THINGS TODO

- [ ] [JPB] Cosmo deck: Talk about work SL has done over the years, have a forward-facing thing at the end
- [ ] GORD announcement
- todo: should each TQ get an emoji?

- berachain, sei: they're building something a little bespoke, pushing everything to te edges
  - ABC++
  - server v2
  - timelines are SO SLOW
  - cycle time is a killer
  - requirements aren't ACTUALLY what the user needs
  - [ ] PUT THE JANICE SAWTOOTH IN HERE EARLY
  - all the forks
  - too much forking => fracture. ppl don't contribute back upstream
  - lots of solutions are band aids
  - NEED A 'WHAT IS A TOOL?' slide. Cooper hewett hand axe => iphone
  - as a community, how/when do we pay down the debt?
  - DEBT != "shitty code". Debt == time-shifting when the bill comes due

Structure

I. We're in danger: Appchain thesis could die

II. There is a way: cut the fat

III. here's how it works

IV. Summary

I. We're in danger

I. We're in danger: Appchain thesis could die

What does success look like?

- ETH / rollup thesis has peaked
- significant incumbent investment
- clear path to > TVL than rollup thesis

What does failure look like?

- ETH / SOL / L2s press their TVL / penetration gains
- catch up w/ our POS-in-production lead
- crypto becomes another VHS-vs-Beta or Qwerty-vs-Dvorak

Why are we at risk?

- unreliable delivery
- unsustainable builds
- poor lotto count

About me

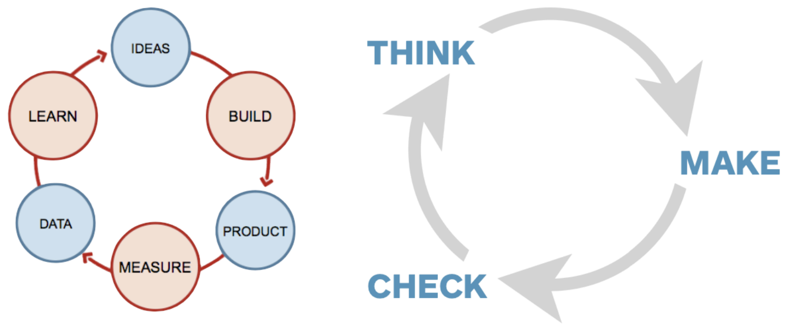
- w2 design => eng & product
- blockchain found me
- methadone clinic vs midnite bball

