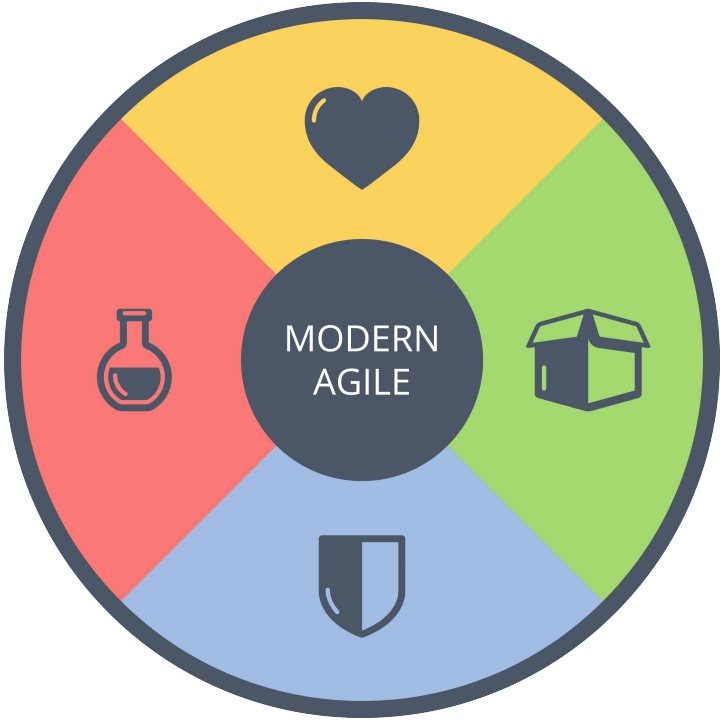
A.B.L.E.

Always Be Learning and Experimenting

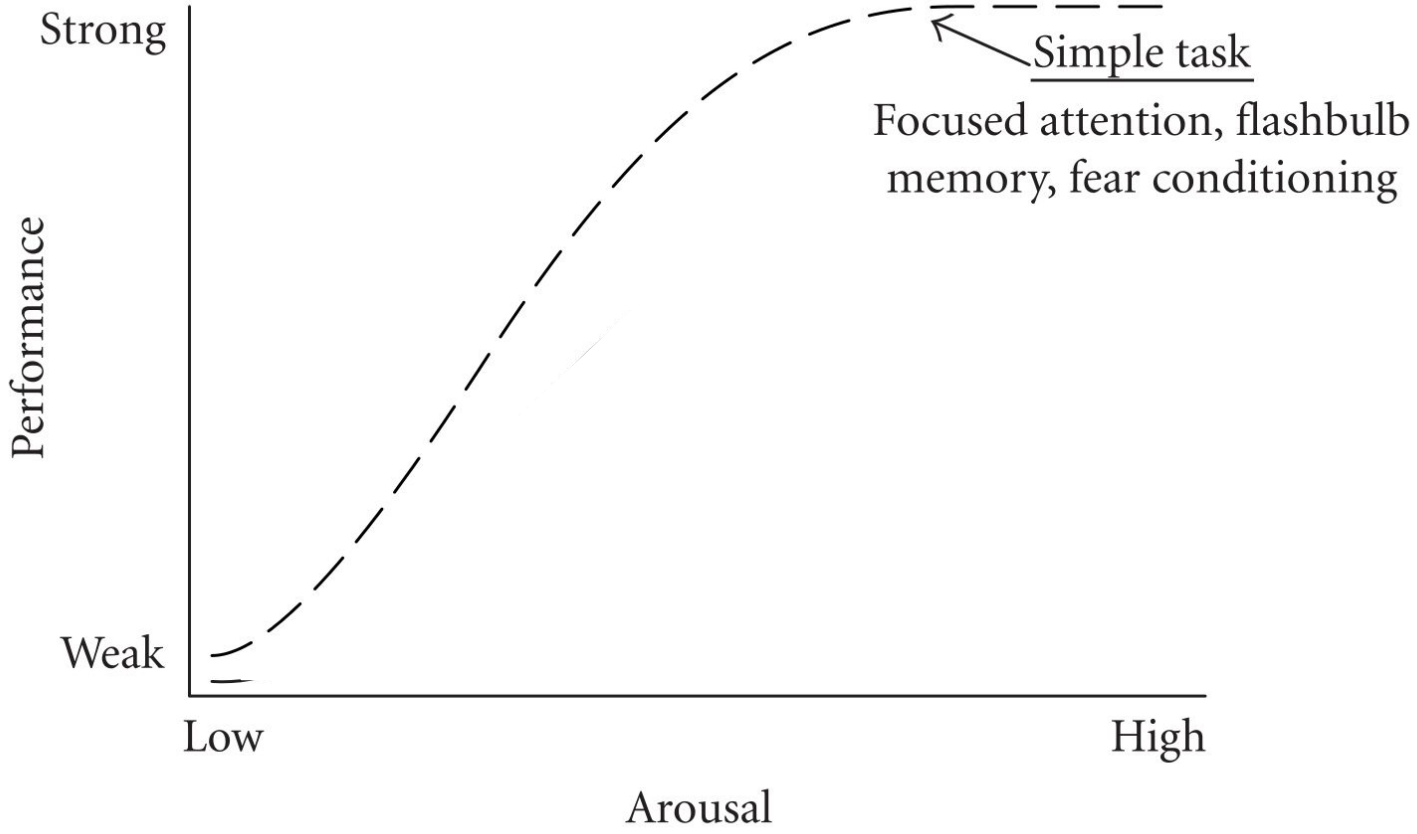


modernagile.org

Make Safety A Prerequisite

Experiment and Learn Rapidly

Principles

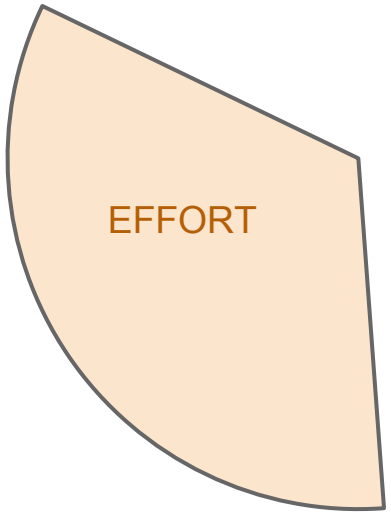


Wikipedia: Yerkes-Dodson Law

$$\text{Productivity} = \frac{E(\text{applied})}{E(\text{required})}$$


UNCERTAINTY
EFFORT
RISK

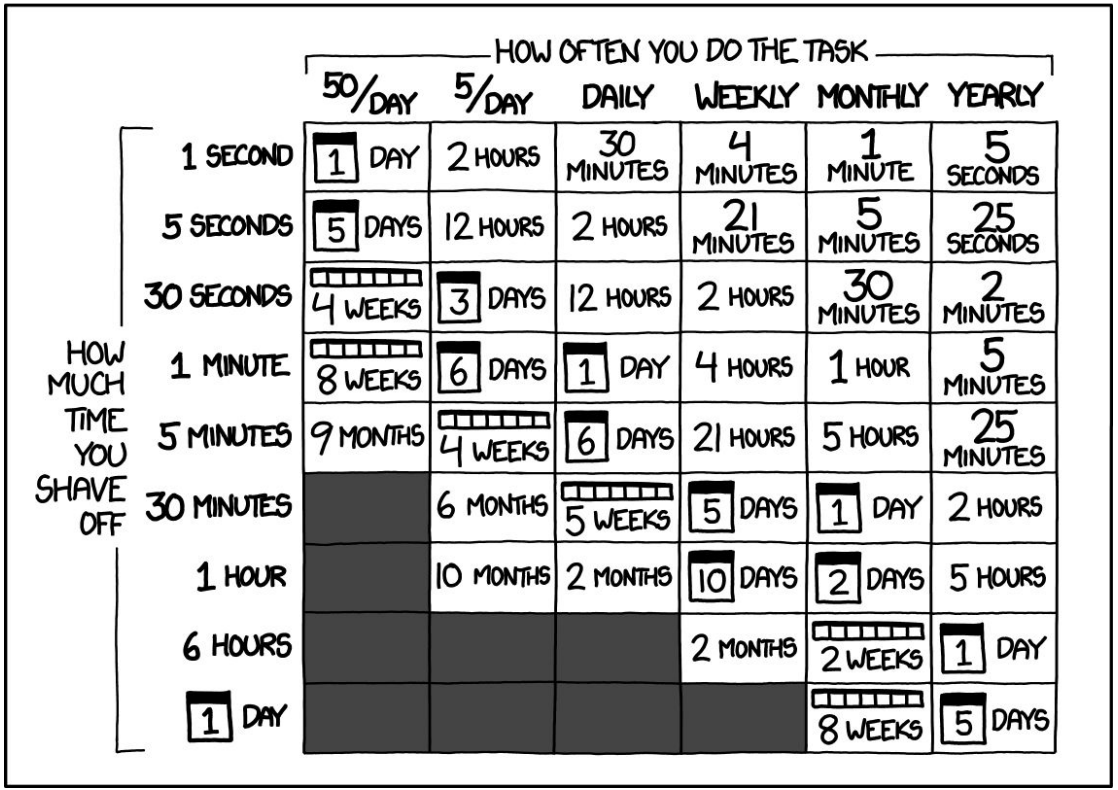
Low risk?
Low uncertainty?



NOTHING TO LEARN!
WHY WASTE HUMAN EFFORT?
AUTOMATE THIS AWAY!

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

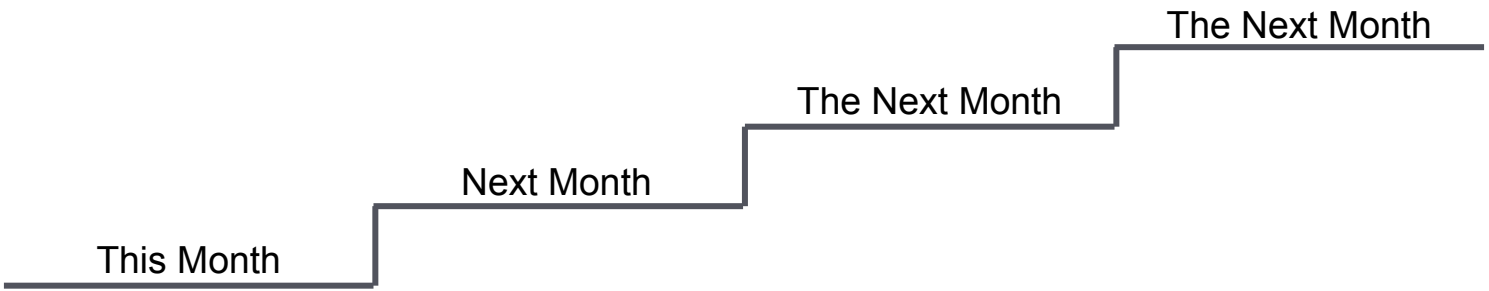
(ACROSS FIVE YEARS)



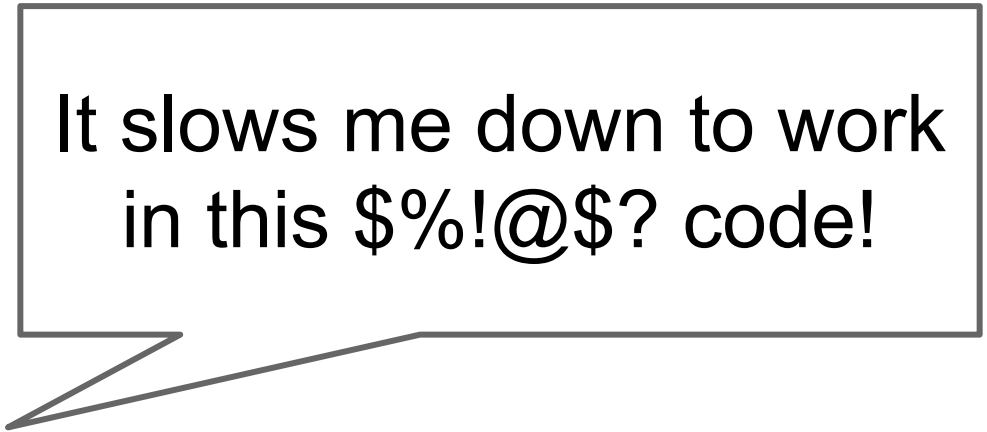
Randall Munroe
xkcd.com/1205/

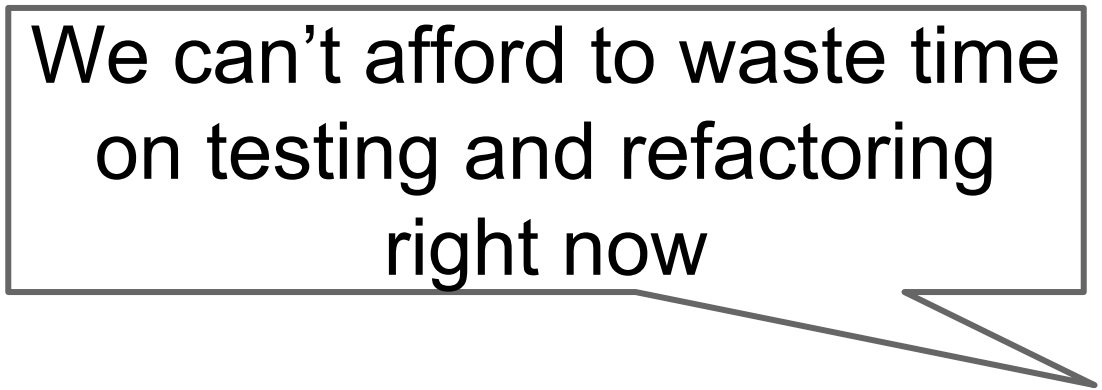
High risk?
High uncertainty?

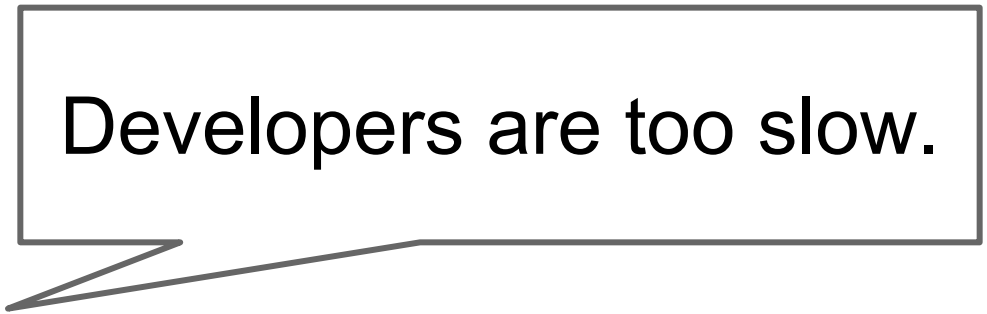
USE MORE BRAINPOWER TO MITIGATE THE KNOWLEDGE GAP!

