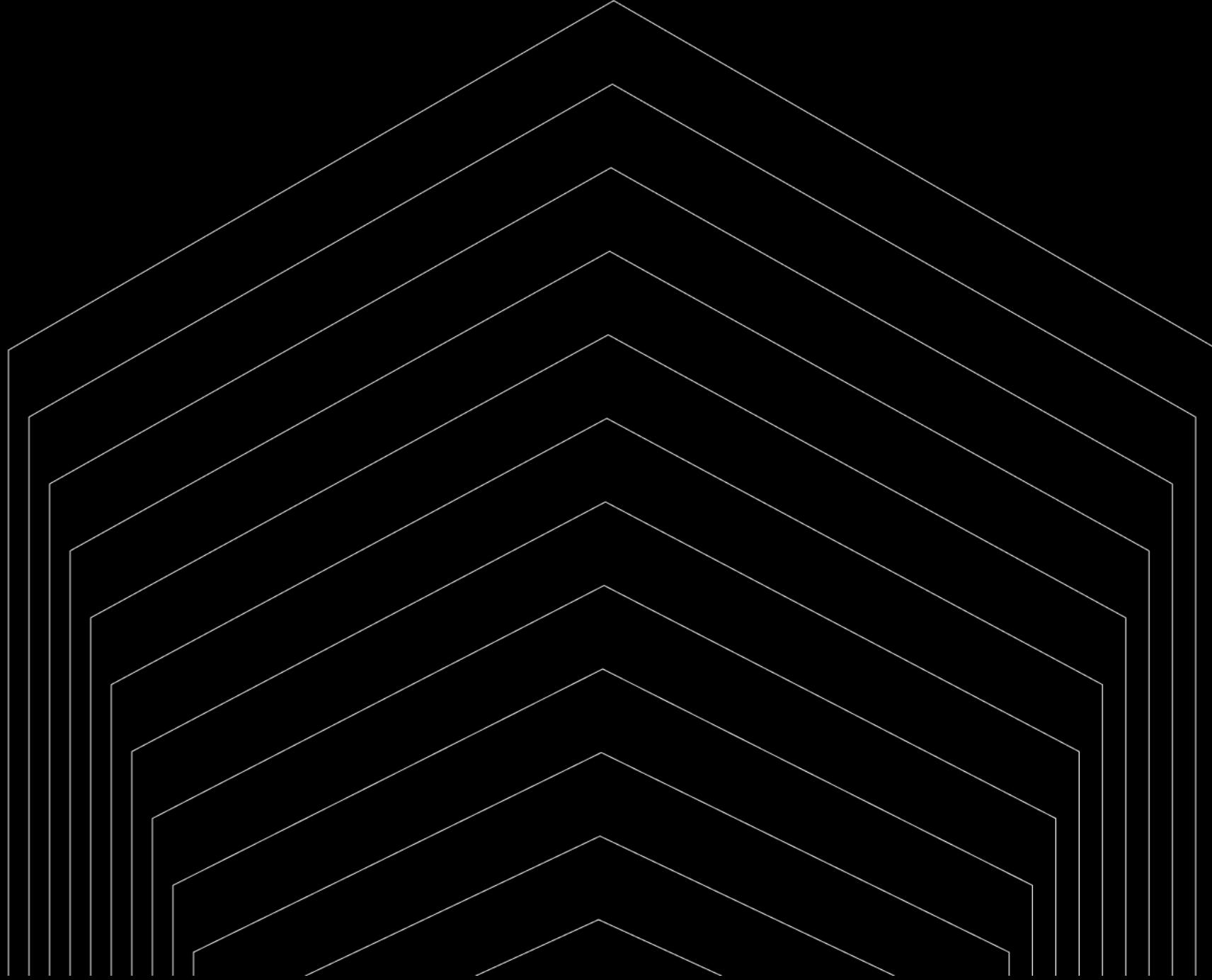


HI!



