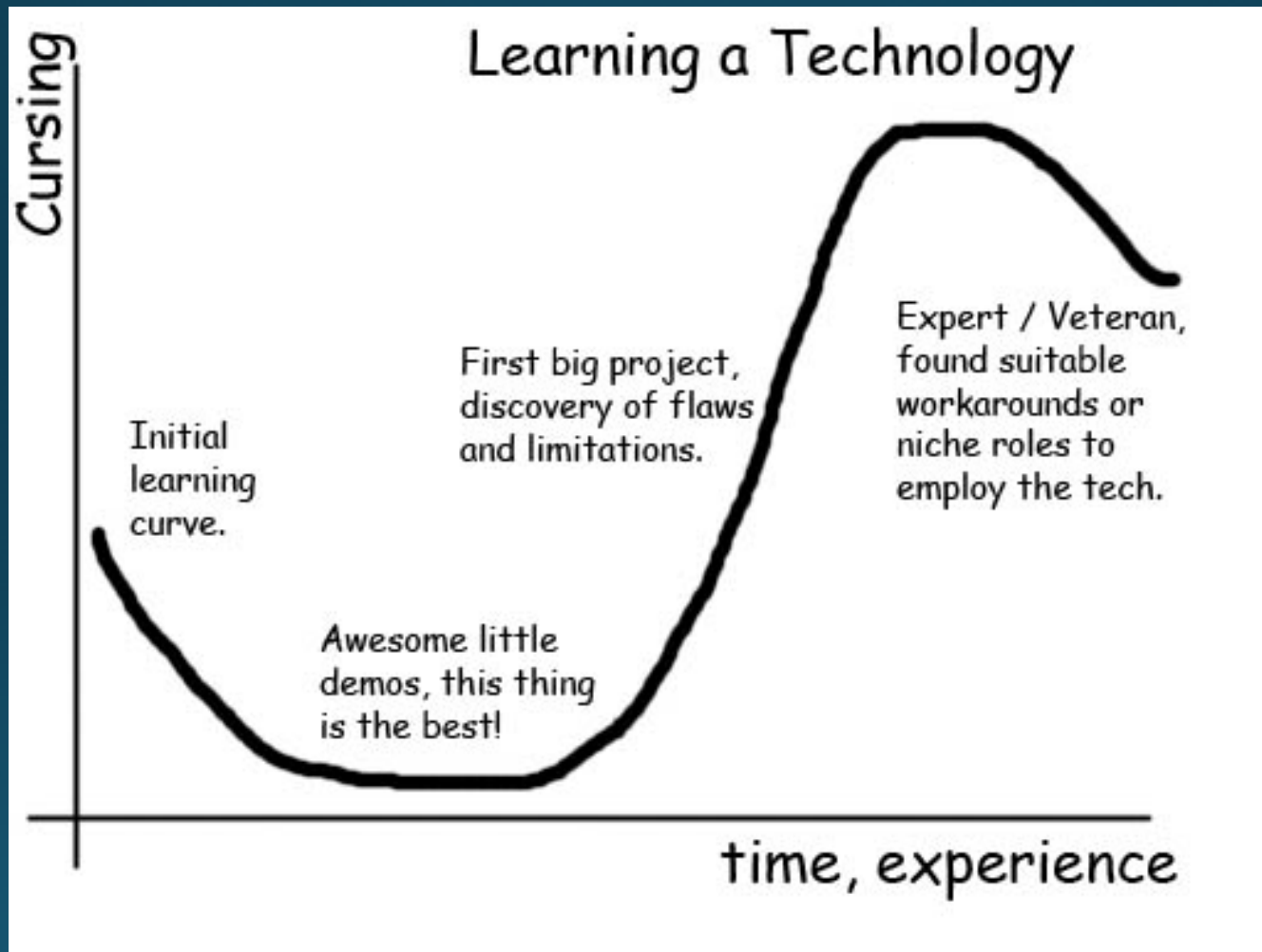
My Related Fact: There is always another tool! Because of this fact it is easy to always be chasing the next best thing!

- We talk a lot about tools, evaluate lots (usually too few), adopt/buy a number of them, initial use some, and very rarely adopt many long term!

From the “Joke” Part of Lecture 1

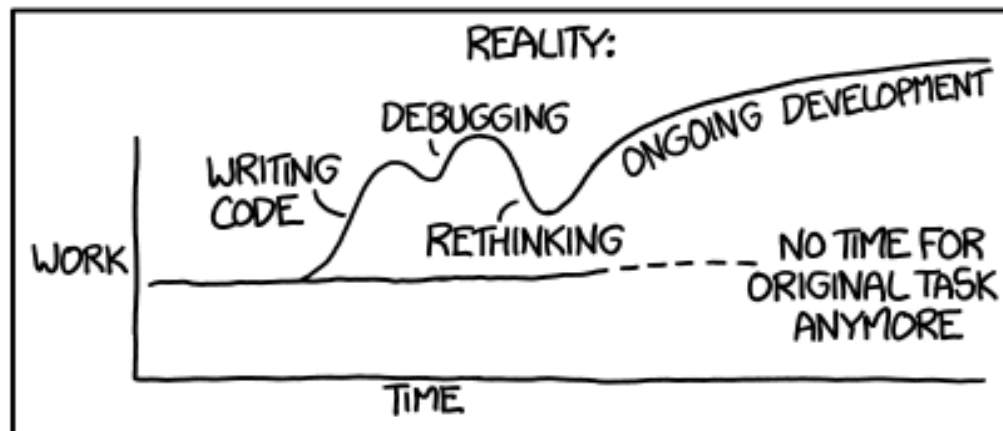
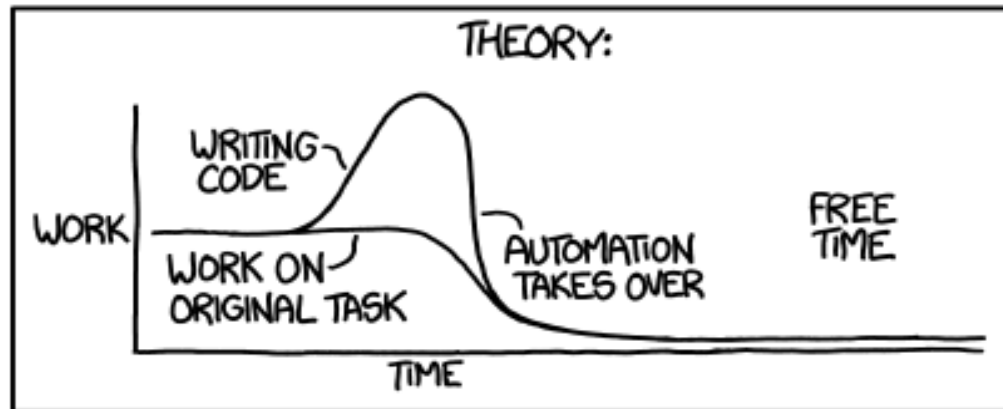


A “Joke” for Lecture 2



Sadly Not a Joke

"I SPEND A LOT OF TIME ON THIS TASK.
I SHOULD WRITE A PROGRAM AUTOMATING IT!"