II. There is a way

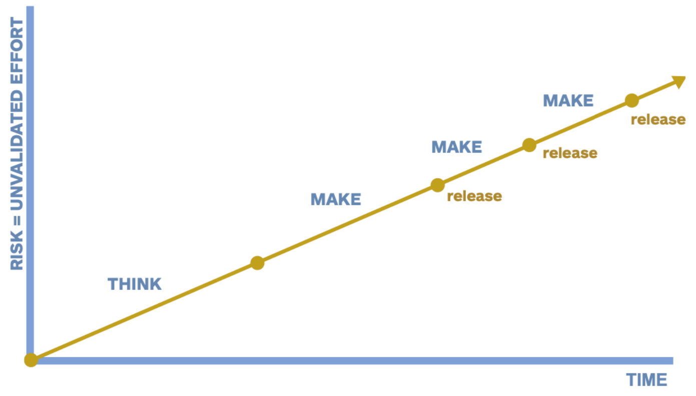
POSSUM

https://www.simplermachines.com/mr-reciprocity/

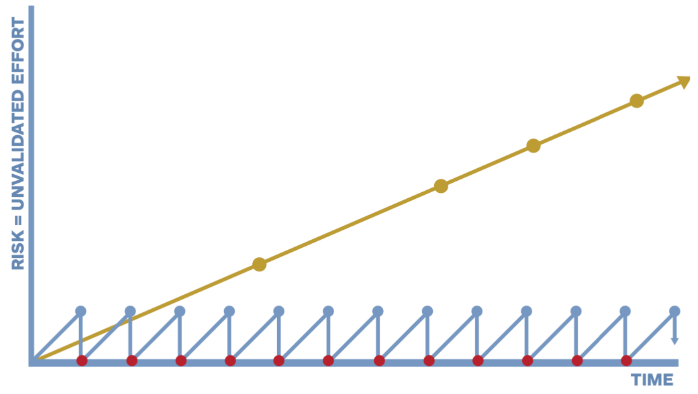
Everyone tends to get stuck



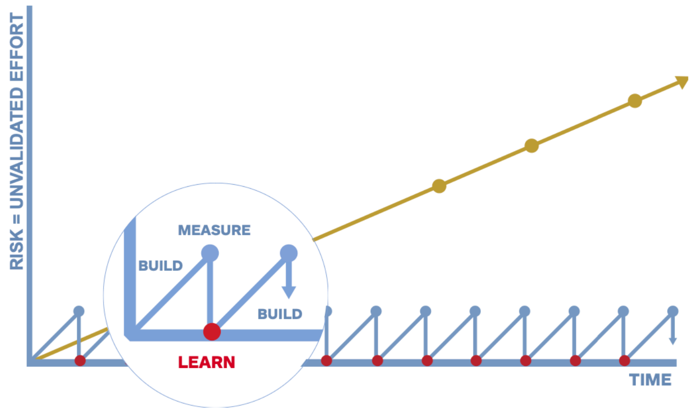
Plot the difference



Lots of little wiggles



Each wiggle is a learning cycle



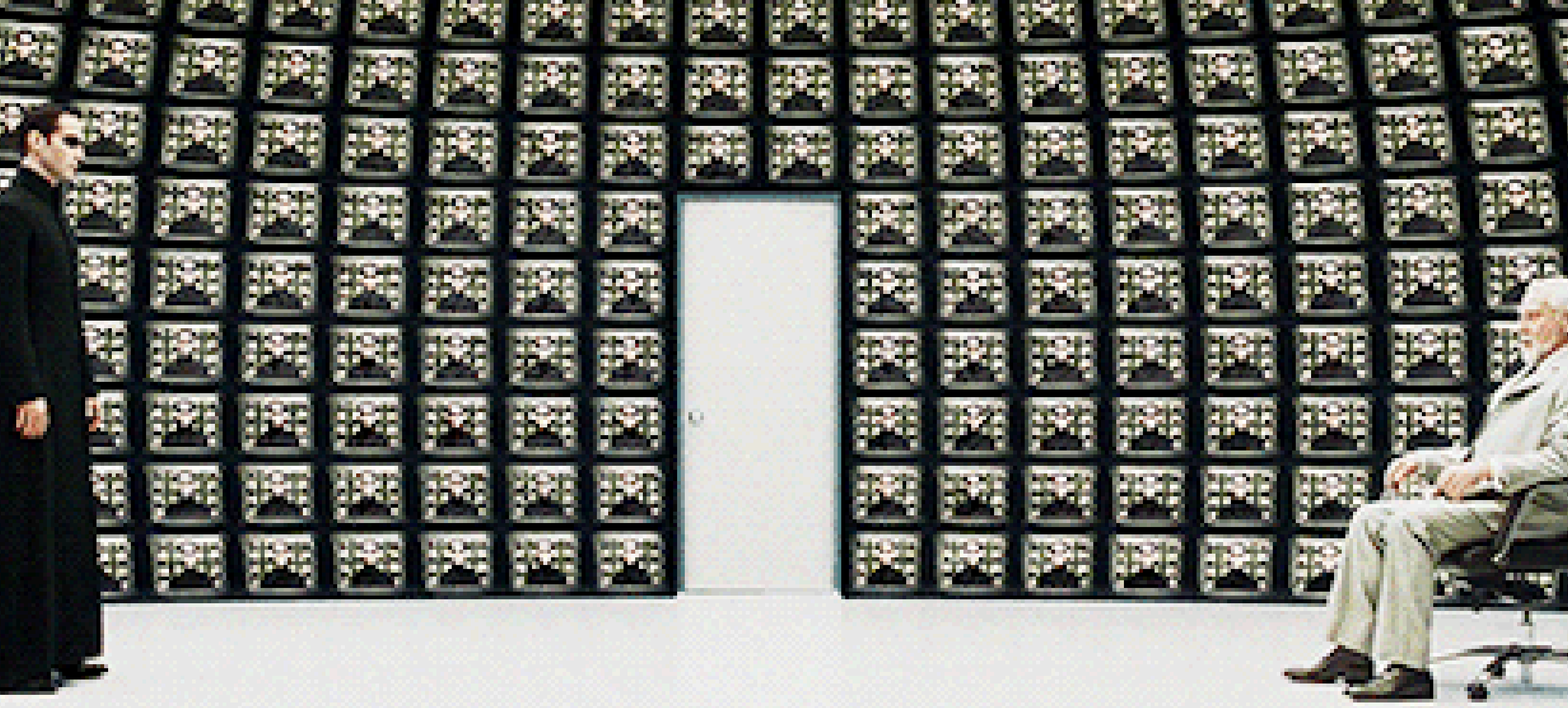
Hypothesis THIS IS THE WAY:

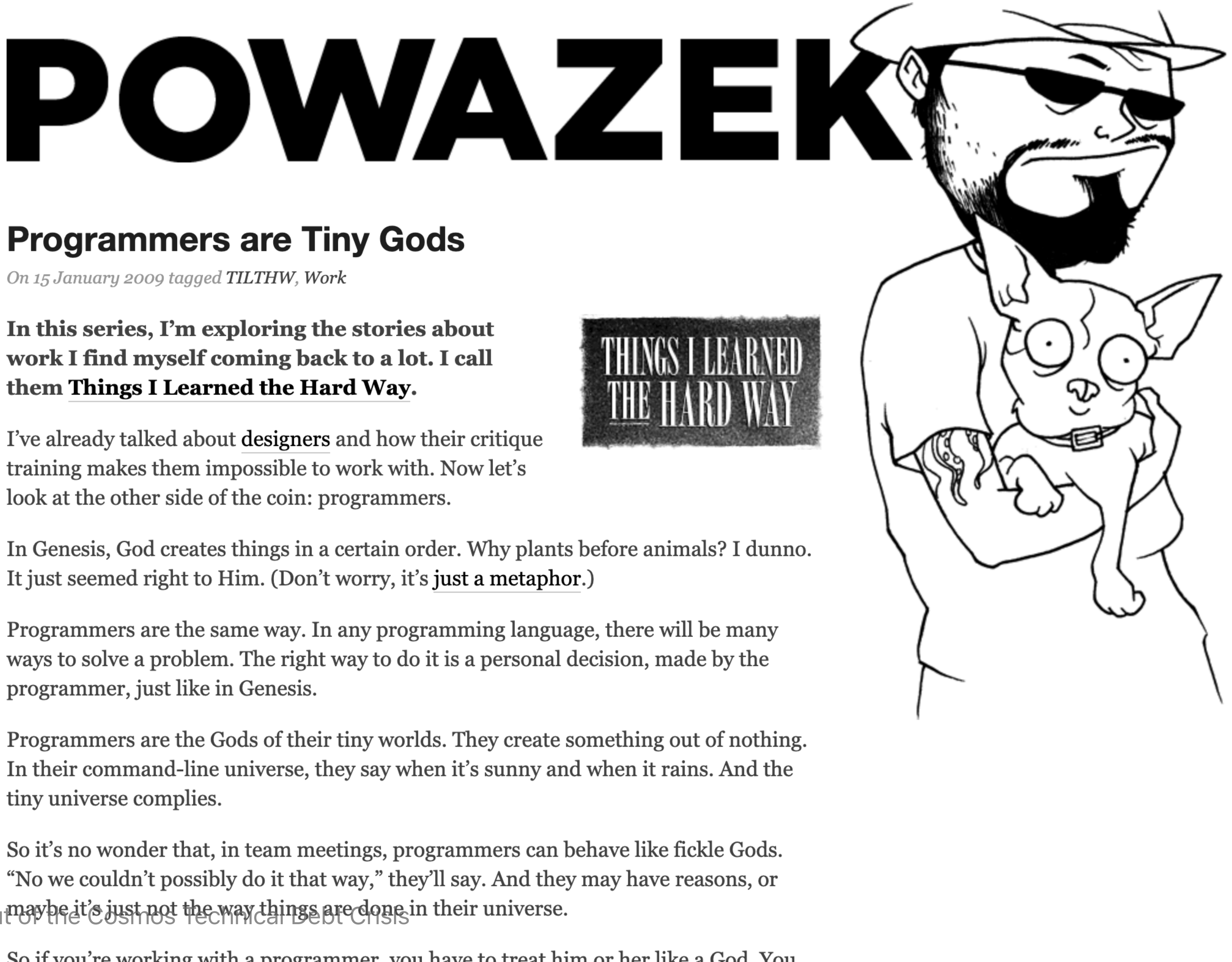
- I (we?) believe **Web 3 engineering culture**...
- **Burns tons of energy re-inventing wheels**, and chronically suffers from problems which have known solutions.
- We can fix it by **revisiting those Wheels and Solutions**, and
- We'll know we're right if we see **positive (qualitative) feedback, improved (quantitative) cycle times, and healthier code (as measured by static analysis, defect rates, etc. etc.)**