STRANGELOVE

(Tooling Our Way Out of) The Cosmos Technical Debt Crisis

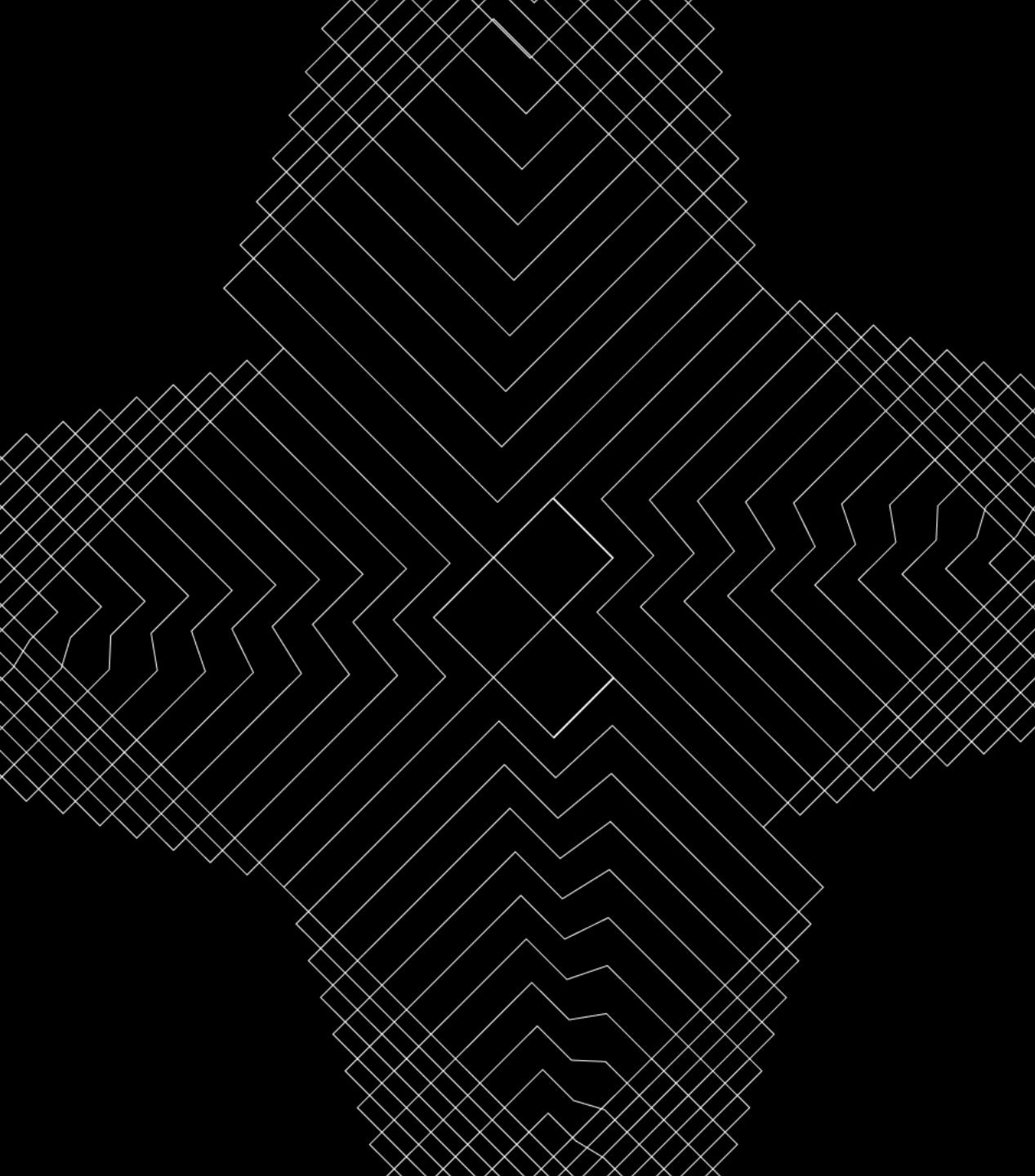
(or, How I Learned to Stop Worrying and Love teh Agile)

What do we mean by "tool"?