Programmer MasterThyself

- “When all is said and done, the ultimate factor in software productivity is the capability of the individual practitioner”
 - If this is true, then our goal is to make the individual practitioners better right!?
- Unfortunately: It is quite difficult to ~~control~~ influence the improvement others
- We have a fighting chance with ourselves so we should start there!

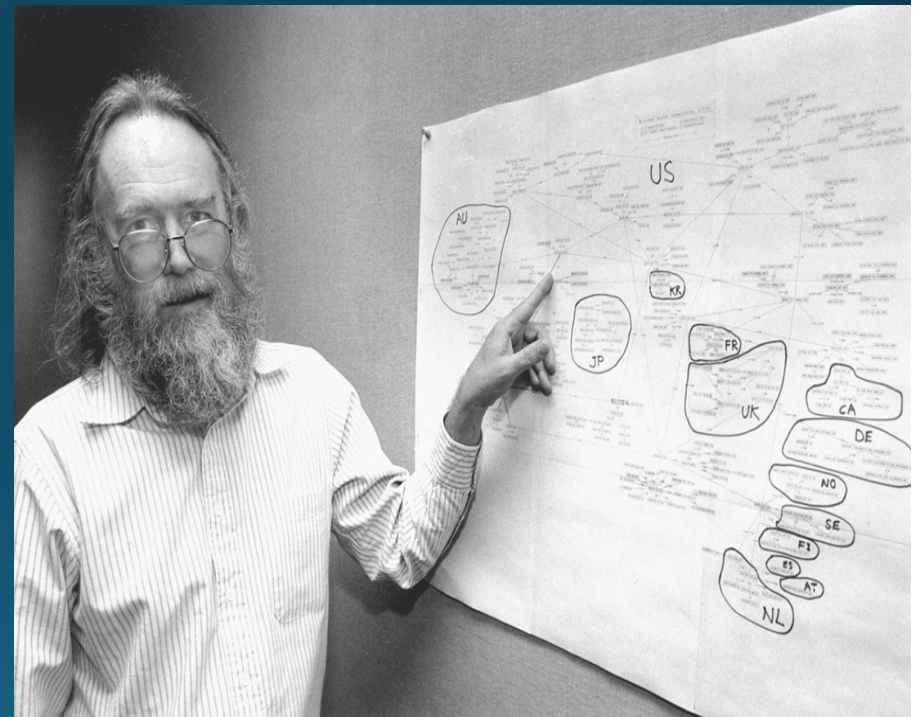
pro•gram•mer

n. [proh-gram-er]

an organism that
turns caffeine and
pizza into software

Postel's Law for Yourself

- Robustness principle
 - *"Be conservative in what you do, be liberal in what you accept from others (often reworded as "Be conservative in what you send, be liberal in what you accept").*
- Roughly applied to our effort you can control what you do and shouldn't contribute to the "mess" but need to be flexible with what others do since they might give you "mess"
- This seems to be a golden rule far beyond computing!



Q: How to make a better programmer?

A: Practice!

- The adage practice makes perfect is clearly not a programming specific idea
 - Example: I want to play the guitar
 - Requirement: Lots of time, need to practice, play chords, play songs badly many times, etc.
- If applied to the act of creating ideas like coding it suggests you need to write lots of bad code before you can write a little great code
 - Long held in writing - the "shitty first draft"
 - Don't forget the rewrite (or in our case refactor!)
- 10,000 hours then!?
 - <http://www.wisdomgroup.com/blog/10000-hours-of-practice/>
- Or not
 - <http://www.businessinsider.com/new-study-destroys-malcolm-gladwells-10000-rule-2014-7>

Ok, so what is my 'guitar'?

- As a coder your 'guitar' is made up of your coding tools
 - Computer
 - Monitor(s)
 - Keyboard
 - OS
 - Editor/IDE
 - Language
 - Framework
 - Misc. Tools – Debuggers, Build Tools, Syntax Analyzers, etc.*
 - * Hesitate the break out since they are often part or heavily dictated by the IDE, framework

So what's your *Trigger*



[https://en.wikipedia.org/wiki/Trigger_\(guitar\)](https://en.wikipedia.org/wiki/Trigger_(guitar))

<https://www.youtube.com/watch?v=b6lBotrJoJU>

Trigger Doesn't Work Without...



You missed the big one!

- You!
 - Your brain
 - Your focus
 - Your energy
 - Your habits
- A large mistake being made by many programmers is ignoring the biologically and emotionally imposed limits of your abilities

Productivity Myths

- Maximum productivity is a myth
- Novice Programmers – believe in utopian work days
- Experienced Software Engineers – accept real world days
- Real world agile does fundamentally agree with this in that in the estimation phase one is supposed to include a velocity factor (often 0.7) to temper work availability and estimates

Work Rhythms Part 1

Time of Day

- Many people argue strongly for morning work regardless of discipline as the most effective approach for productivity increase
- A decent number on the other side cite individual energy patterns and push night work
- Few seem to argue for midday and afternoon work as it appears that human cycles and societal patterns don't seem to make this a great time for productivity
- My experience suggests it is ultimately it is somewhat the way a person is wired. However, rest seems to result in better work energy (thus the morning or the nap or exercise push) and interruptions tend to be a big deal (thus the night or early morning) and tiring out plus food comas are real so watch out for the afternoon

Work Rhythms Part 2

Length of Work

- The problem with too much particularly all at once
- There are clear diminishing returns in focus and quality that have been measured
 - Again this is not at all specific to programming
- Idea: Employ the Pomodoro Technique
 - 25mins on, 5 mins off, work iterations if you like
- Roughly it appears particularly over time it is better to 2-4 hours focused attention (in flow hopefully) daily than 8+ hours of diminishing energy and distraction