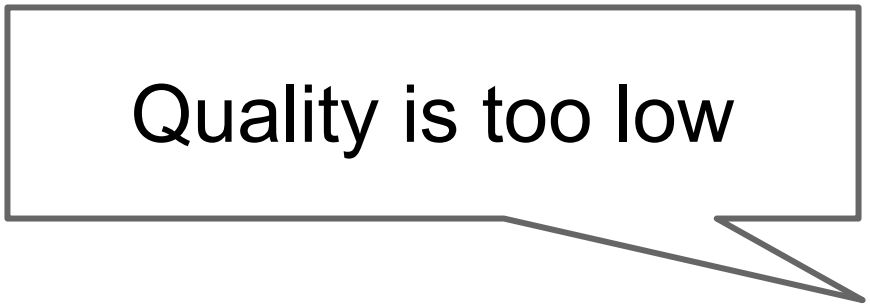

This Month
Next Month
The Next Month
The Next Month

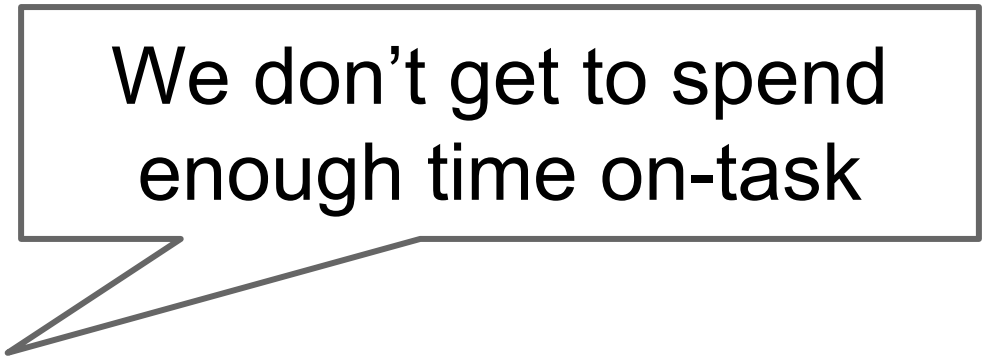
Unquestioned Surface Problems

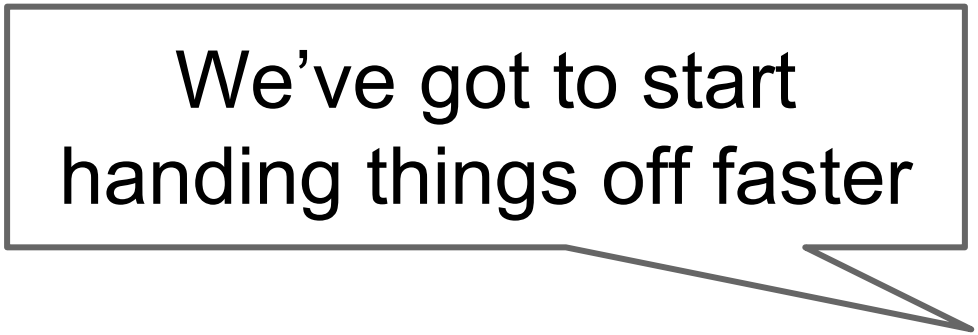

It slows me down to work
in this $%!@$? code!

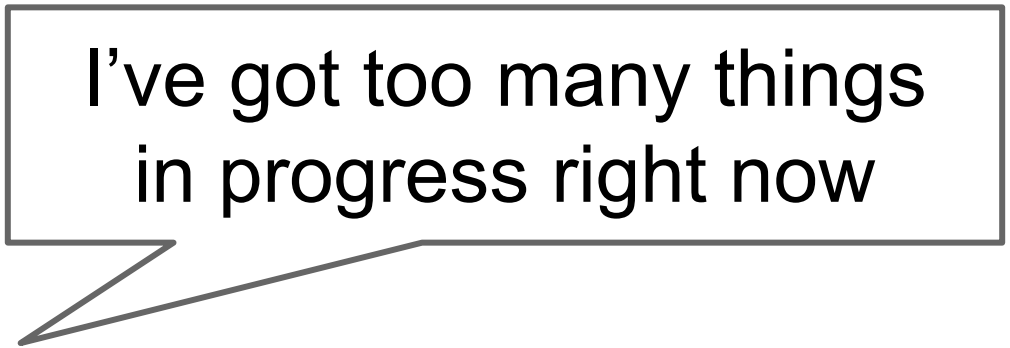

We can’t afford to waste time on testing and refactoring right now

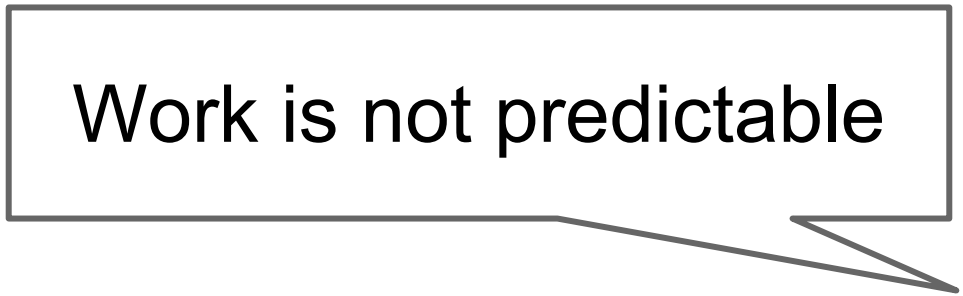

Developers are too slow.