That which enables us to achieve more than we could unaided

"We shape our tools, and thereafter they shape us".
— McLuhan





Strangelove loves to make tools

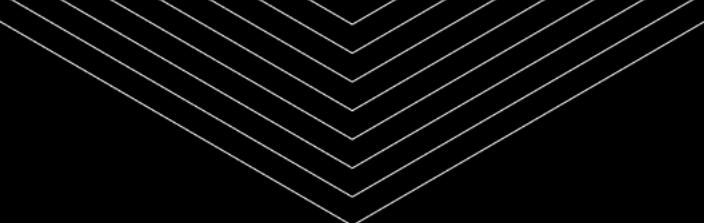
- Spawn
- Interchaintest
- Cosmos Operator
- Heighliner
- etc.

By their nature

**Decentralized Systems
Are
Robust And Resilient
(our Natural Advantage)**

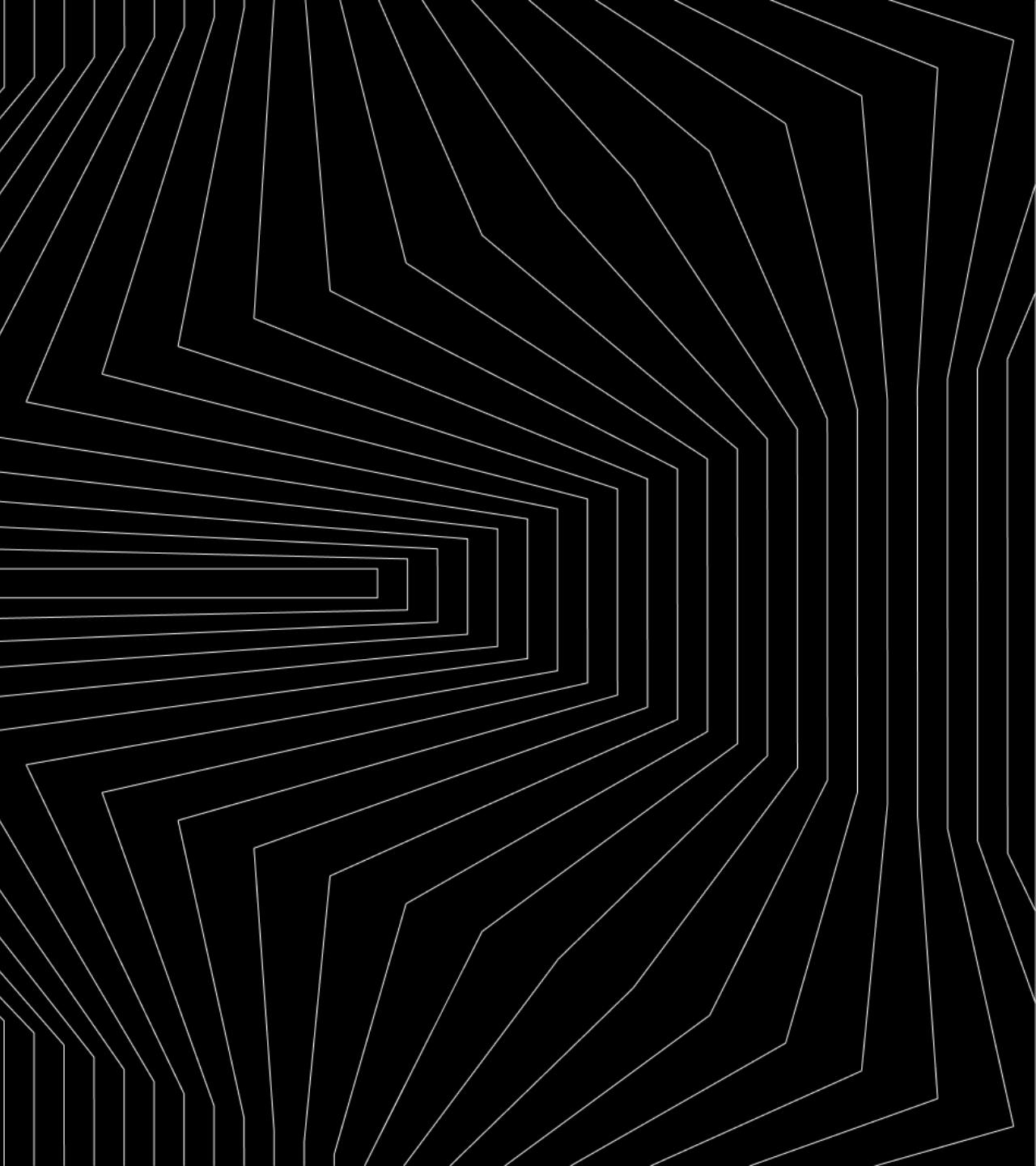
but

**The Cost is
a Heavy Coordination
Load
(our Natural Weakness)**



Do any of these sound familiar?

- 😵 "self-eating dependency graph"
- 😵 forks & cost upstream maintenance
- 😵 direct correlation bw size of project and how bespoke it's become?