Work Rhythms Part 3

Variety and Type of Work

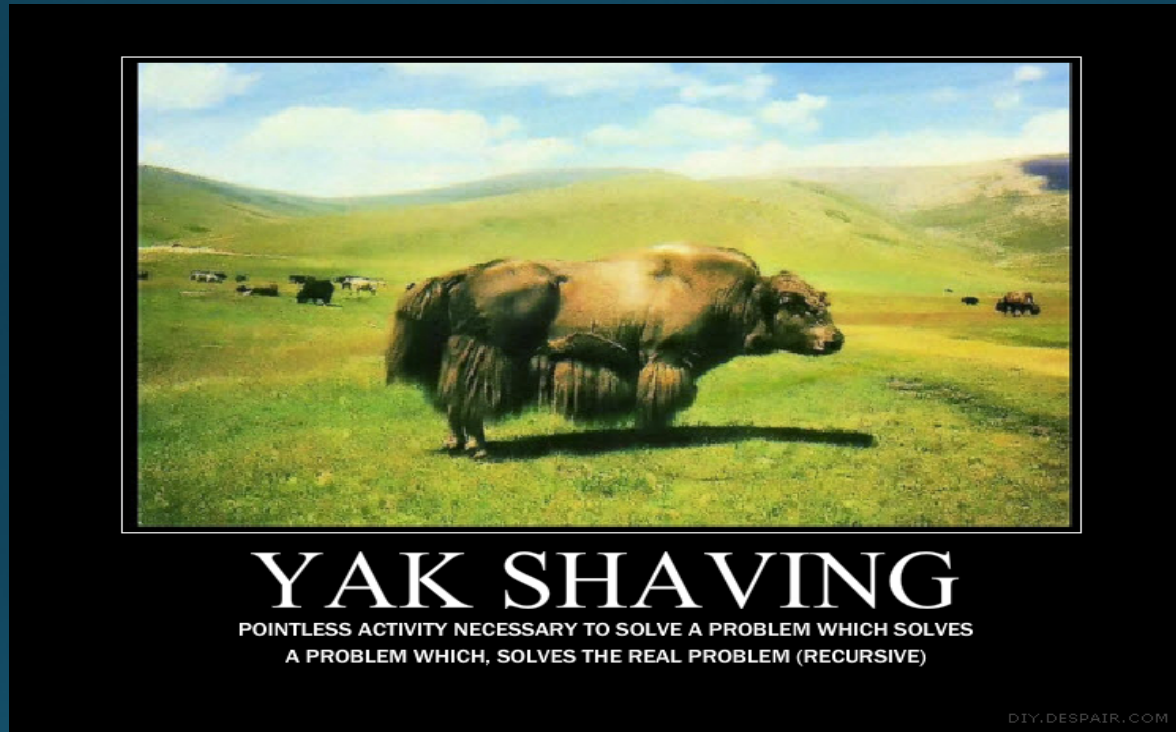
- It is clear that some types of work requires heavy concentration and other type of work doesn't
- Try to schedule your efforts for your energy and focus availability
- Cleaning code and making it meet a style guide can be a great mindless break, but why waste high energy hours doing that?
 - Easy to see people do this when they check off the easy items in an issue tracker right away thinking it builds momentum
 - Productivity research seems to suggest instead you would do better tackling harder things first when energy and attention is most plentiful
- Task switch is like "mental crop rotation" when done right and is distracted and inefficient thrashing when done wrong

Work Rhythms Part 4

Multitasking and Distraction

- Multitasking just doesn't work. Period. Stop.
 - Yes we do it, but at our detriment for programmer productivity
- The Internet sings a constant siren song of better opportunities and information just over the horizon
 - There appears to be a generalized fear of being left behind, whether it be a social update or a new framework
- We can easily trick ourselves to seek out more motivating things often in the guise of being looped in to what we are supposed to be doing
 - Examples: Product Hunt, Reddit, HackerNews, TechCrunch, etc.
 - We even see people with productivity porn like habits always looking for the new GtD system, Todo list, Lifehack, etc.!

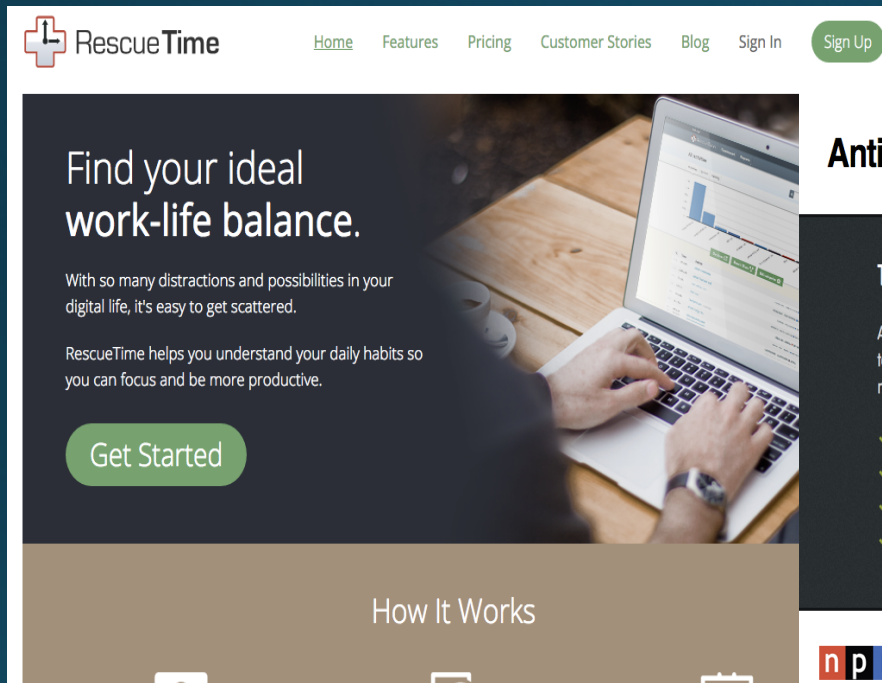
Yak Shaving != Productivity



<http://www.hanselman.com/blog/YakShavingDefinedIIIGetThatDoneAsSoonAsIShoveThisYak.aspx>

http://sethgodin.typepad.com/seths_blog/2005/03/dont_shave_that.html

Why do we need these?



RescueTime [Home](#) [Features](#) [Pricing](#) [Customer Stories](#) [Blog](#) [Sign In](#) [Sign Up](#)

Find your ideal work-life balance.

With so many distractions and possibilities in your digital life, it's easy to get scattered.

RescueTime helps you understand your daily habits so you can focus and be more productive.

[Get Started](#)

How It Works



Anti-Social 

[Need Support?](#) Call 415-968-9172 [Email Us:](#) support@80pct.com

Target Your Digital Distractions

Anti-Social is the amazing app that makes it easy for you to target and block any distracting website so you can be more productive.

- ✓ Works with Windows and Macs, including El Capitan
- ✓ Block any site that wastes your time
- ✓ Keeps you honest, hard to turn off!
- ✓ Fully supported, 60-day money back guarantee

\$15

60 Day Money Back Guarantee

[TRY NOW >](#)

As seen in

[Features](#) [Product Tour](#) [Try Anti-Social](#) [Press Coverage](#)