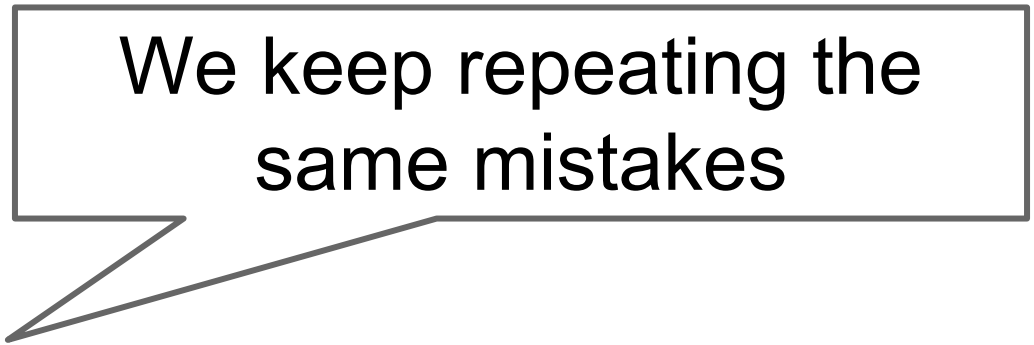

Quality is too low

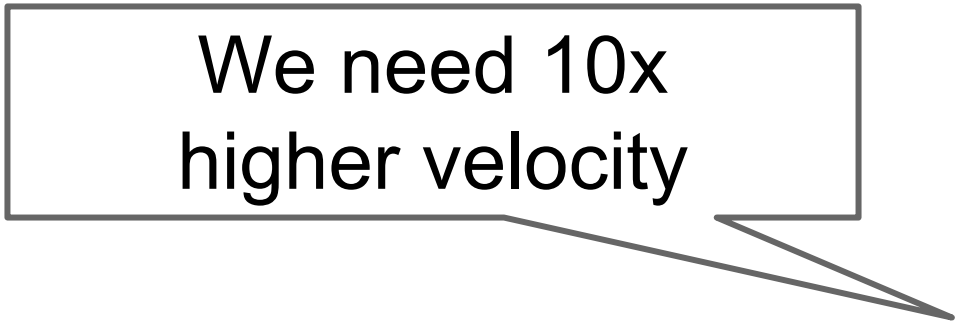

We don’t get to spend
enough time on-task

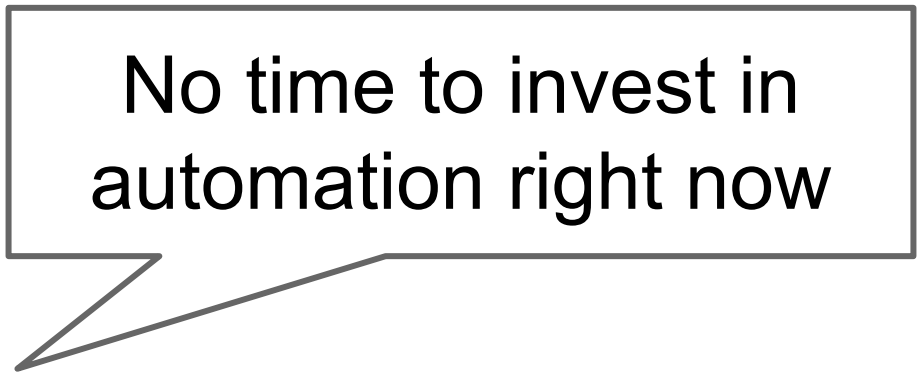

We’ve got to start
handing things off faster

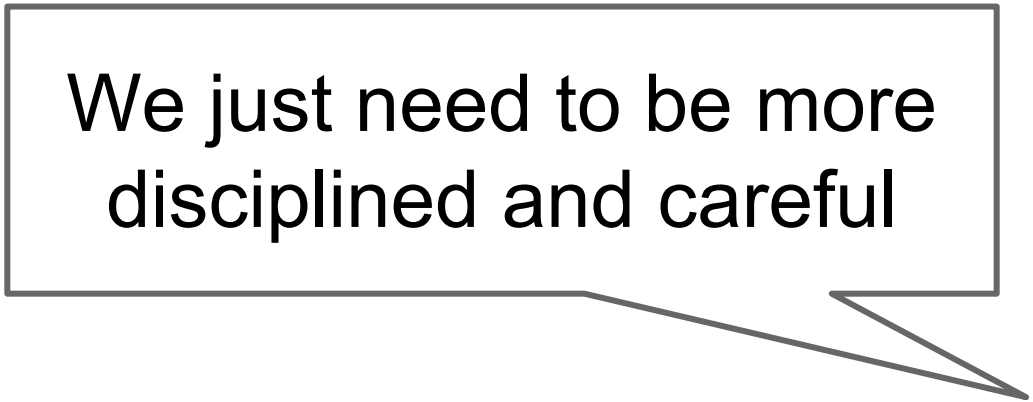

I’ve got too many things
in progress right now

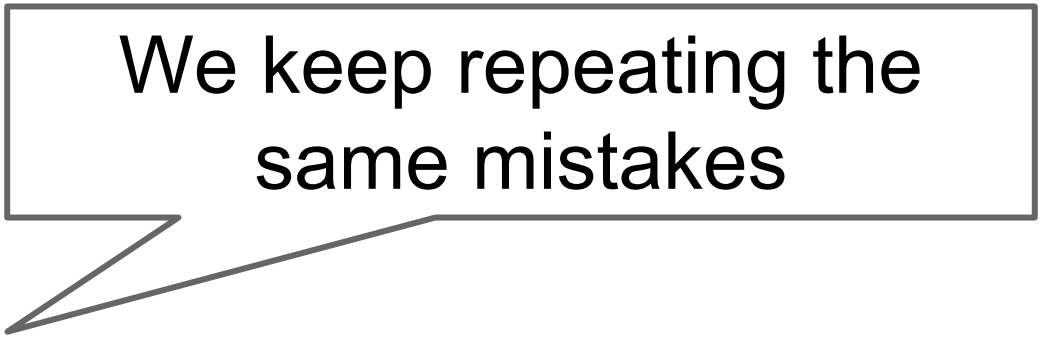

Work is not predictable

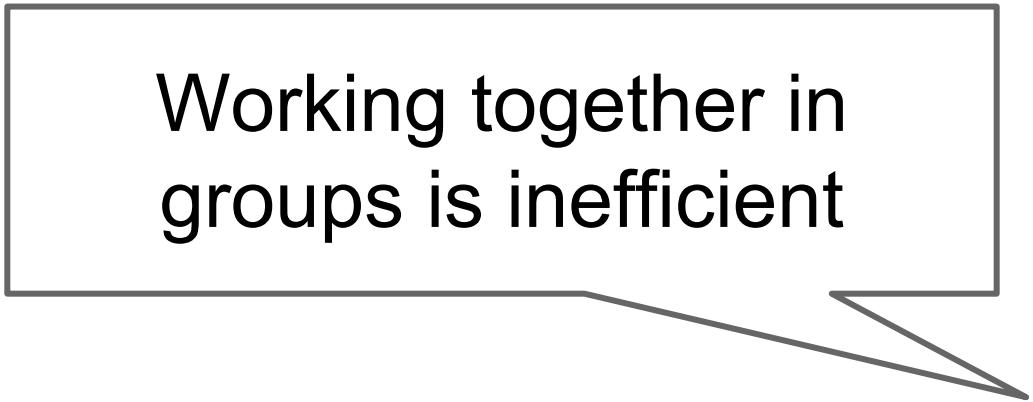

We keep repeating the
same mistakes

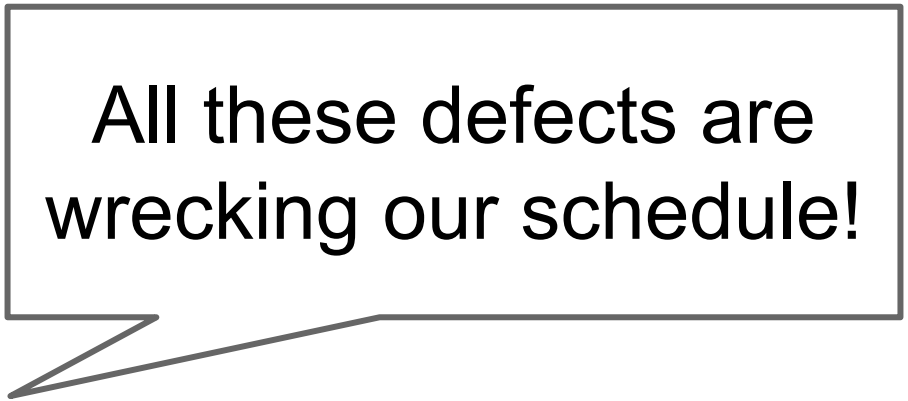

We need 10x
higher velocity

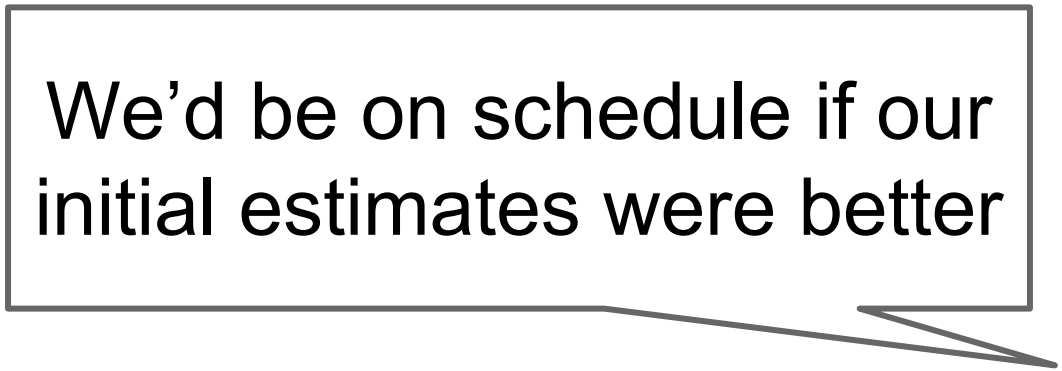

No time to invest in
automation right now

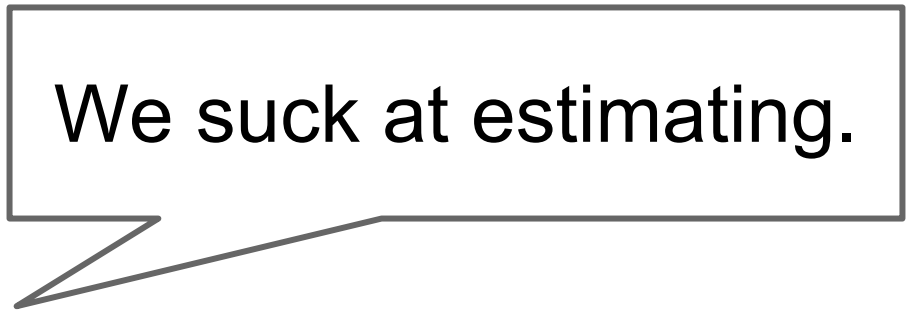

We just need to be more disciplined and careful

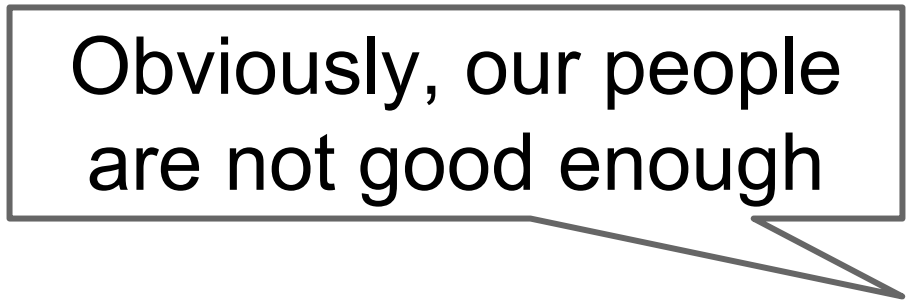

We keep repeating the
same mistakes


Working together in
groups is inefficient

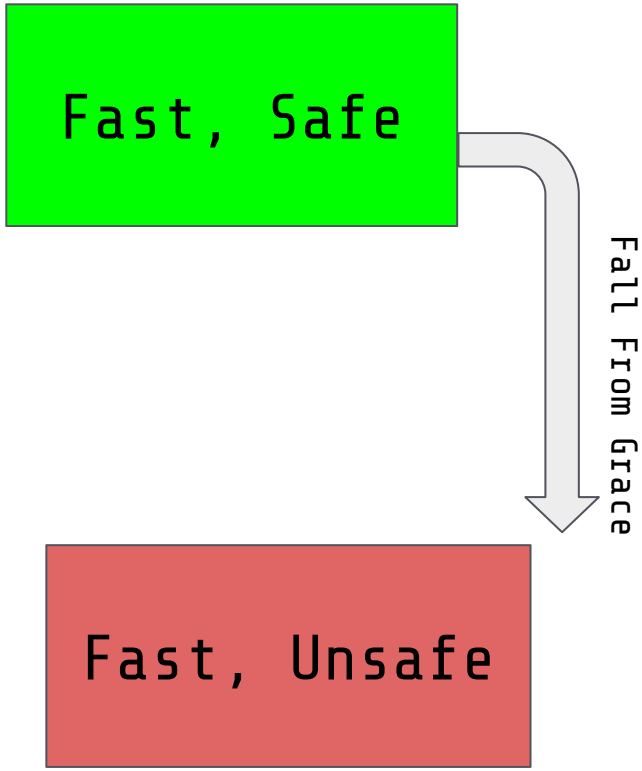

All these defects are
wrecking our schedule!

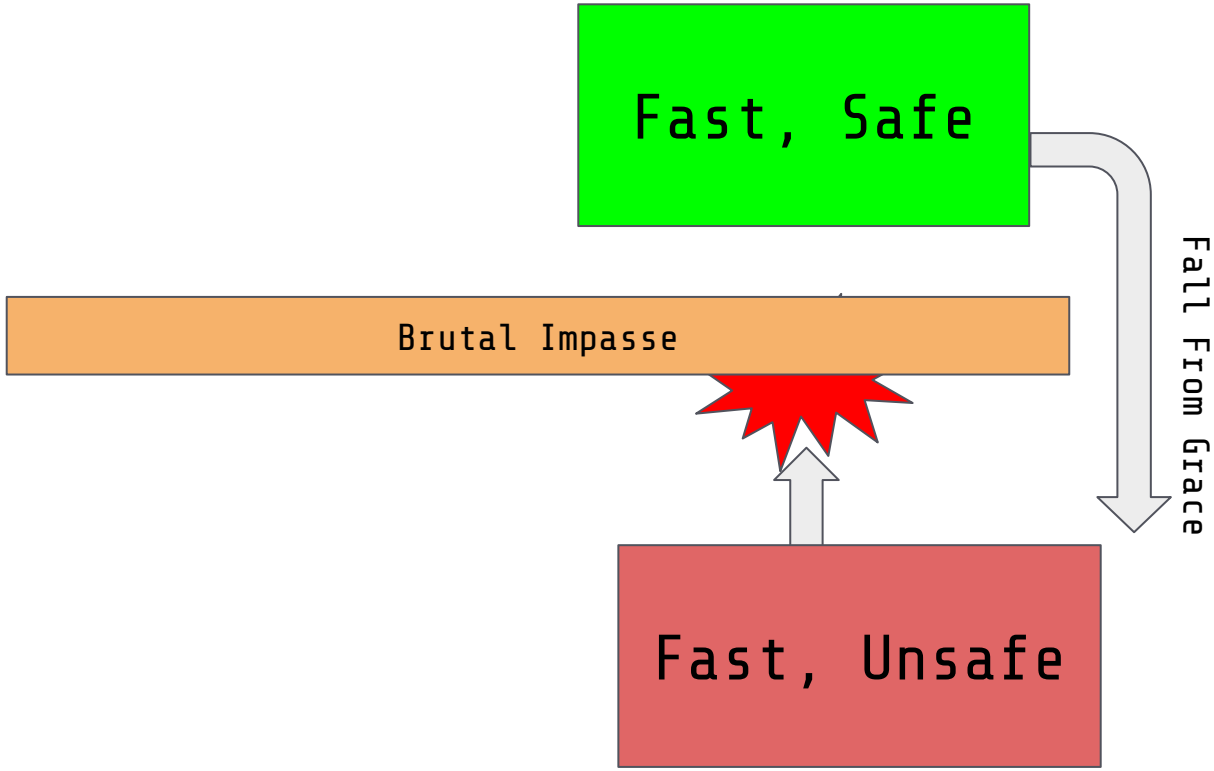

We’d be on schedule if our initial estimates were better

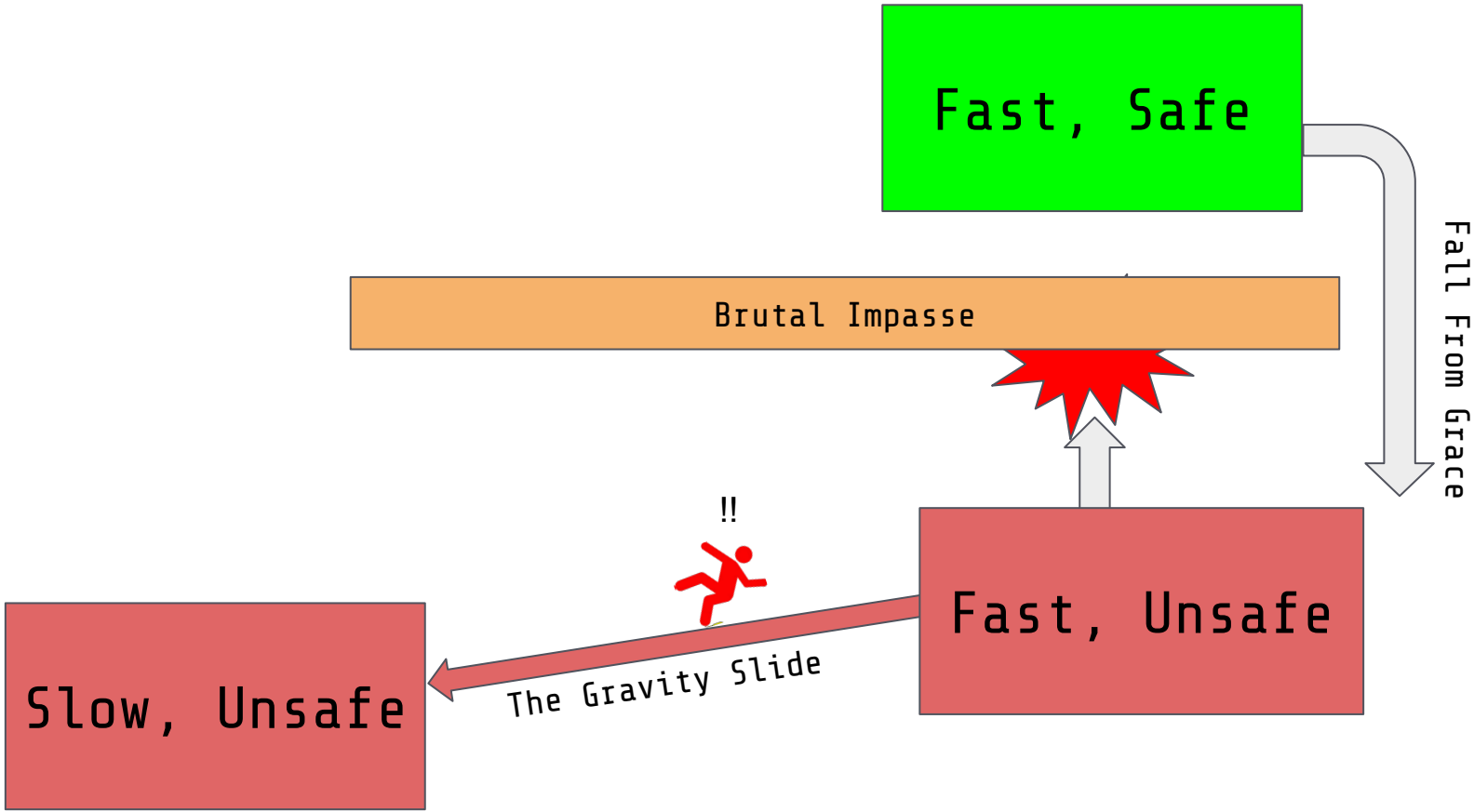

We suck at estimating.