Corollary:

"Agile" is an old word for "Decentralization"

0. People Who Write Software Are ~~Uniquely~~ ~~Distinctly~~ Peculiarly Bad At Remembering Their Own History





POWAZEK

Programmers are Tiny Gods

On 15 January 2009 tagged TILTHW, Work

In this series, I'm exploring the stories about work I find myself coming back to a lot. I call them Things I Learned the Hard Way.

THINGS I LEARNED THE HARD WAY

I've already talked about designers and how their critique training makes them impossible to work with. Now let's look at the other side of the coin: programmers.

In Genesis, God creates things in a certain order. Why plants before animals? I dunno. It just seemed right to Him. (Don't worry, it's just a metaphor.)

Programmers are the same way. In any programming language, there will be many ways to solve a problem. The right way to do it is a personal decision, made by the programmer, just like in Genesis.

Programmers are the Gods of their tiny worlds. They create something out of nothing. In their command-line universe, they say when it's sunny and when it rains. And the tiny universe complies.

So it's no wonder that, in team meetings, programmers can behave like fickle Gods. "No we couldn't possibly do it that way," they'll say. And they may have reasons, or maybe it's just not the way things are done in their universe.

So if you're working with a programmer, you have to treat him or her like a God. You

1. Why Agile? (Atoms vs. Bits)

Waterfall

is what happens when you
optimize for predictability.