Work Rhythm Part 4

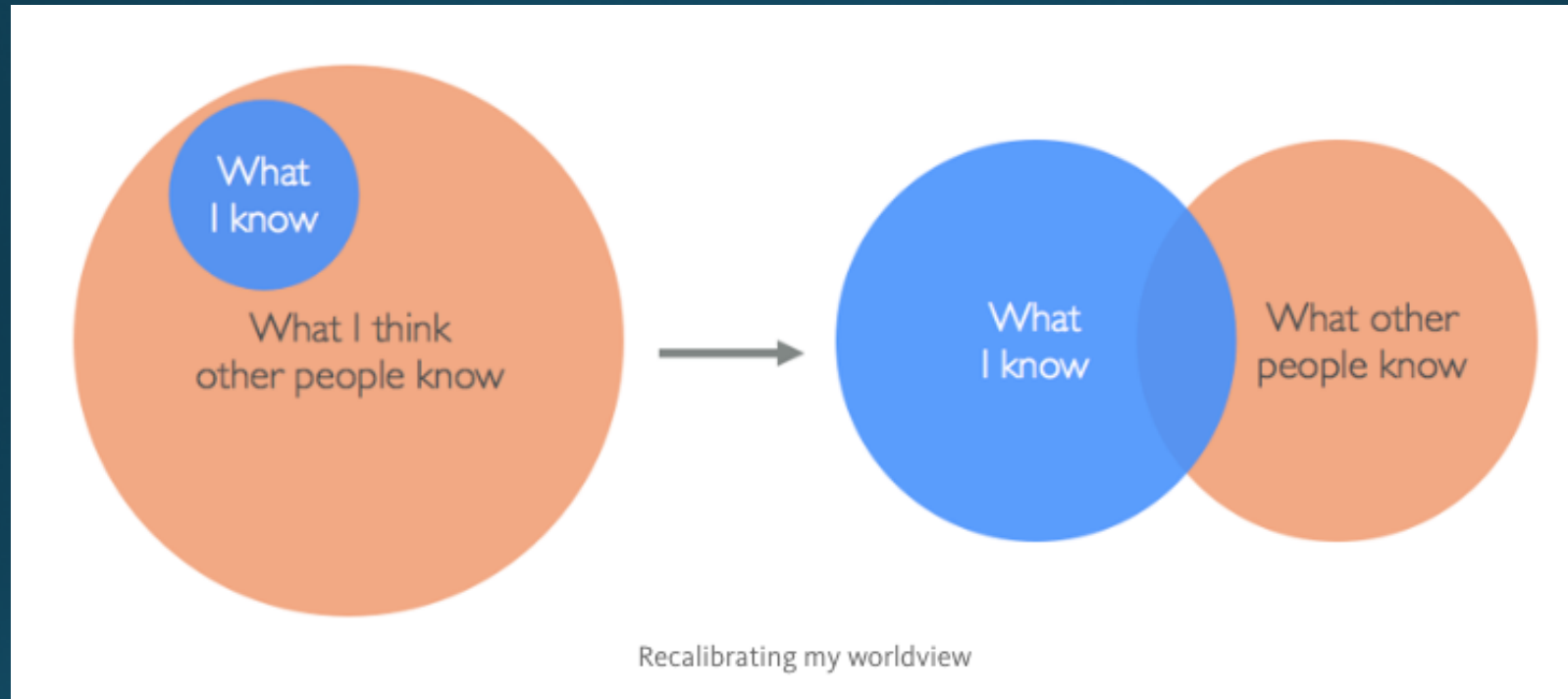
Flow State

- Focused concentration where lots of other things melt away and you often are highly effective (and calm)
 - [https://en.wikipedia.org/wiki/Flow_\(psychology\)](https://en.wikipedia.org/wiki/Flow_(psychology))
 - <http://psygrammer.com/2011/02/10/the-flow-programming-in-ecstasy/>
- Getting into flow state is difficult somewhat like meditation and requires practice
 - Getting knocked out of it is easy with interruptions
 - Headphones, private offices over open floor plans
 - Distractions – chat, email, the Internet 😊
- Generally a few hours of flow is better than a day of bouncing around, even though you might feel you worked a lot you might have accomplished a little

Living with Imperfection

- Fact: Some days just don't go well and you won't get into flow at all
- Very productive good days shouldn't be interpreted as what should always happen and in fact may only be good relatively or be enabled because of many "less good" days
- Ultimately you will fail
 - You will ...
 - write some boneheaded code
 - push broken code to master
 - overwrite working code
 - get very little done
 - Add your own
- Be very careful relating a unproductive day, bad code review, screwed up algorithm, crash, etc. with your self worth.

Impostor Syndrome



<https://medium.com/@aliciatweet/overcoming-impostor-syndrome-bdae04e46ec5#.jl5rmjs2u>

[https://www.reddit.com/r/cscareerquestions/comments/1tbmmm/how do you deal with impostor syndrome as a/](https://www.reddit.com/r/cscareerquestions/comments/1tbmmm/how_do_you_deal_with_impostor_syndrome_as_a/)

Healthy View of Being Novice

This is not Impostor Syndrome
This is Reality for every beginner



<http://alicialiu.me>

Perfectionism



- [https://en.wikipedia.org/wiki/Perfectionism_\(psychology\)](https://en.wikipedia.org/wiki/Perfectionism_(psychology))
 - This is actually not a positive trait, some tendencies are but in general it is very counter productive
- Perfectionism really seems anti-Agile as agile embraces continuous improvement
 - In my experience perfectionists don't ship software often and when they do it isn't perfect anyway
- Procrastination is related to perfectionism

Programmer Personalities

The 5 types of programmers

July 18th, 2010 – Desktop Development, Web Development (Opinion)

272 Comments »



In my code journeys and programming adventures I've encountered many strange foes, and even stranger allies. I've identified at least five different kinds of code warriors, some make for wonderful comrades in arms, while others seem to foil my every plan.

However they all have their place in the pantheon of software development. Without a healthy mix of these different programming styles you'll probably find your projects either take too long to complete, are not stable enough or are too perfect for humans to look upon.

The duct tape programmer

The code may not be pretty, but damnit, it works!

This guy is the foundation of your company. When something goes wrong he will fix it fast and in a way that won't break again. Of course he doesn't care about how it looks, ease of use, or any of those other *trivial* concerns, but he will make it happen, without a bunch of talk or time-wasting nonsense. The best way to use this person is to point at a problem and walk away.



The OCD perfectionist programmer

You want to do what to my code?

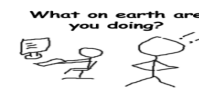
This guy doesn't care about your deadlines or budgets, those are insignificant when compared to the art form that is programming. When you do finally receive the finished product you will have no option but submit to the stunning glory and radiant beauty of perfectly formatted, no, perfectly beautiful code, that is so efficient that anything you would want to do to it would do nothing but defame a masterpiece. He is the only one qualified to work on his code.



The anti-programming programmer

I'm a programmer, damnit. I don't write code.

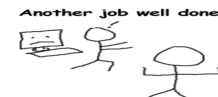
His world has one simple truth; writing code is bad. If you have to write something then you're doing it wrong. Someone else has already done the work so just use their code. He will tell you how much faster this development practice is, even though he takes as long or longer than the other programmers. But when you get the project it will only be 20 lines of actual code and will be very easy to read. It may not be very fast, efficient, or forward-compatible, but it will be done with the least effort required.



The half-assed programmer

What do you want? It works doesn't it?