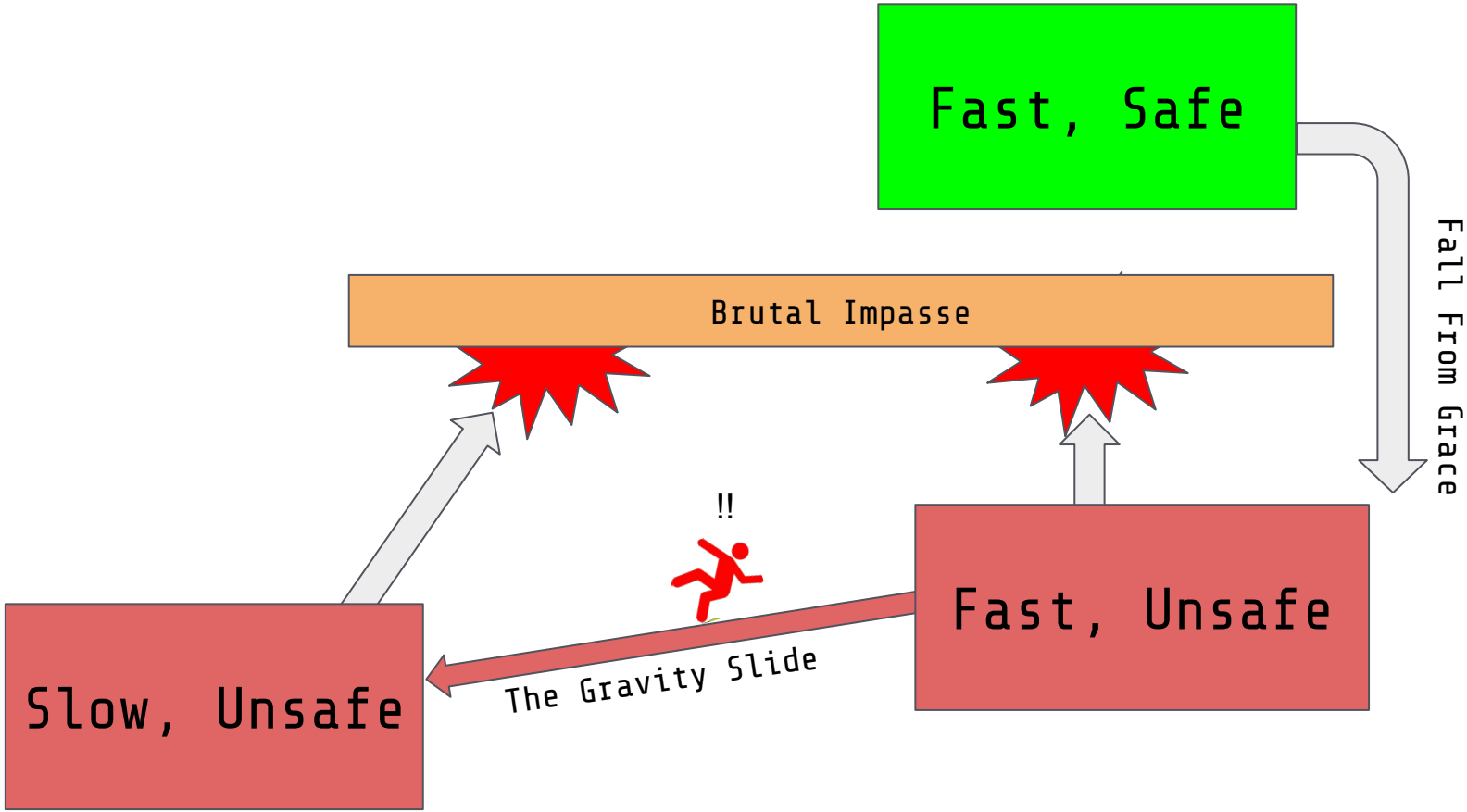

Obviously, our people
are not good enough

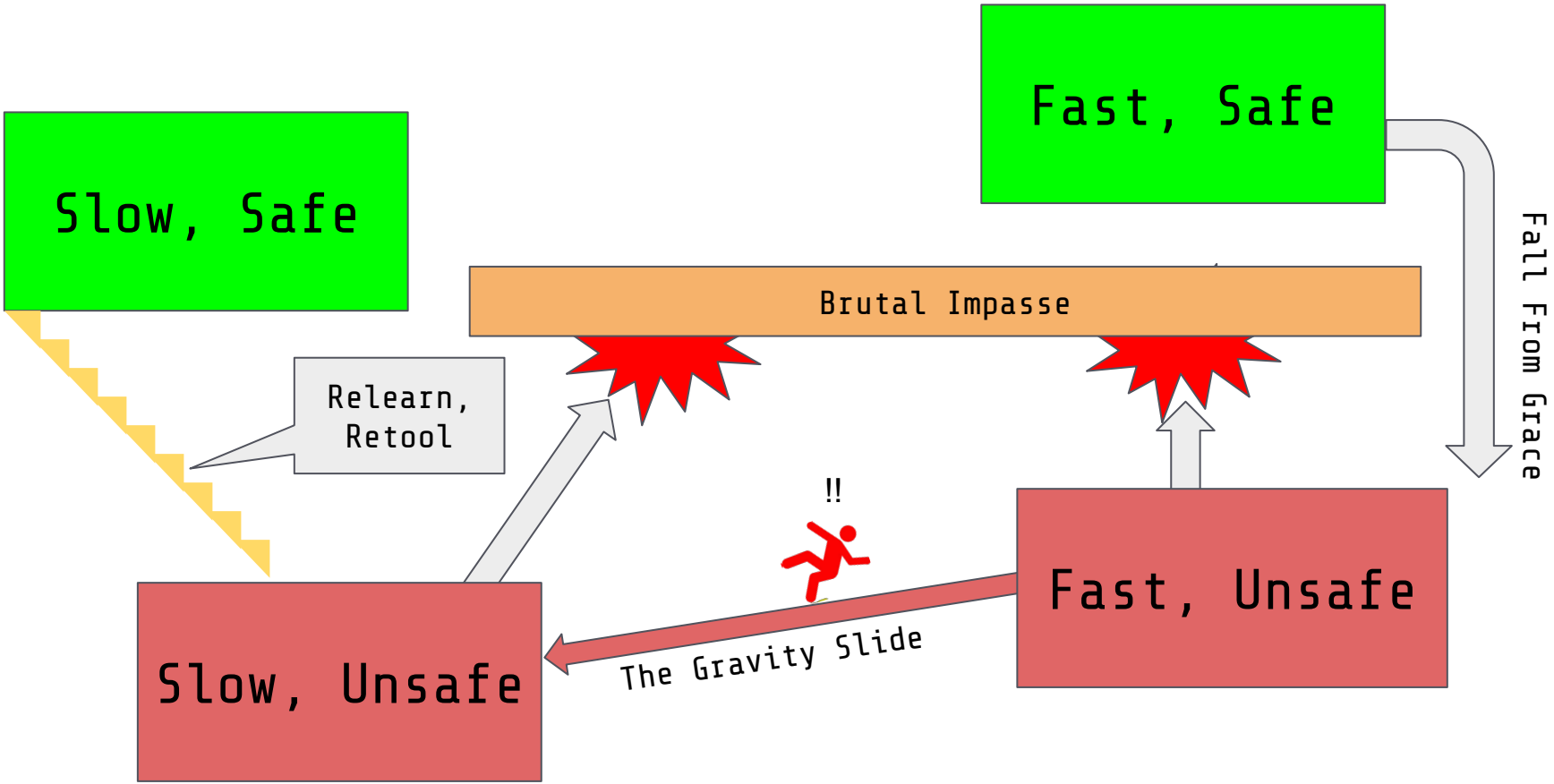
It is too slow.
And too buggy.
And we can work on that.

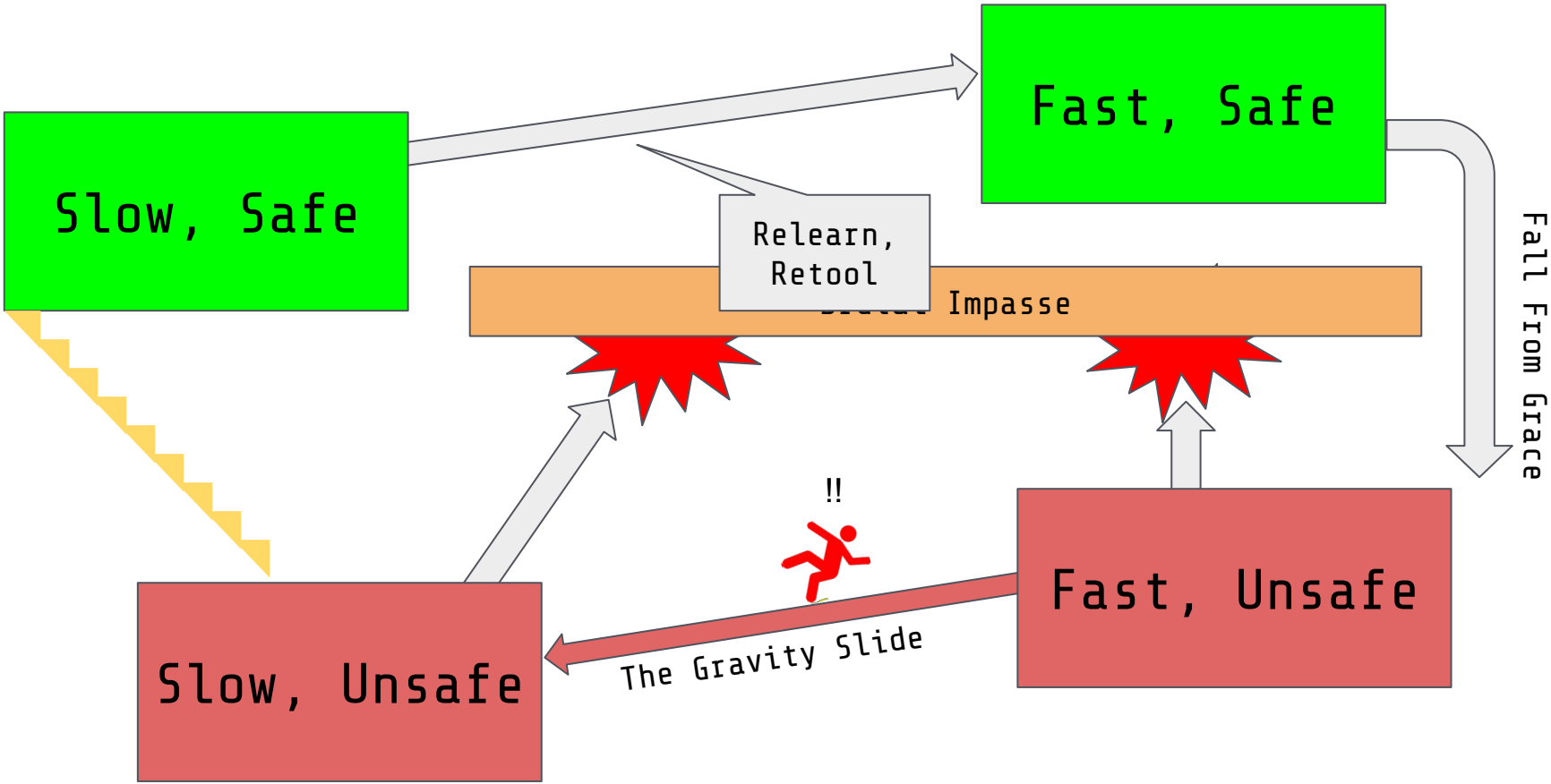

Fast, Safe
Fall From Grace
Fast, Unsafe

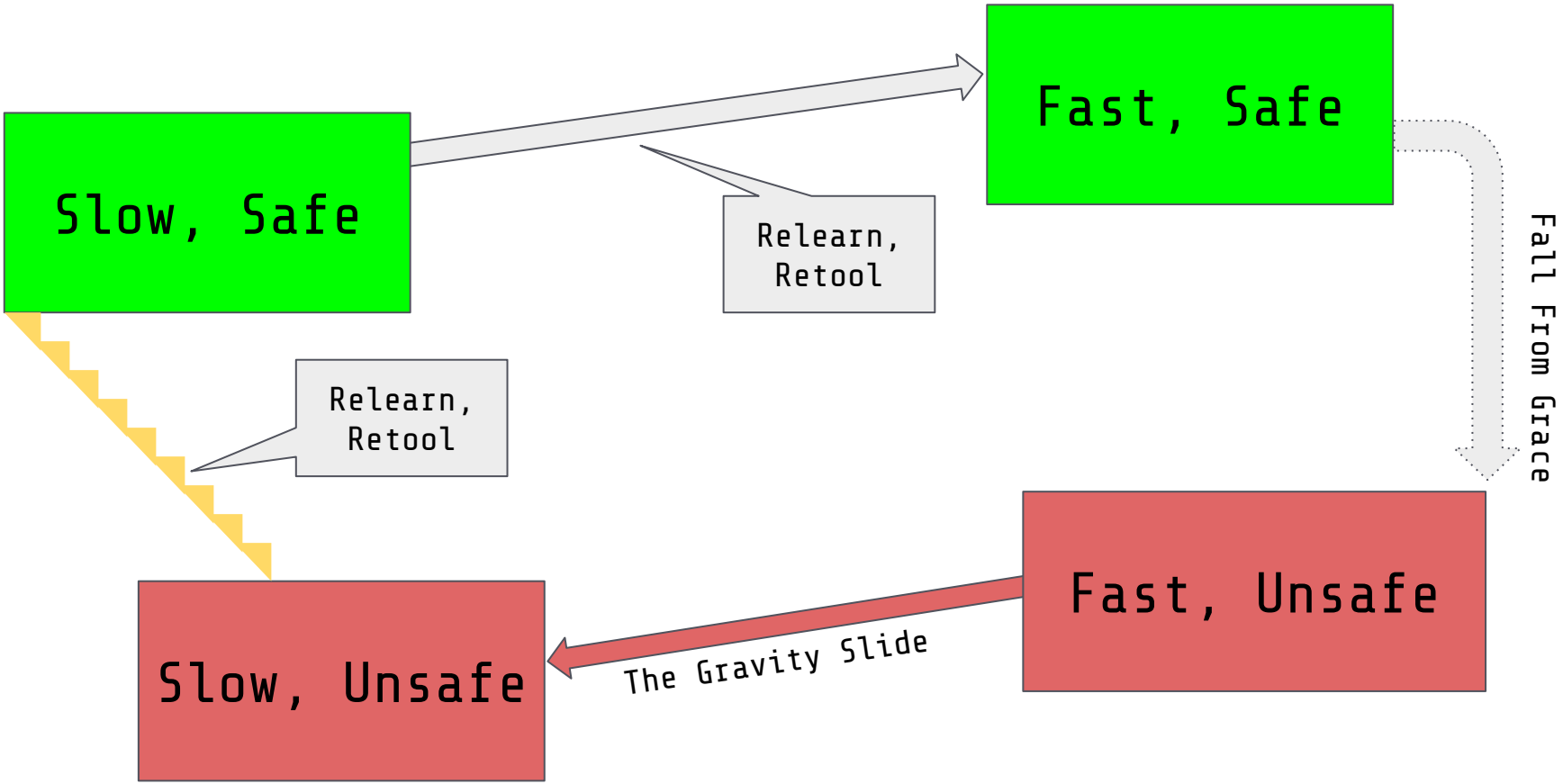

Fast, Safe
Brutal Impasse
Fall From Grace
Fast, Unsafe

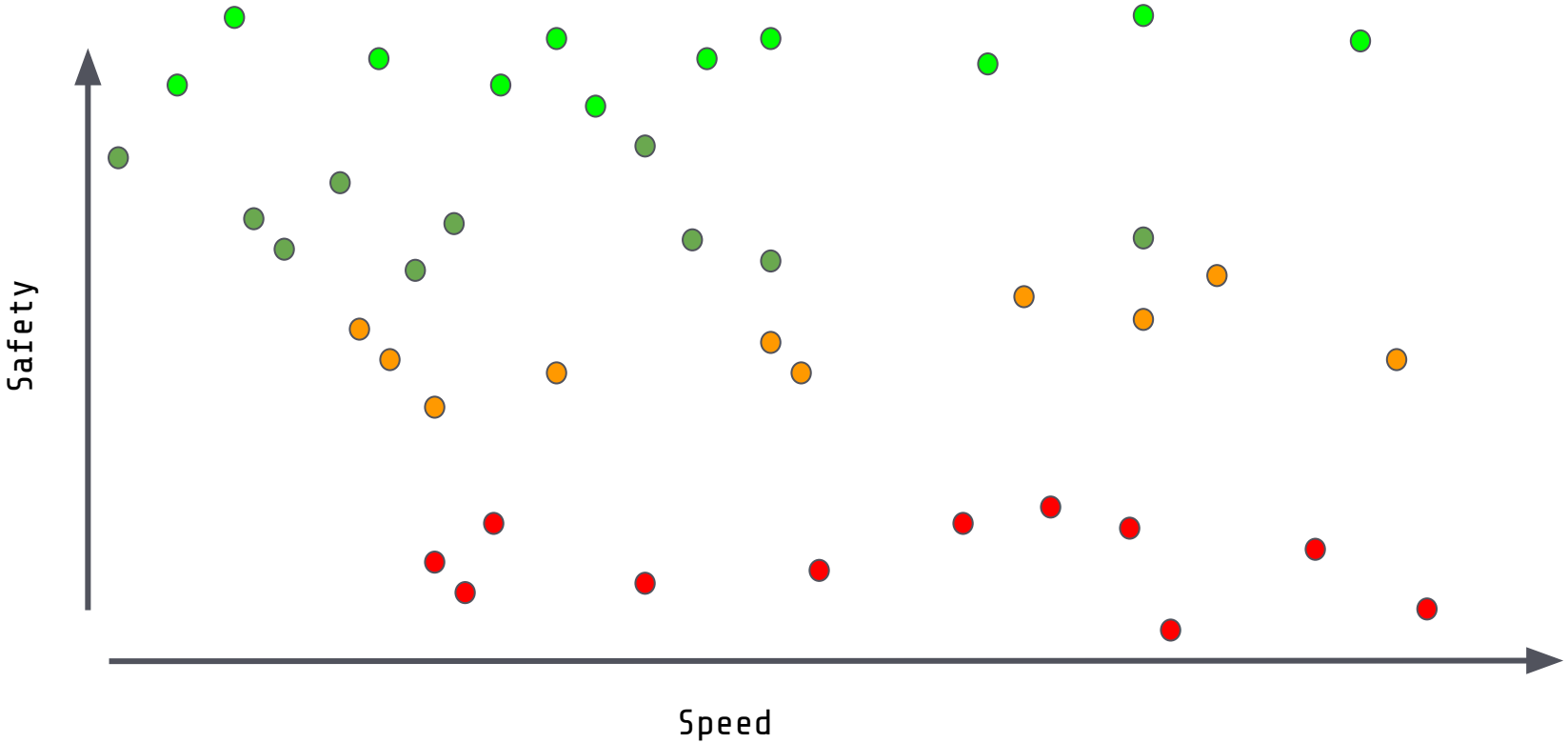

Fast, Safe
Fall From Grace
Brutal Impasse
!!
Fast, Unsafe
The Gravity Slide
Slow, Unsafe