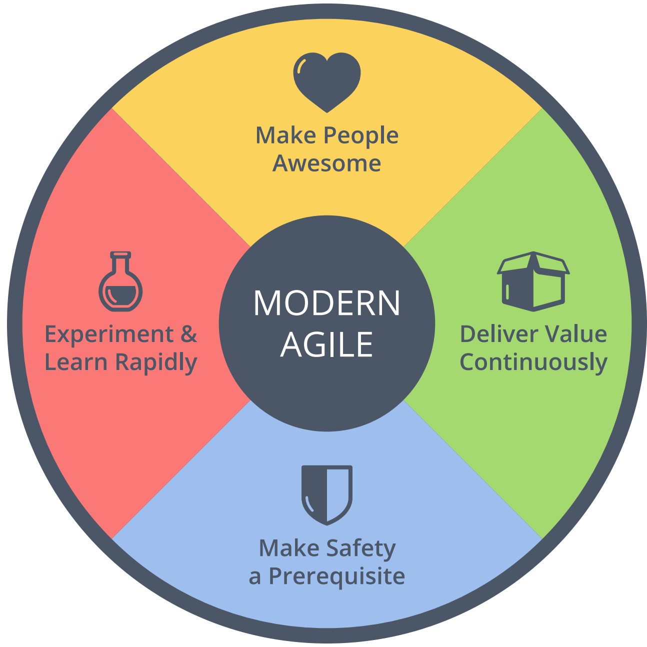
Agile

is what happens when you
optimize for low cost-of-change.

The "Standing on one foot" explanation of agile:

- *"Optimize for*
  Short Feedback Loops"
- (...all the rest is commentary...)

Original Manifesto (2001)		Modern Agile (2016)
Individuals and interactions over processes and tools	➡	🟨 Make People Awesome
Working software over comprehensive documentation	➡	🟩 Deliver Value Continuously
Customer collaboration over contract negotiation	➡	🟦 Make Safety a Prerequisite
Responding to change over following a plan	➡	🟥 Experiment & Learn Rapidly


Make People Awesome
Experiment & Learn Rapidly
MODERN AGILE
Deliver Value Continuously
Make Safety a Prerequisite

Sustainable Pace:

- "A pace we can persist, indefinitely."

III. Promises

IDEA: After I go through these, use one of those phone vote apps to pick a single TQ to go deep on

PROMISE: a shared, falsifiable understanding of "MVP", aka a *Definition of 'Done'* at an ~Epic level of resolution.

[Lean Hypotheses][]

PROMISE: Cheap, simple confidence in shared expectations

The "[Well-formed stories][]" format

1. Describe **Acceptance Criteria in Gherkin** ("Given", "When", "Then", "And")
2. Describe **motivation** using "As a" / "I want to" / "Because"
3. Story titles ought to include the word **"should"**

PROMISE: Minimal, necessary, sufficient team comms

FUTURE, PRESENT, PAST, TIME

Every team should have a Canonical Source Of Truth for:

1. ...where we keep our **FUTURE PLANS**
   (Usually a backlog. GH Projects? Jira? A legal contract?)
2. ...where (and how) to have **CONVERSATIONS IN THE PRESENT**
   (Slack? TG? Respond to emails that same day?)
3. ...where to keep **WORK THAT'S BEEN DONE IN THE PAST**
   (Code repos? gDocs? Published to a PHEELblog?), and finally
4. ...what's our regular rhythm of meetings?

PROMISE: confine planning overhead to 7.5% of yr week. The other 37 hrs are for coding.

A Responsible Recipe for Fewest Possible Meetings

- 50m weekly: Daily Standup (5x weekly, 10m each)
- 60m weekly: Iteration Planning Meeting ("IPM), 1x weekly, 60m each
- 60m weekly: Friday Afternoon Mtg (rotate bw Team Retro, Tech Retro, and Release Planning)

That's it. ~3hrs/wk for builders to keep teams and comms healthy.

PROMISE: A shared, actionable, falsifiable rubric for "this mtg could have been an email FIND MEME"

💀 MEMENTO MORI MEETINGS (PRESENT)

Life is too short to waste in meetings.

Meeting	Outcome	example
DELIBERATION	decisions	whiteboard a feature, prioritize a backlog
BONDING	relationships / bonds	karaoke
BIG NEWS	knowledge is disseminated	"We've exited!" or "${PERSON} is leaving".

Anything else should be an {email, information radiator, etc.}

PROMISE: minimal, necesarry, sufficient

How Should We Think About Backlog / Story / Kanban Flow?

Status	How do we know?
☃️ Icebox	**Unprioritized** but worth doing.
📋 Backlog	Prioritized **unstarted** work.
🤸 In Progress	**Active**ly being worked on.
👀 Waiting / In review	**Depends** on something we don't control.
🥳 Done	**No more effort** is required.