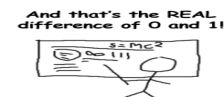
The guy who couldn't care less about quality, that's someone else's job. He accomplishes the tasks that he's asked to do, quickly. You may not like his work, the other programmers hate it, but management and the clients love it. As much pain as he will cause you in the future, he is single-handedly keeping your deadlines so you can't scoff at it (no matter how much you want to).



The theoretical programmer

Well, that's a possibility, but in practice this might be a better alternative.

This guy is more interested in the options than what should be done. He will spend 80% of his time staring blankly at his computer thinking up ways to accomplish a task, 15% of his time complaining about unreasonable deadlines, 4% of his time refining the options, and 1% of his time writing code. When you receive the final work it will always be accompanied by the phrase "if I had more time I could have done this the right way".



<http://stevenbenner.com/2010/07/the-5-types-of-programmers/>

Mental Costs of Decisions

- Optimal decisions and will power have been studied to come from a finite source
 - This source is related to rest and glucose availability
- Generally this idea is related to what is termed decision fatigue
 - https://en.wikipedia.org/wiki/Decision_fatigue
- General Effect

"...if your willpower is depleted, you're more likely to make a decision that's good in the short-term, but bad in the long-term. Once your willpower is depleted, you become less likely to thoroughly consider the complex tradeoffs some decisions require."

<http://greatnotbig.com/2011/08/decision-fatigue/>
- Trouble: Coding is filled with decisions!

Decision Fatigue and the Modern Programmer

- Programmers are people this stuff effects us
 - Maybe more than the average person since we work so heavily with decisions
 - <http://effectivesoftwaredesign.com/2011/08/23/how-decision-fatigue-affects-the-efficacy-of-programmers/>
- Example: Copy-Paste coding
 - What's easier thinking out an abstraction or copying paste some working code?
 - Answer: The later
 - Downside: Your code bloated! Short-term gain but long term consequence of software rot

What!!?? Why?

```
if (me.initAfterRender) {  
  if (me.initIf && eval(me.initIf)) {  
    me.widget.on('afterrender', me.init, me);  
  } else {  
    me.widget.on('afterrender', me.init, me);  
  }  
} else if (me.initOn) {
```

Stupid Mistake?

Tired?

Both?

Tech Fatigue

- Computing and the web space in particular suffers from a proliferation of technologies
 - <http://www.sitepoint.com/drowning-in-tools-web-development-industry/>
 - <https://medium.com/@ericclemmons/javascript-fatigue-48d4011b6fc4#.tvzld2bjg>
 - TodoMVC example
 - Languages that compile to JS
- Generalized false assumption: 2.0 better than 1.0
 - If so why do the majority of movie sequels suck so bad!?
- Combine this with impostor syndrome (if I don't learn X I am not worthy) and perfectionism (better tool or approach just over the horizon) and you can spend more time evaluating than doing (remember Yak Shaving?)

Decision Off Load

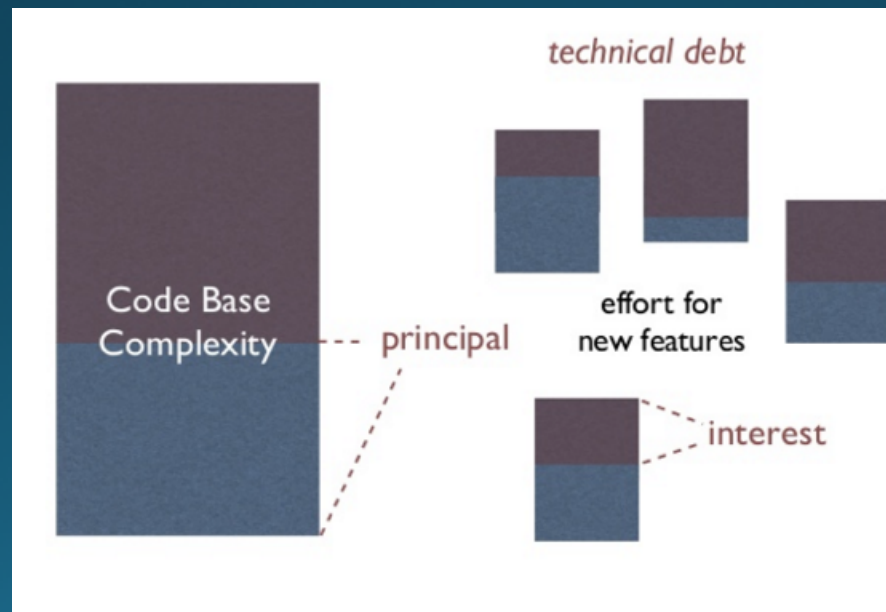
- Employing deep higher level thinking uses lots of decision energy and is 'expensive'
- Solution 1: habits offload decisions
 - Think CPU vs GPU, we want to move lots of our stuff to the "HPU" (Habit Processing Unit) of coding
 - Conclusion: We should develop good programmer habits or programming practices
- Solution 2: outsource decisions
 - Use mental short-cuts like social proof to make decisions
 - Opinionated Programming and Frameworks – people make a decision and you just should accept it, it's easier
 - Solution 2 seems pretty dangerous and goes against our premise of only being able to control for ourselves

Good Habits

- Habit - an acquired behavior pattern regularly followed that becomes almost involuntary
- Habits are the kind of the things that can lead to some forms of 'multitasking'
 - The muscle memory of editing and reformatting for example
- How long to develop a habit?
 - 21 days, 6 months, a long time
- Hard to stick to habits
 - Try 'Micro Habits' – 1 yard plays Joe and I called it
 - Code cleaning, relate to job site cleaning and technical debt
- Keystone Habits can lead to much much more

Habit Example

- Cleaning up the work site I mean code base is a really good habit to have
 - If you don't clean up, generally bad things happen (eventually)
 - "Sweeping" your code can be the boring part but it is vitamins that avoids your eventual need for aspirin