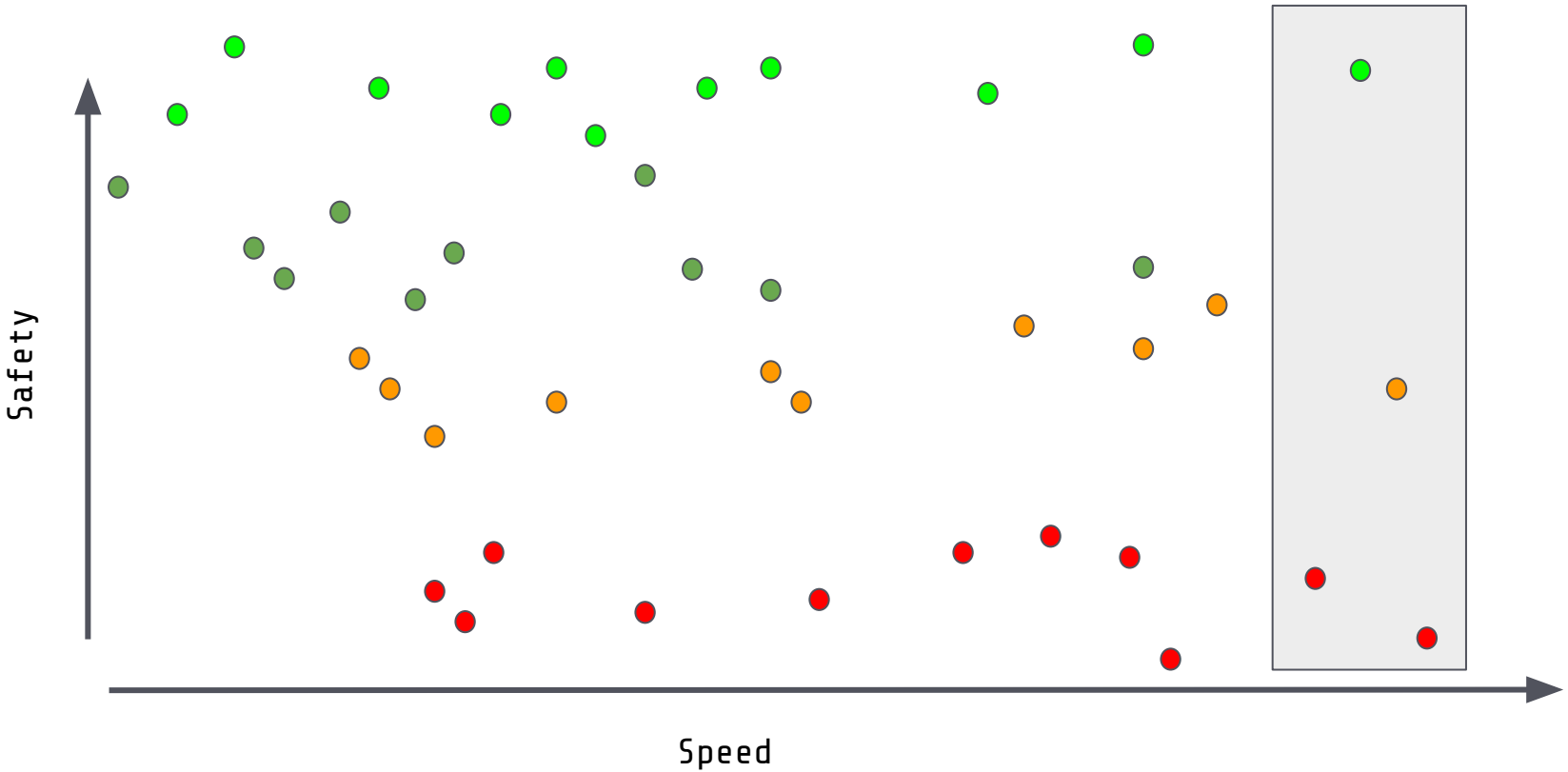

Fast, Safe
Fall From Grace
Brutal Impasse
!!
Fast, Unsafe
The Gravity Slide
Slow, Unsafe

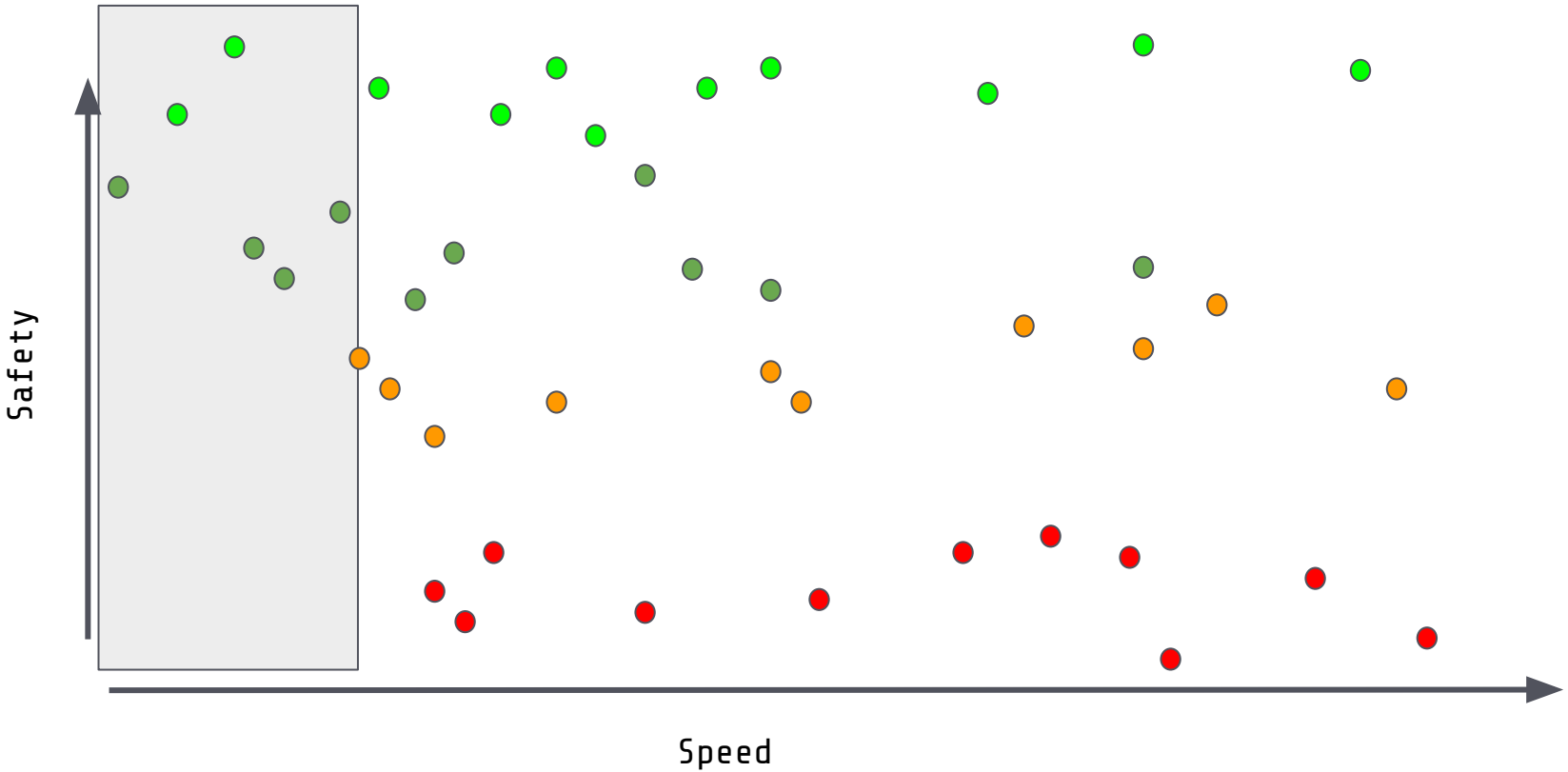

Fast, Safe
Slow, Safe
Brutal Impasse
Fall From Grace
Relearn,
Retool
!!
Fast, Unsafe
Slow, Unsafe
The Gravity Slide

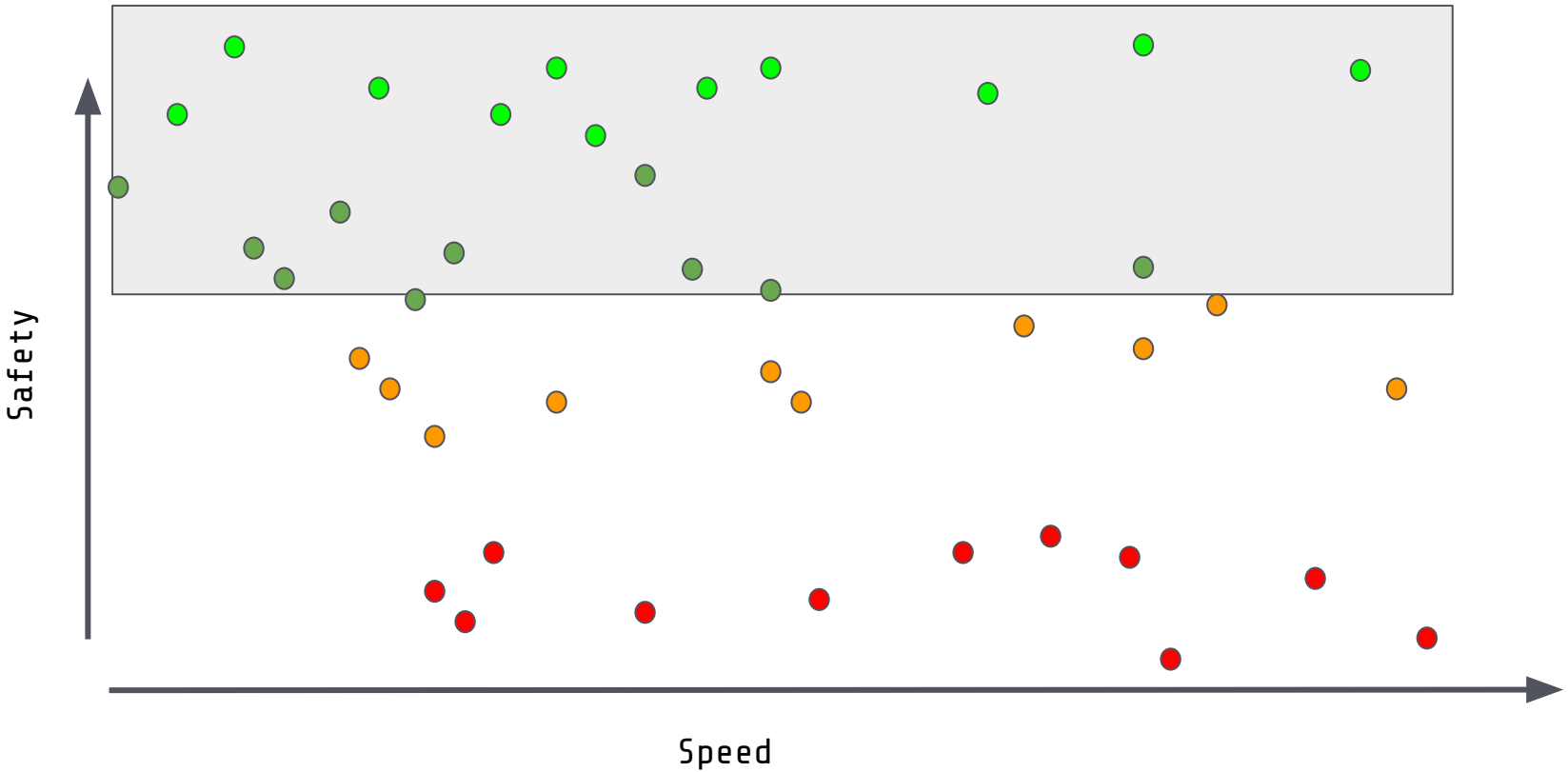

Slow, Safe
Fast, Safe
Relearn,
Retool
Impasse
Fall From Grace
!!
Slow, Unsafe
The Gravity Slide
Fast, Unsafe