PROMISE: win by minimizing muda by "progress is the primary measured in working software"

Custom field: Story types

⭐ User story	User-facing value
🐞 Bug	Regression (*not* new requirements)
⚙️ Chore	No user-facing value
👑 Epic	Anything bigger than a story
🏁 Release Marker	Fixed time (scope or cost *MUST* float)

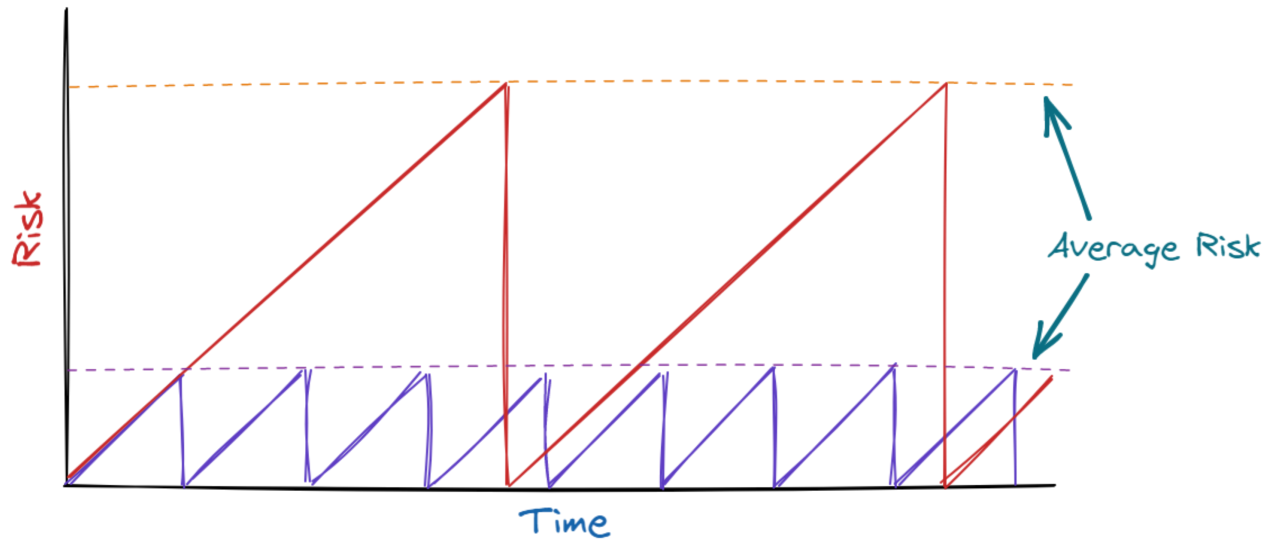
Continuous Iterative Development

Why?

- Premature Optimization is the Mind Killer
- Learning Requires Mistakes

How?

- Optimize for **Small Feedback Loops**
- resist BDUF ("Big Design Up Front")
- Fidelity Of Planning Must Be Appropriate ("Flag Down The



ERRATA

```
# 5 Forgotten Lessons of Web 2 That Can Save Web 3

## The Problem

- tactical problems

- this isn't a w3 problem (THIS HAS ALL HAPPENED BEFORE meme)
- it's a more general problem w/ lots of prior art
- Who's this for? Core Dev teams (rather than 5-person startups)

## Zoom out: what's the bigger picture
- midnight basketball vs methadone clinic

## Lesson 1. Software Craftsmanship
  - Sustainable Pace
  - Testing
  - Good Hygiene == compounding interest
    - Those bad decisions you made? You *will* see them again
    - managing technical debt
      - debt is a tool

## Lesson 2. YAGNI

## Lesson 3. Continuous Delivery => FEEDBACK LOOPS

## Lesson 4. The Right Kind of Laziness

## Lesson 5. Focus on Business Value


## DEDUCTIVE "one more thing" (or "it has not escaped our attn...") that draws them back together

## TAKE ACTION: ppl say "this sounds right. how can I do it on my team?"
```