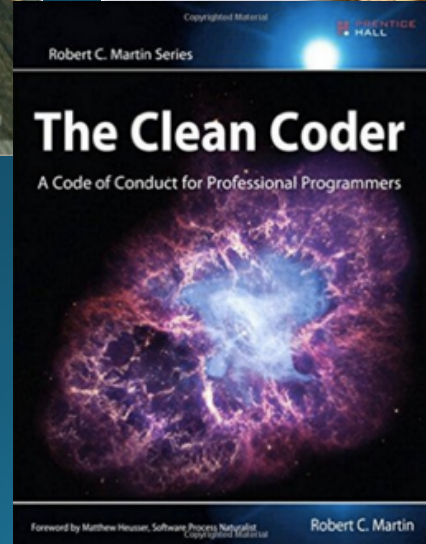
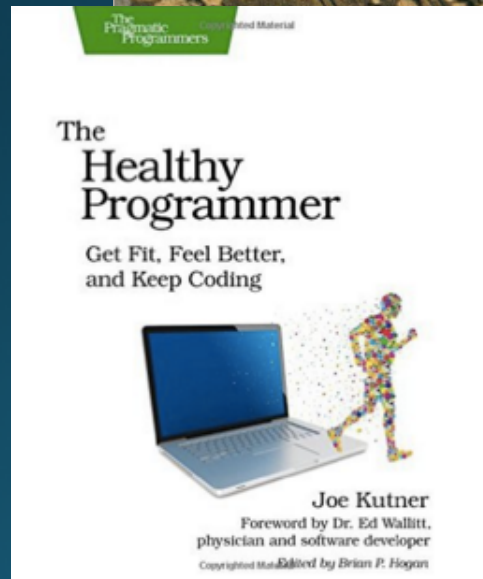
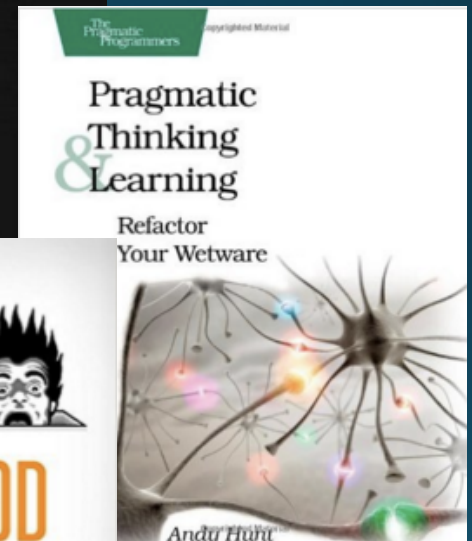
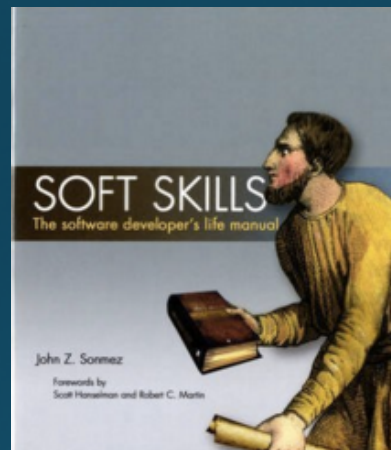
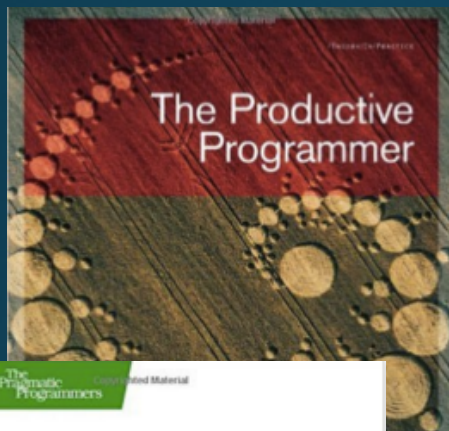
Habits of The “Good” Programmer Trending Again

Summary

- Write clean code
- Test
- Script everything
- Learn your tools by heart
- Optimize for flow