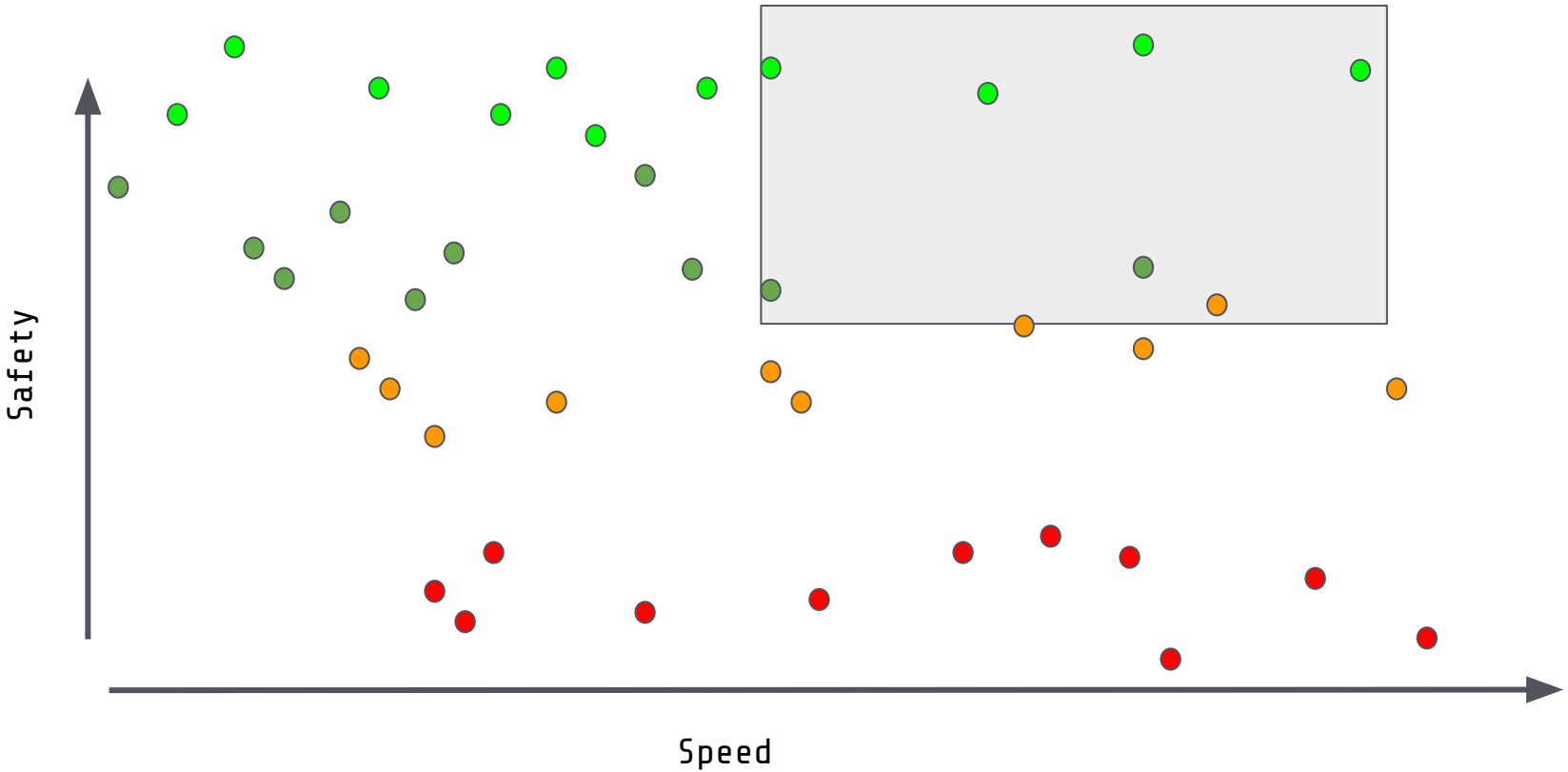

Slow, Safe
Relearn, Retool
Fast, Safe
Fall From Grace
Relearn, Retool
Slow, Unsafe
The Gravity Slide
Fast, Unsafe

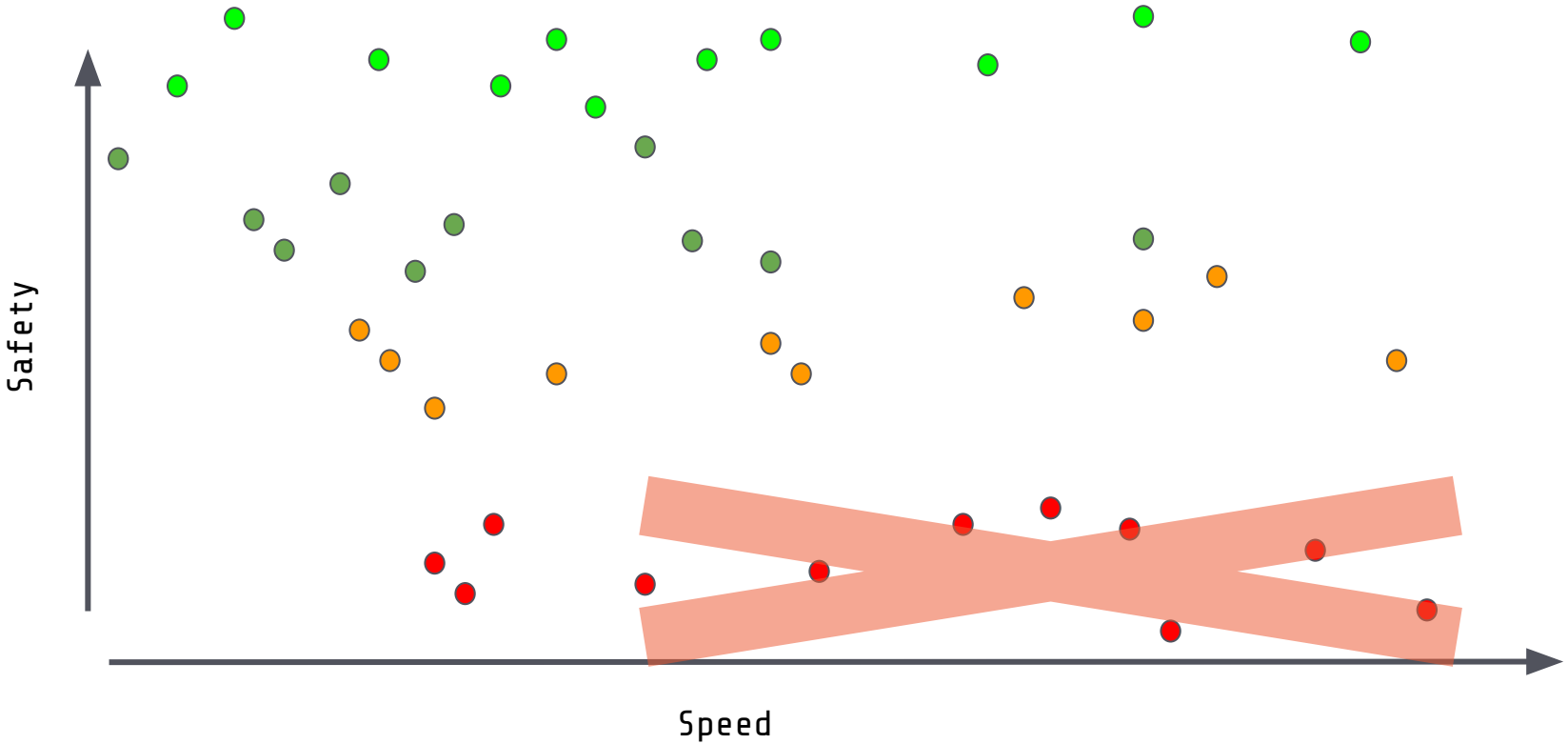

Safety
Speed


Safety
Speed


Safety
Speed


Safety
Speed


Safety
Speed


Safety
Speed

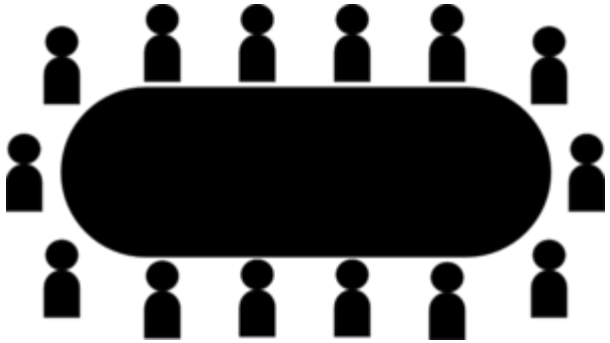
WHAT IS THE REAL WORK?

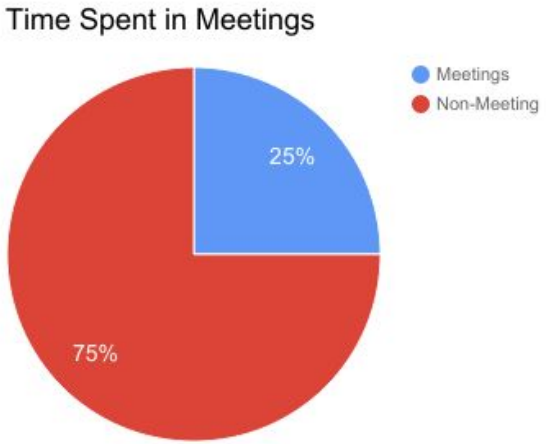
Real Work

(RW)

Bureaucratic Silliness

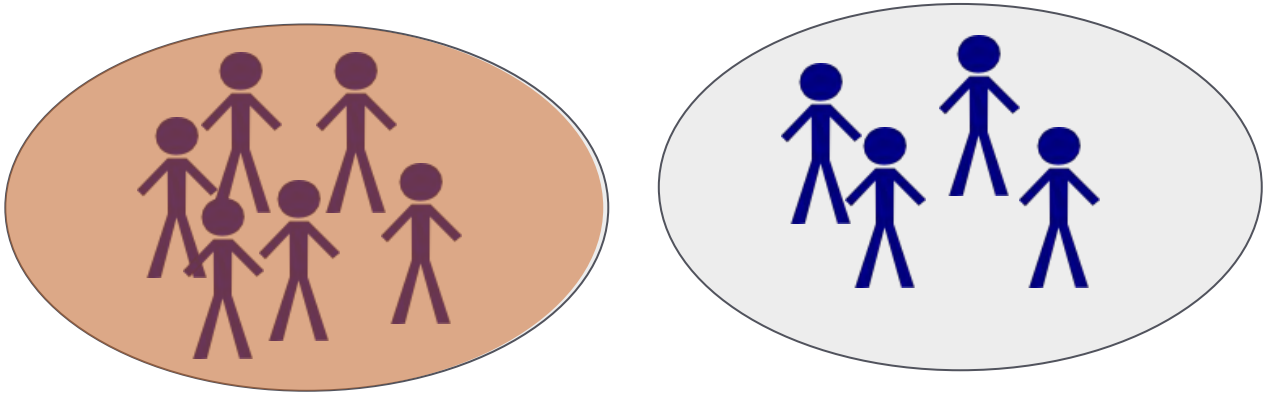
(BS)



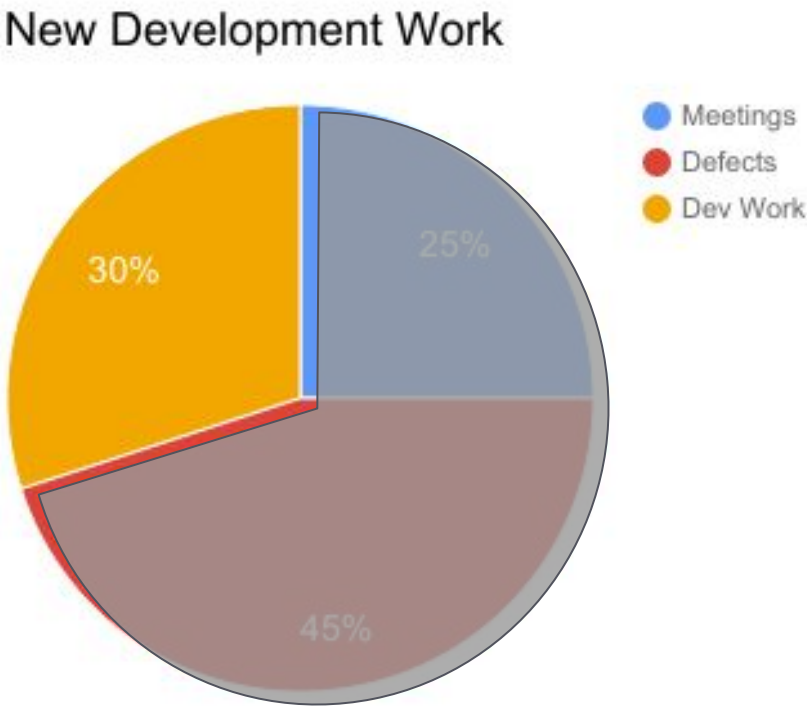


The more junior you are, the fewer. It goes up with experience.

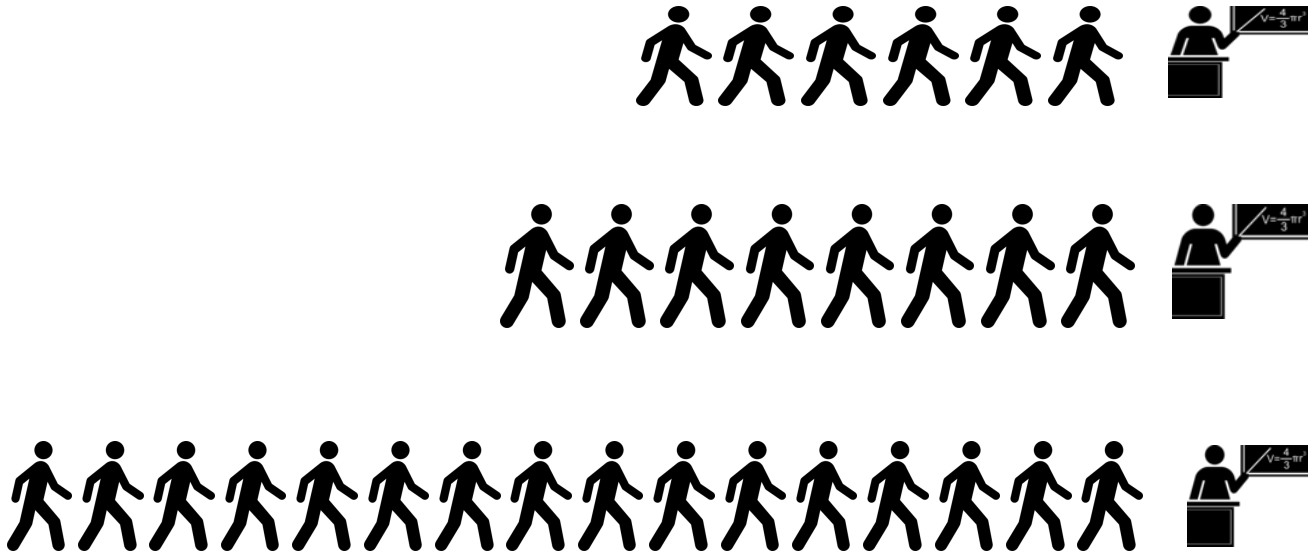
Defect work is 60%(+) of team's total effort



Informal survey


New Development Work
Meetings
Defects
Dev Work
25%
30%
45%

Waiting/Queuing



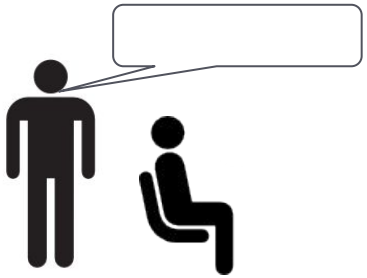
Informal survey

Waiting on Answers

(swapping tasks to tolerate waiting)