- 😵 timelines are slow and unpredictable; cycle time is a killer
- 😵 requirements aren't ACTUALLY what the user needs
- 😵 (or are obsolete by the time they get to market)



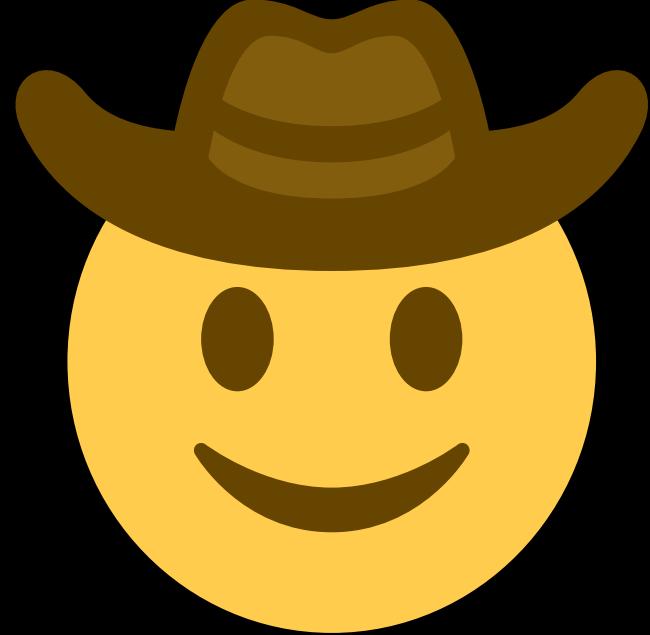
The good news

- THESE ARE SOLVED PROBLEMS
- software development has 20+ years dealing with this
- ...we just need to look past the dross of the Agile Industrial Complex and get past bad experiences

Why listen to me?

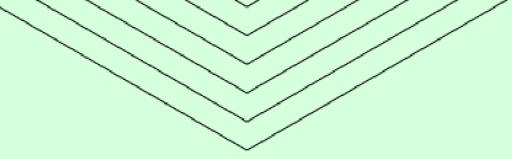
- Web 2 {design, engineering, product}
- blockchain found me
- How I got here



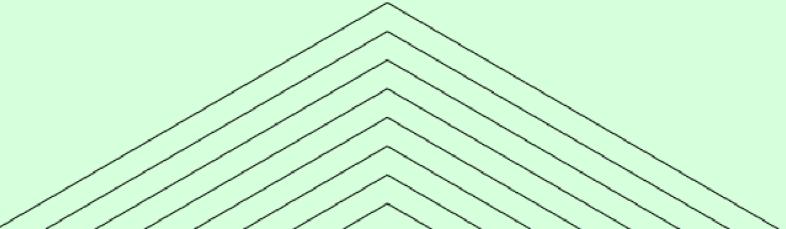


The Plan

- 0. The Backstory
- I. The Stakes
- II. The Way Out
- III. The Nitty-Gritties
- IV. The Recap