<http://www.slideshare.net/andersjanmyr/habits-of-a-responsible-programmer>

Hmmm...There's a Reason

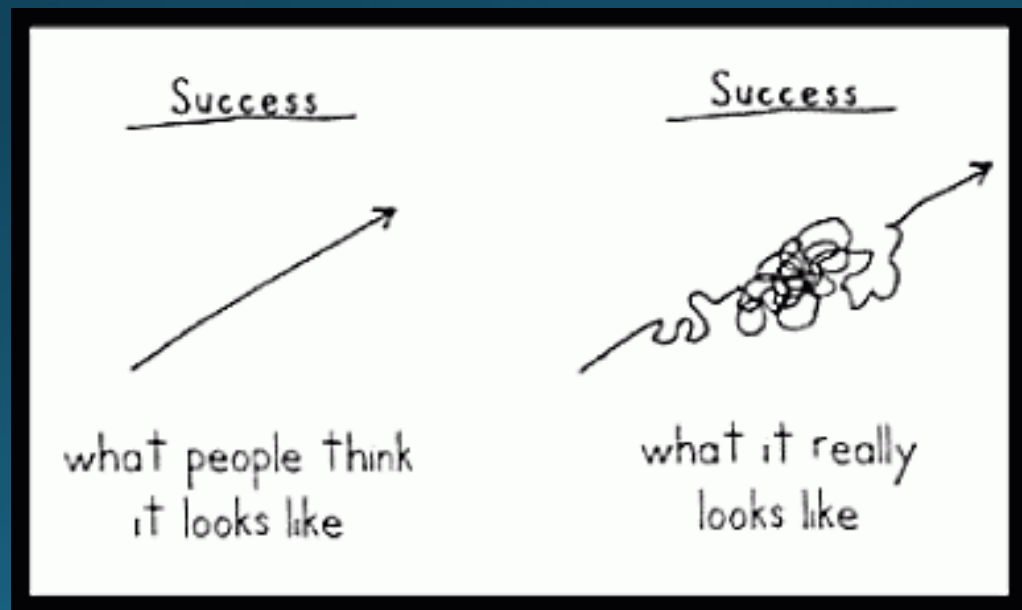


Class Notes on Productivity

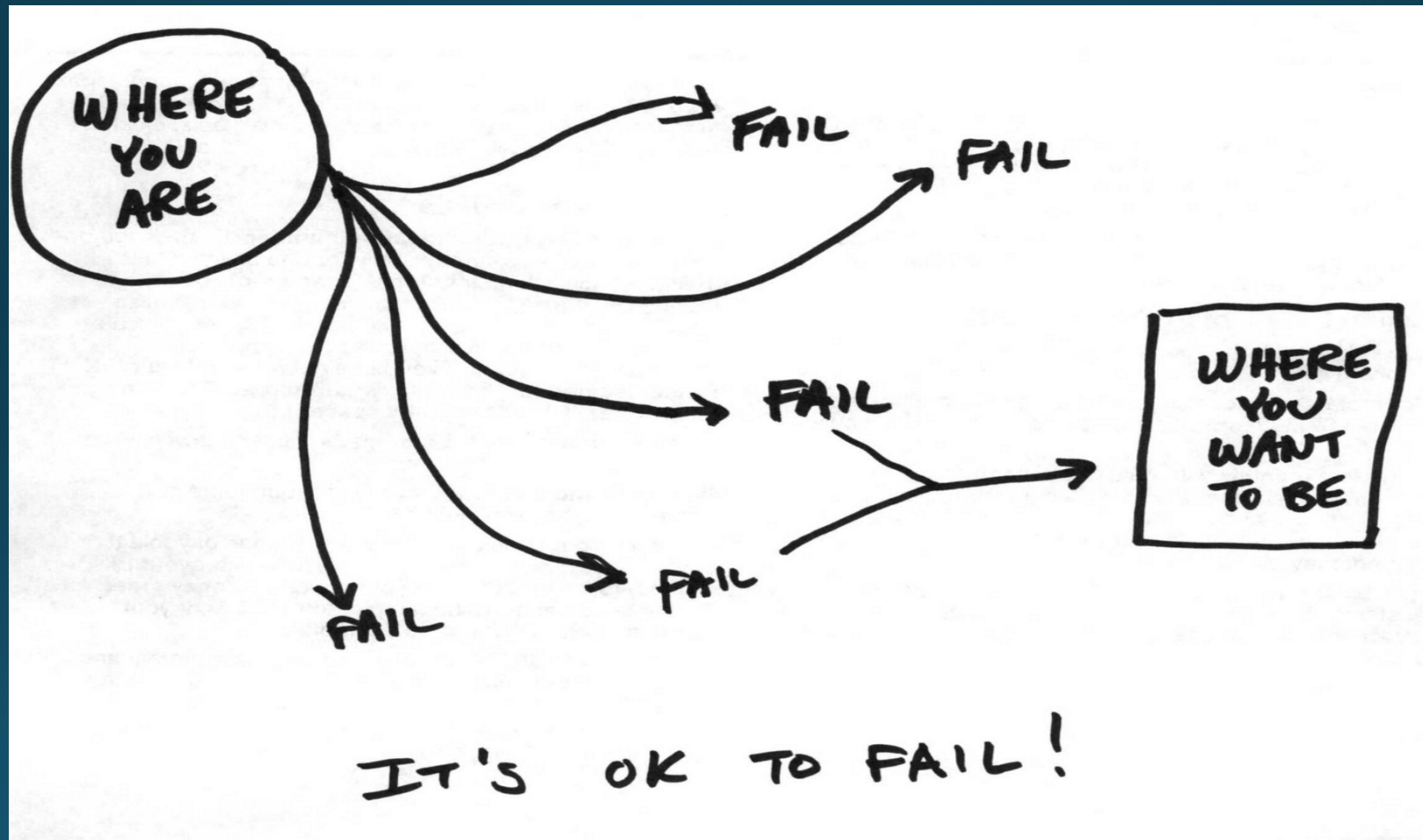
- Clearly we can get really off track and end up deep in self-help tactics and introspection
 - Maybe not a bad thing unless it leads to analysis paralysis
 - However, none of this stuff is good for us as aspiring Software Engineers unless we find some applied use for it
- Participation Homework Choice #1 – Spend some time between now and next Tuesday looking over ideas presented online and in some of the mentioned texts and present 1 or 2 items that seem useful to YOU if you are called on in class

Embrace Failure

- Generally getting to a successful outcome takes many failed outcomes.
 - Well known quotes from Edison and others
- This often goes against the romantic idea of things



Seeing the path to success from the place you failed



The pain of learning and failure is
certainly not a new idea

**LEARNING IS NOT
CHILD'S PLAY; WE
CANNOT LEARN
WITHOUT PAIN.**



Aristotle

Greek philosopher and polymath

(384 BC - 322 BC)

QuoteHD.com


SUCCESS IS A LOUSY TEACHER.



**IT SEDUCES SMART PEOPLE INTO
THINKING THEY CAN'T LOSE.**

Train for a Marathon, Not Sprints

- Most self-aware successful people tend to acknowledge the role of perseverance, timing and luck in their eventual success
- Writing meaningful code in reality takes a LONG time
 - If your code becomes legacy you win...it generally means it is useful!
- Careful mistaking that you prove the reverse with the speed code effort using the proof of idea by PoC aim of MVP development, hackathons, market effects, etc.
- Code can be like wine, it can age well, but only under the right conditions and processes, take UNIX as an example. Conversely code can be like milk and spoil quickly, especially if exposed to unstable dependencies and environments

A man with brown hair and glasses, wearing a dark shirt, is shown from the chest up. He has a thoughtful expression, resting his chin on his right hand. The background is a plain, light-colored wall.

**TIMING, PERSEVERANCE,
AND TEN YEARS
OF TRYING WILL EVENTUALLY
MAKE YOU LOOK LIKE
AN OVERNIGHT SUCCESS.**

-BIZ STONE

Gear Up

- Computer – buy the best system you can always
 - I know we are students but really this is like being a musician please buy a good instrument!
- Focus on I/O
 - Disk no, SSD yes
 - Multiple monitors or single wide monitor
 - Avoid cheap pointing devices
 - Keyboard (see dedicated slide)
- Connectivity
 - AirCards, WiFi access plans, VPN software if you must use public access points, fiber if you can get it

Typing

- Being a code programmer requires you to be a good (fast and accurate) typist
 - <http://blog.codinghorror.com/we-are-typists-first-programmers-second/>
- Given the importance of typing
 - Get a good keyboard see next slide
 - Learn to type
 - Try to keep your hands on the keyboard at all times
 - Translation: Learn key commands, expanders, and build finger muscle memory
 - Window management, text expansion examples
 - Skeptical but strong opinions also surround possibility of changing to Dvorak keyboard -
https://en.wikipedia.org/wiki/Dvorak_Simplified_Keyboard#Comparison_of_the_QWERTY_and_Dvorak_layouts

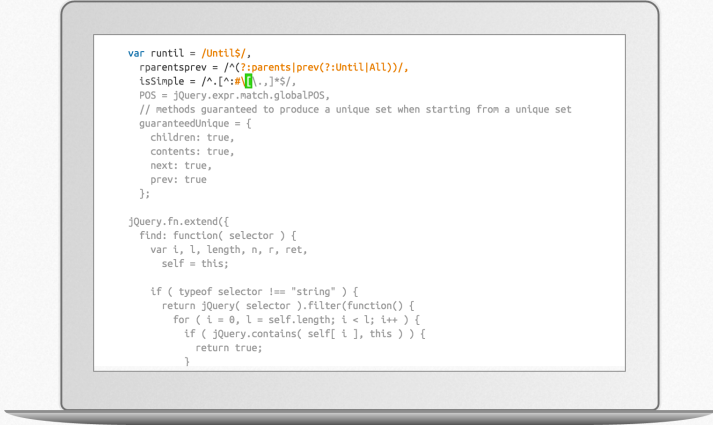
Good Keyboards

- A mechanical keyboard is widely thought as the way to go
 - <http://blog.codinghorror.com/the-keyboard-cult/>
 - <http://www.tested.com/tech/accessories/460198-you-should-use-mechanical-keyboard/>
- <https://codekeyboards.com/>
- <http://www.daskeyboard.com/>
- Check e-Bay to see how much vintage mechanicals still fetch

Interesting Launch

[typing.io](#)[plans & pricing](#)[lessons](#)[sign in](#)

Typing Practice for Programmers



```
var runtil = /until$/,
    rparentsprev = /^(?:parents|prev(?:until|All))/,
    isSingle = /^[\^$@!~\.\,\*\+\-\=\&\|\:\;\'\`\<\>\(\)\{\}\[\]\<\/>]*$/,
    POS = jQuery.expr.match.globalPOS,
    // methods guaranteed to produce a unique set when starting from a unique set
    guaranteedUnique = {
      children: true,
      contents: true,
      next: true,
      prev: true
    };

jQuery.fn.extend({
  find: function( selector ) {
    var i, l, length, n, r, ret,
        self = this;

    if ( typeof selector !== "string" ) {
      return jQuery( selector ).filter(function() {
        for ( i = 0, l = self.length; i < l; i++ ) {
          if ( jQuery.contains( self[ i ], this ) ) {
            return true;
          }
        }
      });
    }
  }
});
```

Practice typing the awkward characters in code.