Interruptions

(defect, outage, QA "bounce", questions, expert finishes)



Waiting on Machines

(or skipping tasks to avoid waiting)

Undelivered tasks in play per individual:

5

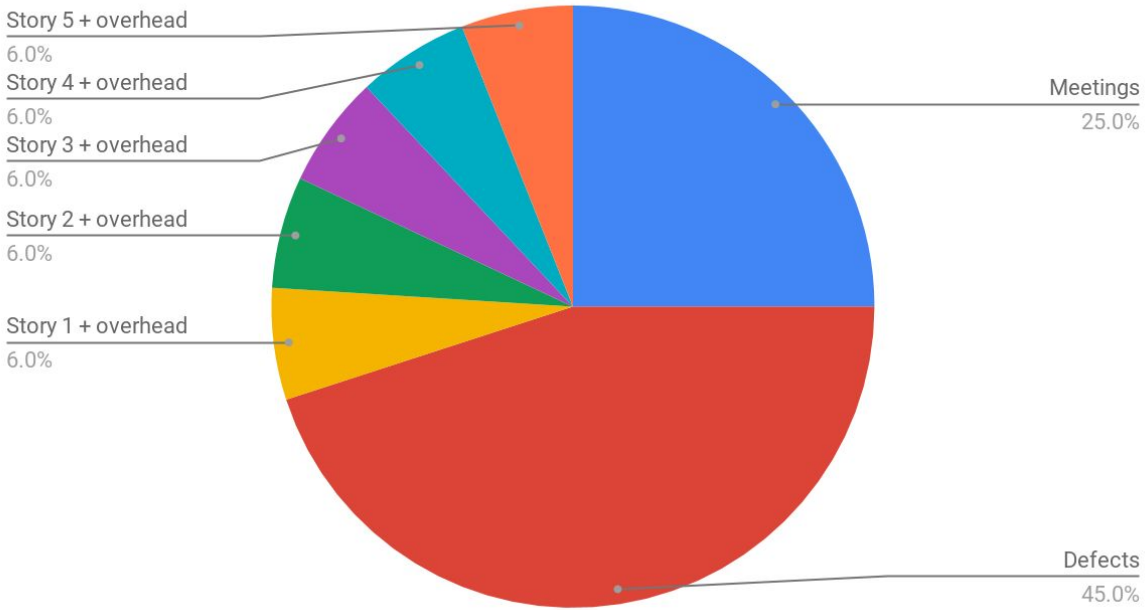
(assumes individual task environment)

Five jobs?

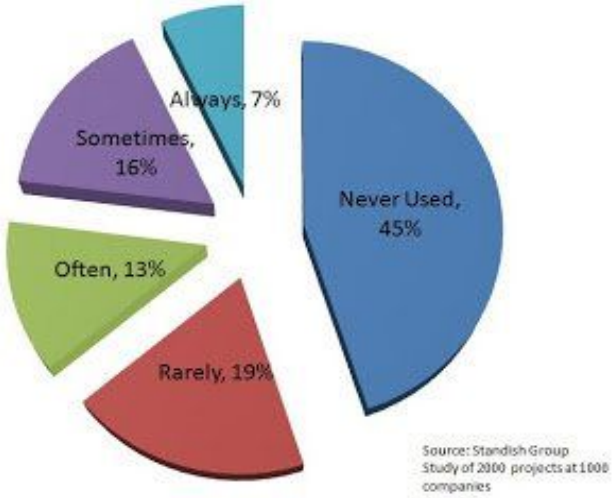
That means when one is getting attention, the other 4 are not.

One job holding four others hostage?

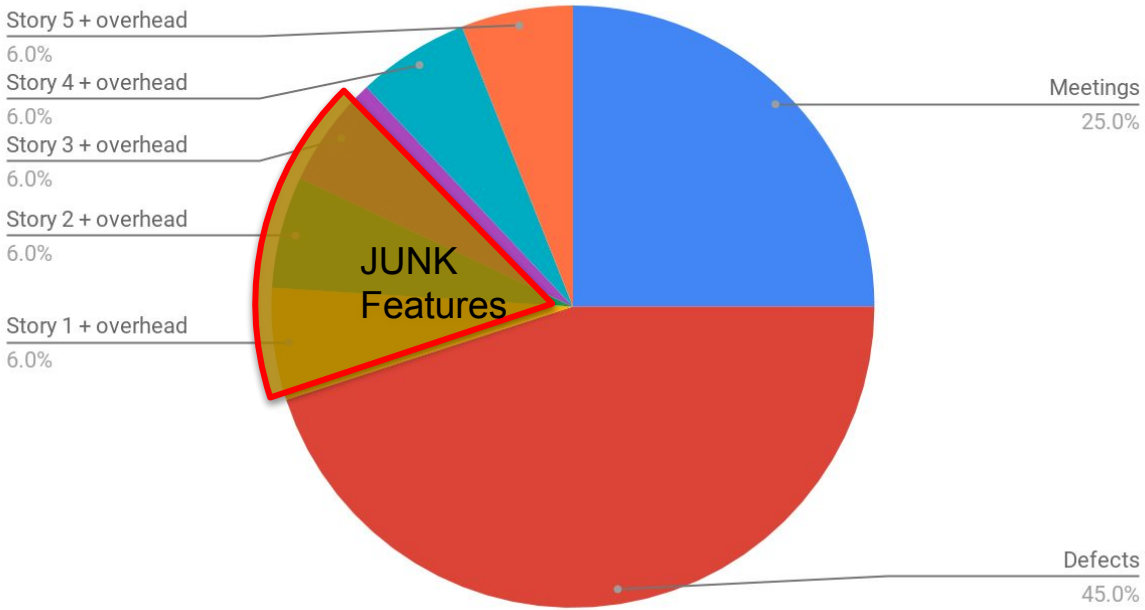
Time Slices


Story 5 + overhead
6.0%
Story 4 + overhead
6.0%
Story 3 + overhead
6.0%
Story 2 + overhead
6.0%
Story 1 + overhead
6.0%
Meetings
25.0%
Defects
45.0%

Roughly 64% of delivered features are seldom/never used.



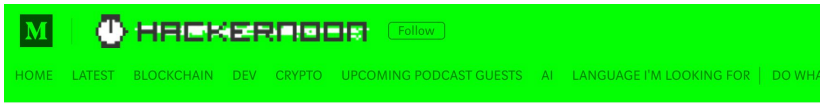
Time Slices


Story 5 + overhead
6.0%
Story 4 + overhead
6.0%
Story 3 + overhead
6.0%
Story 2 + overhead
6.0%
Story 1 + overhead
6.0%
JUNK
Features
Meetings
25.0%
Defects
45.0%

So, roughly...

A given *useful* feature gets maybe 6% of a programmer's time and attention...

... and not all of that 6% at the same time.



Applause from you and 192 others

John Cutler
Multiple hat-wearer. Product development nut. I love wrangling complex problems and answering the why with qual/quant data. @johncutlefish on Twitter.
Aug 18 · 2 min read

Feels Like Faster vs. Is Actually Faster

I put together this comparison table a couple days ago and shared it on Twitter. Apparently it struck a chord.

To be clear, "fast" is not the end goal. The end goal is to sustainably generate beneficial outcomes. Crap shipped fast is still crap. That said, *impact* velocity matters—it buys you options, and lets you move later and pivot more quickly.

This feels like going faster	But this actually makes us faster
Starting	Finishing
Less slack	More slack
Parallelizing work	Serializing work
"Filling up" timeboxes	Clear, overarching timebox goals
Higher work in progress	Lower work in progress (to a point)
"Getting ahead" of the work	Starting together
Specialization	T-shaped (to a point)
Shipping and jumping to next project	Leaving time to respond to feedback
Cutting corners (we'll fix it later)	Qualify focus (fix it now)
Refactor as special effort	Regular refactoring
Handing off to test. Starting new work	Pairing with test. Finishing together
Hiring more people	Tooling, infrastructure, environments, quality
Throwing new team members into fray	Careful and safe onboarding
Individual assignments	Team goals, pairing, swarming, mobbing
Chase efficiency	Encourage messy but effective collaboration

The Primal Scenario

Systems in general work poorly or not at all.

Complicated systems seldom exceed 5% efficiency.

J. Gall

The Systems Bible - 1975

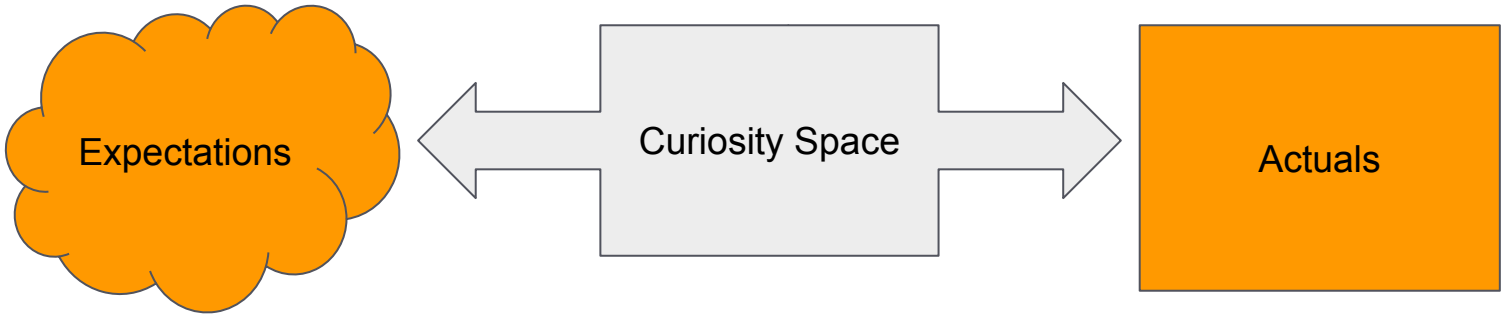
$$B = f(P, E)$$

Dr. Kurt Lewin

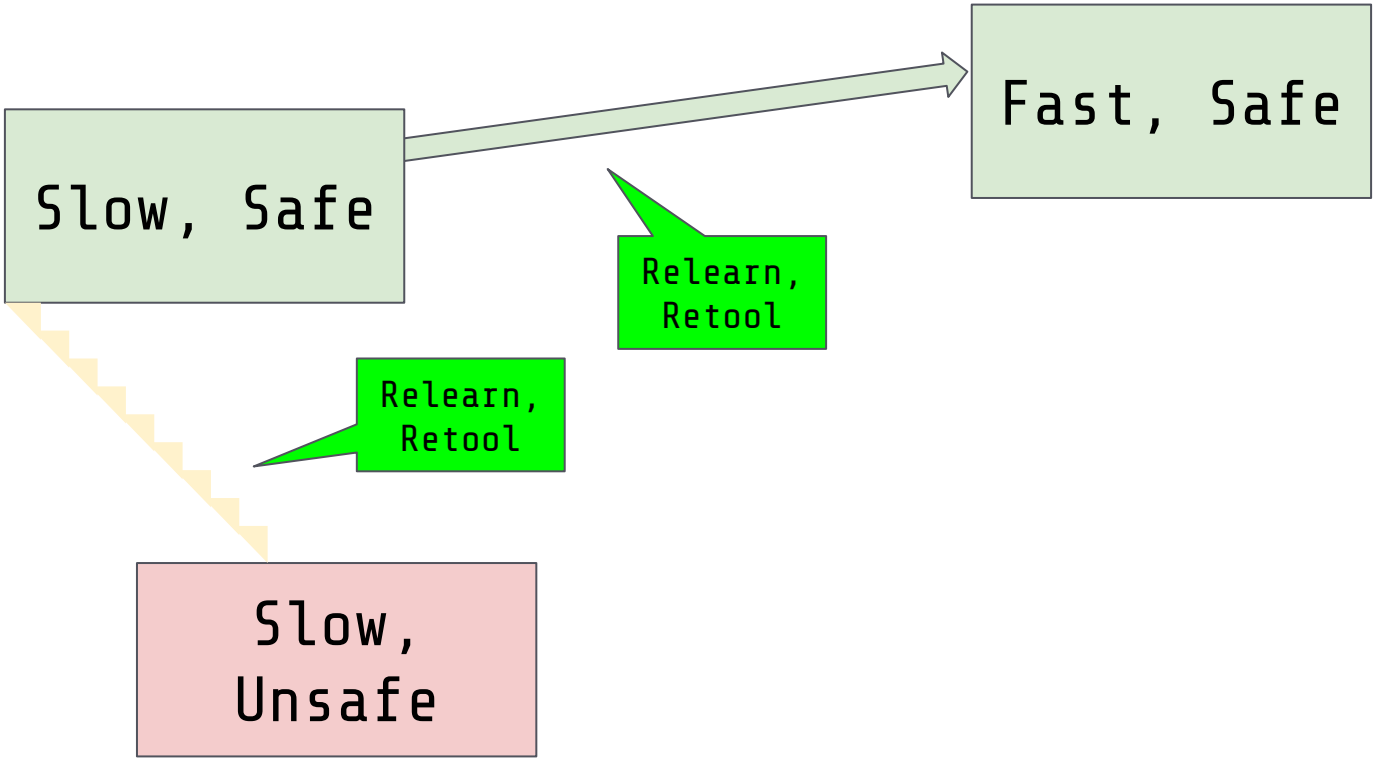
Principles of Topological Psychology (1936)

Your productivity is unreasonably HIGH*

*given the circumstances


Expectations
Curiosity Space
Actuals

Benjamin Zander: “How Fascinating!”


Slow, Safe
Fast, Safe
Relearn, Retool
Relearn, Retool
Slow, Unsafe

… and we've not even touched on ***building the right thing*** (customer development).

See:

Andy Van Fleet

Useful, Usable and Desirable

UX is not about making things pretty. It's about designing products that work. Products that work are typically useful, usable and desirable. In this talk, we'll focus on the UX process and how to

Learning isn't a task we do *instead* of working.

11/12ths of our work *is* learning and thinking.

The other 1/12th is just typing.