No drills – type through open source code in JavaScript, Ruby, C, C++, Java, PHP, Perl, Haskell, Scala, and more.

Eliminate the mistyped keys delaying every edit-compile-test iteration.

It's free, just sign in.

[Sign in with Google](#)

[or try the demo](#)

<https://typing.io/>

“Tools”

- Clearly we should certainly select good tools
- Rule: A good _____ does best with good tools, but good tools don't make good _____s.
- We should not be completely resistant to the evaluation of new tools after we mastered a tool though
- Tool analysis of course has to happen with Fact #5 and #6 firmly held
- Tool analysis also should be in light of specific need as opposed to generalized solution
- The newness or popularity of a tool is a far from perfect indicator of the value or efficacy of the tool

Tool Belts and Boxes

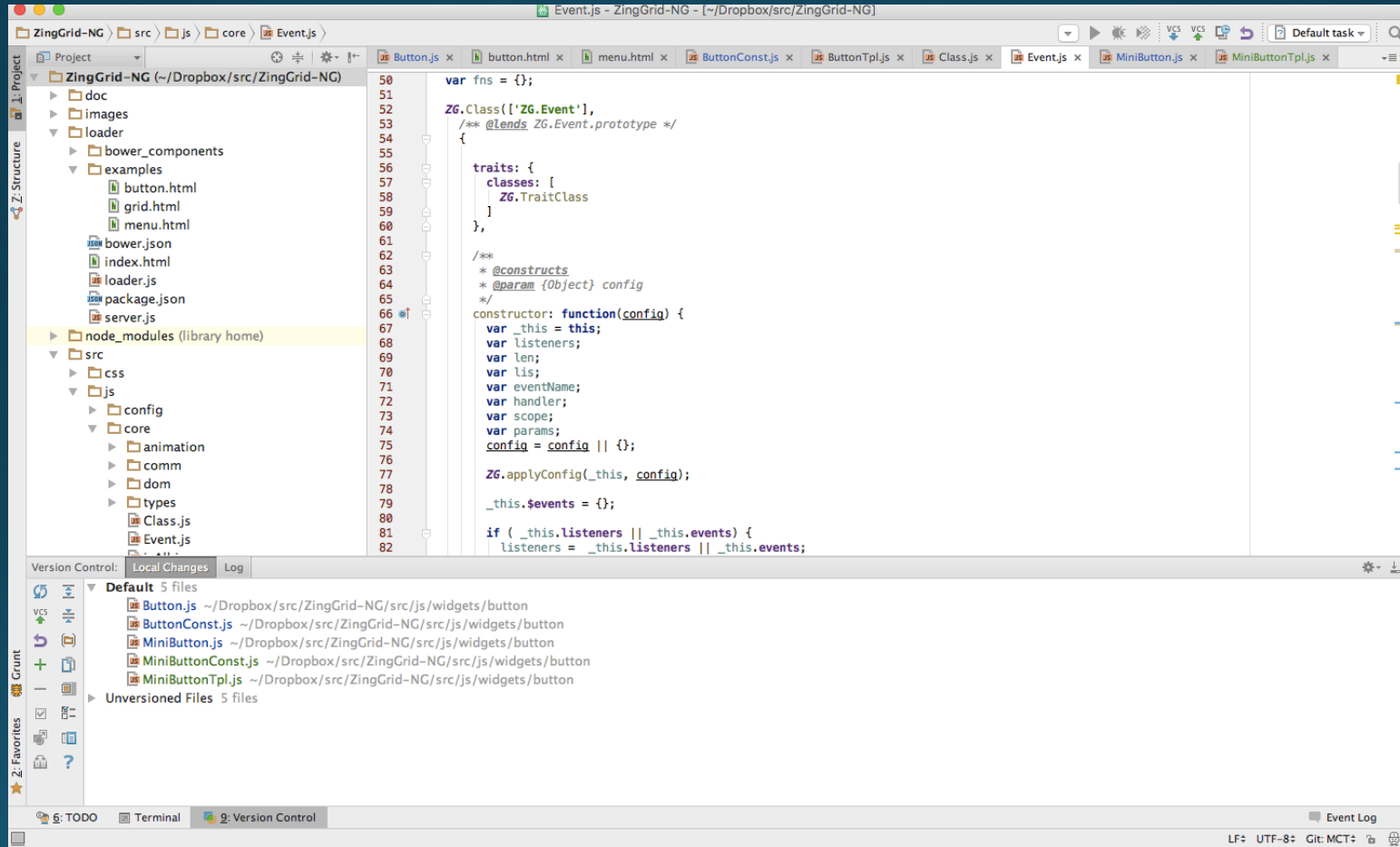
- Given the programmer tendency to look for general solutions we see in industry an over emphasis on generalized tools or a search for the “one true tool”
 - Silver Bullets
 - LoTR Fixations
- Tools have trade-offs like anything else, best to have the correct tool for the job than a single tool for all jobs
 - Must acknowledge that too many tools leads to being good at nothing both because of lack of heavy use or tool compromise
- My pragmatic metaphor is the tool box or tool belt then with some ‘go to’ tools within



Editors

- Compare to hammers - there is a range
 - Finishing, General/Claw, Framing, etc.
- Simple Text Editor
 - vi, emacs, Sublime, etc.
- Light Weight IDE
 - Coda
- Full IDE
 - Visual Studio, IntelliJ (WebStorm), Eclipse
- Emphasising
 - Mouse or Keyboard
 - Mode or not moded
 - Manual or Auto(complete)
 - Single Experience or Multi-Windowed/App Experienced

WebStorm



<https://www.jetbrains.com/webstorm/>

Class Notes on WebStorm

- Purposeful fixed (likely appropriate) choice
 - JavaScript focused (so is our project)
 - Web Dev focused (so is our project – at least mostly)
 - Supports SE workflows within (Git, CodeReview, Refactor, Ticketing, Building, etc.) as opposed to multi-tooling which should clear gains once understood
 - Refactoring tools are pretty good and that is clearly going to be a big point of the course
- Participation Homework Choice #2 – Spend some time between now and next Tuesday and learn some features that you are motivated by in WebStorm. Be prepared to explain and have me try it in class

Coming Up in CSE 112

- Participation Homework Review
- Professor's In-Class Tour of WebStorm*
- Review of Projects from Gold and Blue Team's efforts from last year
- Discussion of possible approaches going forward
- Peter's pressure to get going