Practices

Kaizen

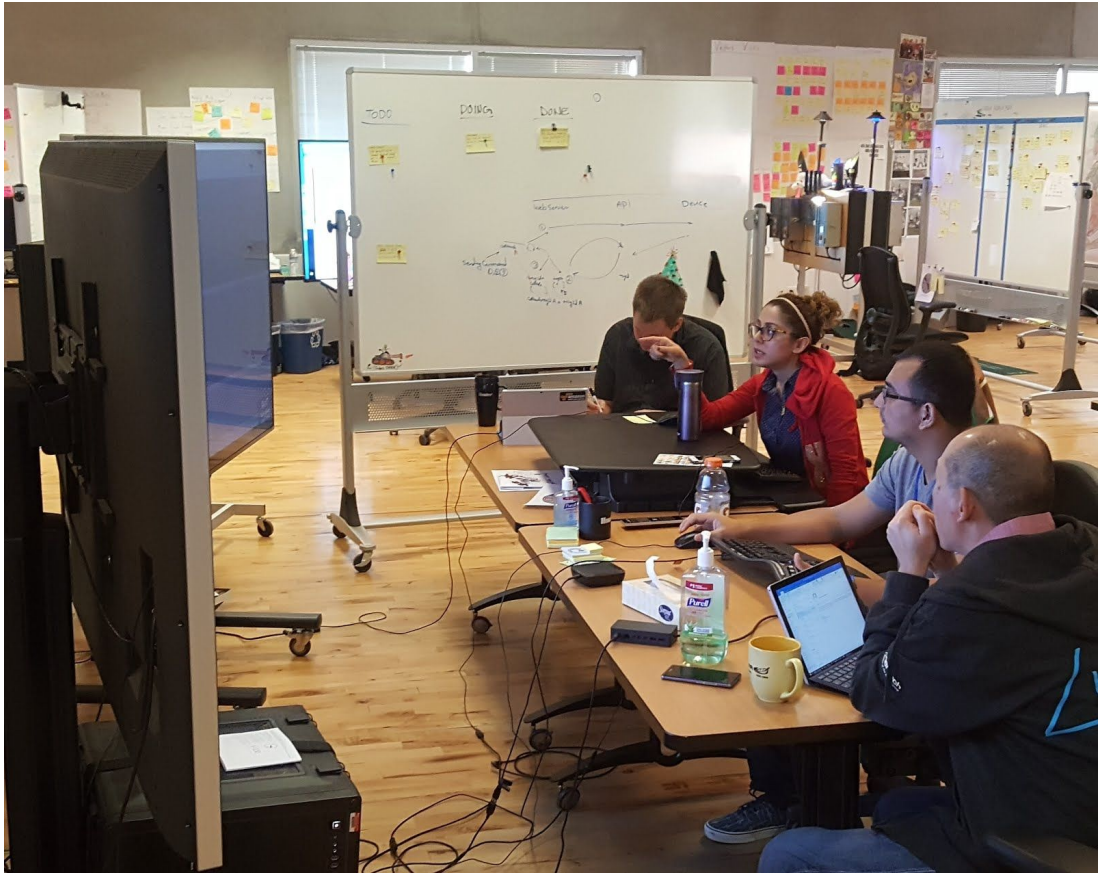
- Daily Kaizen (2-second lean)
- Kaizen Events (see Fastcap videos)

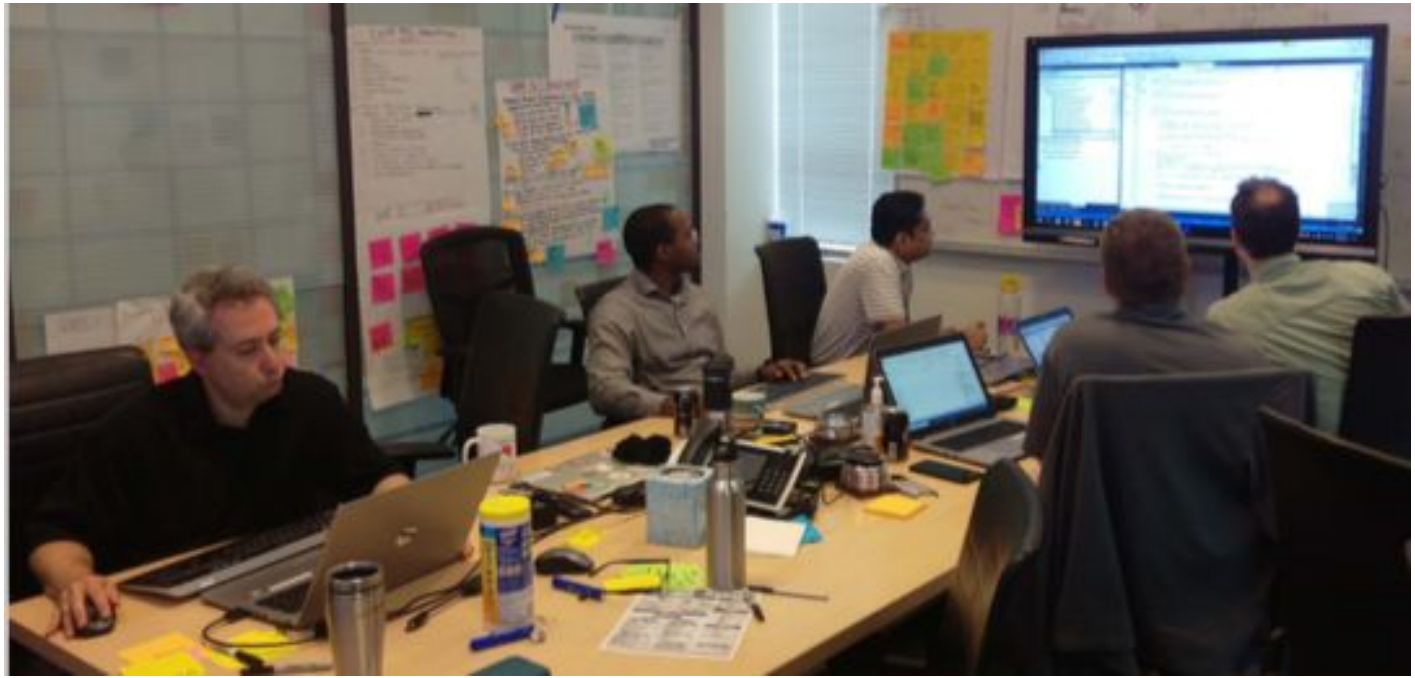
Paul Akers

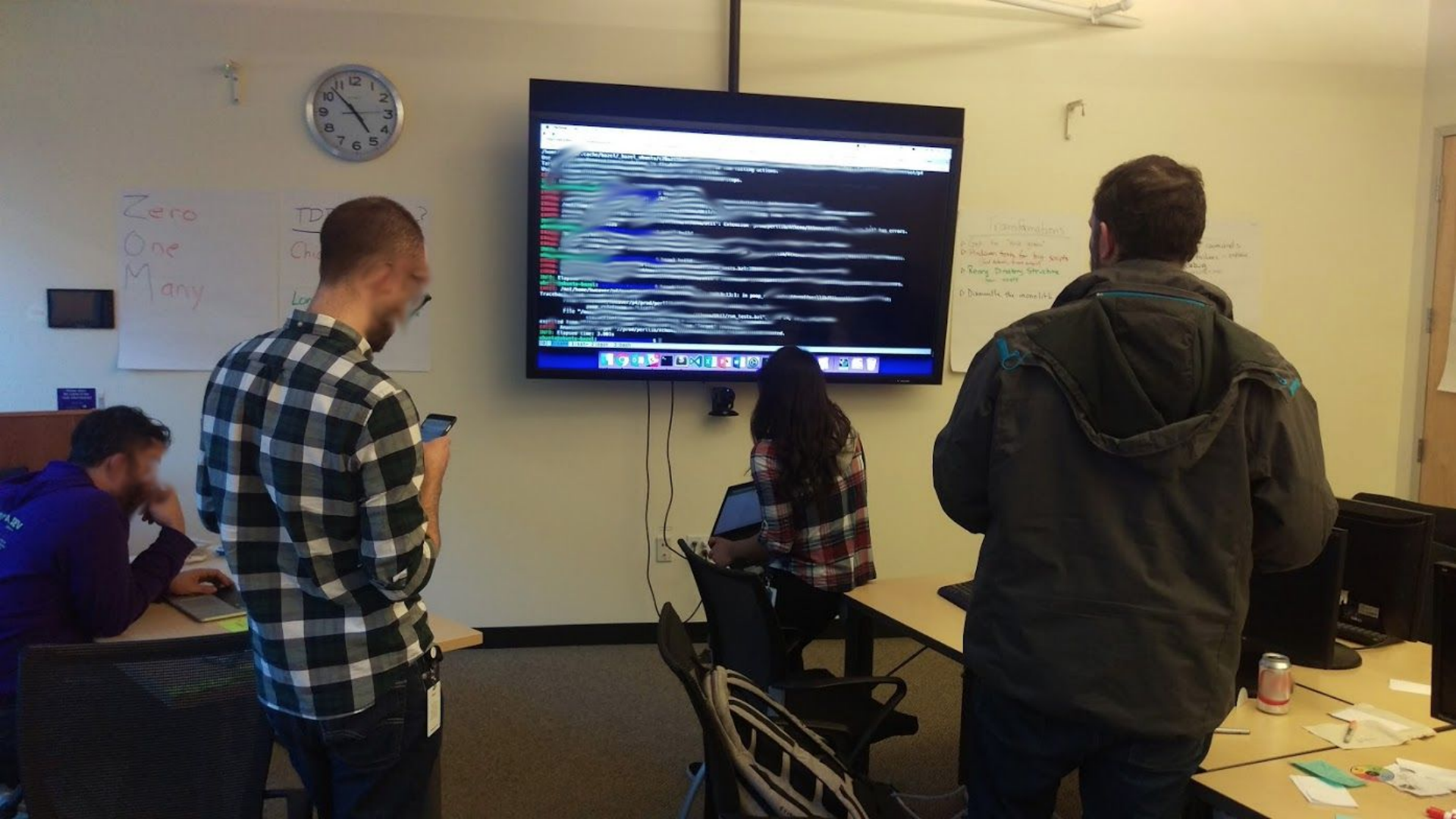
2 Second Lean
http://paulakers.net/books/2-second-lean

Teaming

- Pair programming
- Mob programming
- Swarming


TODO
DOING
DONE
Purell




Zero
One
Many

Waste Snake

Waste Snake


Wait for Build
10 min
Irrelevant xyz
Meeting
6 * 1 hr
Wait for Build
15 min
Wait for Build
10 min
Flaky Test
10 min
Product Owner
Unavailable
1 hr
Wait for Build
35 min
Wait for Mgr
Decision
1 d
Side Project
2 d
Wait for Build
20 min
Wait for Buil
10 min
Windows Updat
6 * 20 min
Surprise Worl
1 hr
Production Issu
4 hr
Wait for Build
20 min
Lousy x Code
3 hr
Copyright 2017, by Industrial Logic, Inc. All rights reserv


WASTE Snake
DB has invalid SP and invalid FK
- Dev + DBA could clean this. Safety must be established
No stable, known db for testing
MS VC 2017 MSTEST CAN IGNORE IGNORES (until Reboot)
Didn't Know How to layout subdivision dropdown (4 Designer out)
Long Builds
VS can't Build Clarify: a Frank's machine
Did Not Verify Clean Environment Before Development
CAN'T Run STEP-by-STEP Dbg on SQL
Shelving with all the SP mechanism.
Cannot run Back office test locally
Long Lock Times on maintine website
Markup DB directly
Switching Machines hurts us.
Inconsistent design on fetching data some screen
Last Access To Schedule Box(DB)
Git ignore hard-build slow cycle time
No commit at end of day. Have to remember changes
Missing Project Reference
Sauron unit-test Setup
Learning that test class must
"LOST"

- Awareness of inhibitors
- Fodder for kaizen events
- Opportunity for daily kaizen
- Raise issues to management

Promise Debt

List of things we have deferred, which keep work from being **done done.**

- So you don't forget
- Manage cognitive load
- Make remaining work visible

“Experiment” Budget

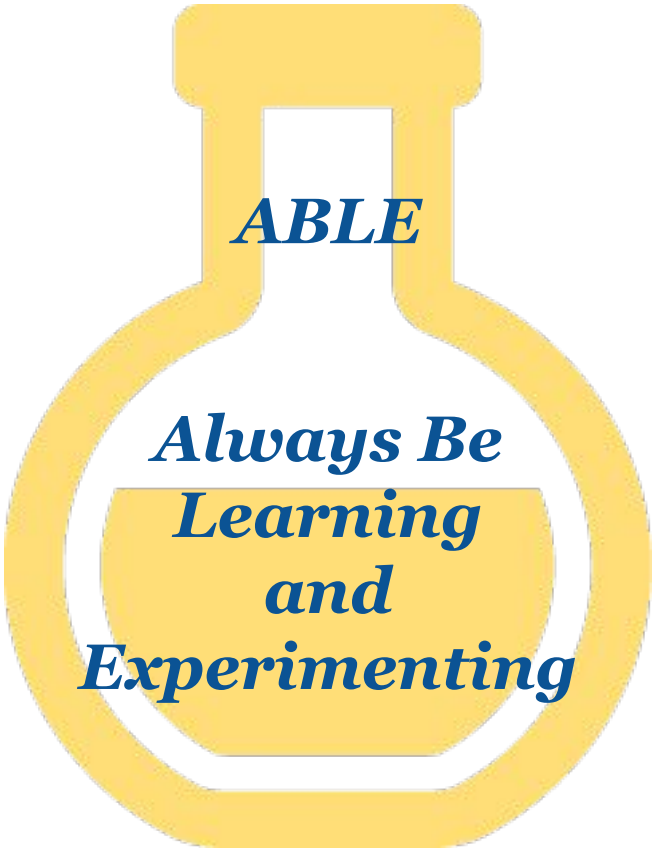
- Try things without committing
- Learn to evaluate merit of approaches
- Constantly seek “better”
- Call kaizen events
- Move improvement to “RW” side of line

Some results:

- Test reporting tool automation
- Local database for engineering builds
- New testing libraries
- Faster build
- IDE/editor auditions

TIL

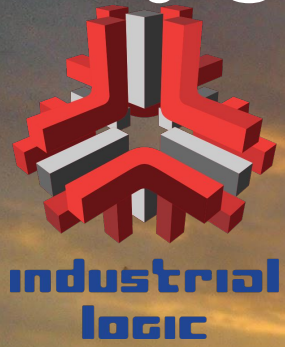
- Today I Learned
- Media doesn't matter
- Must refresh regularly
- Celebrate wins
- Share techniques
- Don't have to be big wins

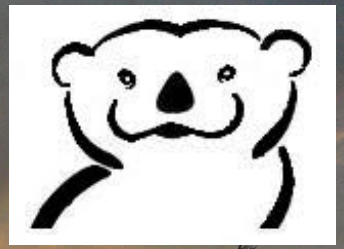


- Kaizen
- Teaming
- Waste Snake
- Promise Debt
- TIL

… and what else?

tottinge@IndustrialLogic.com
http://agileotter.blogspot.com





@tottinge