0. The Backstory



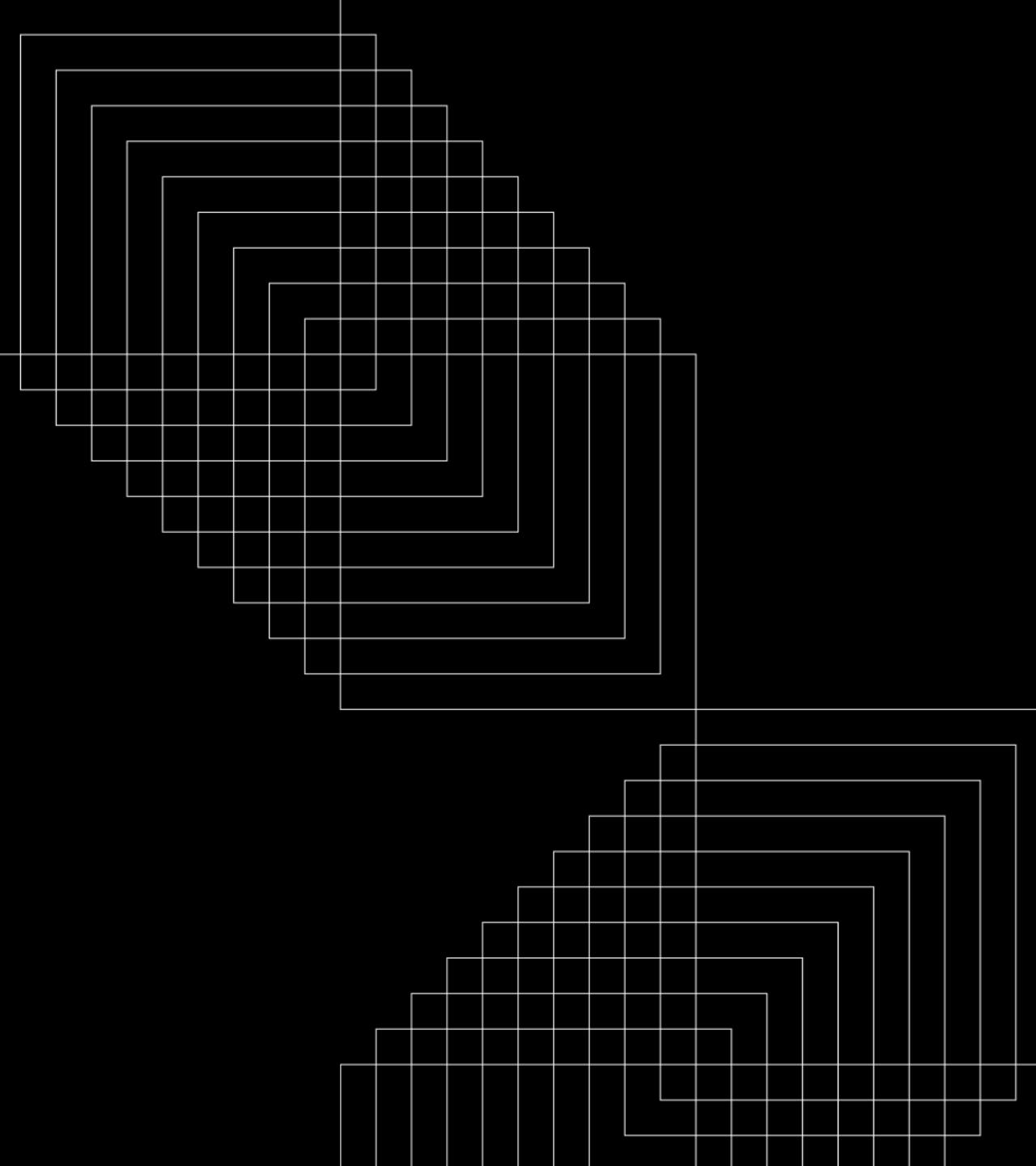
Previously, on *The 20th Century*

- WWII leaves USA as the only major intact industrial base
- 1960s: Japan is making terrible cars
- 1980s: Japan is making incredible cars
- Why? "Lean Manufacturing"
i.e., *the Toyota Production System*

Describe *Lean* while standing on one foot:

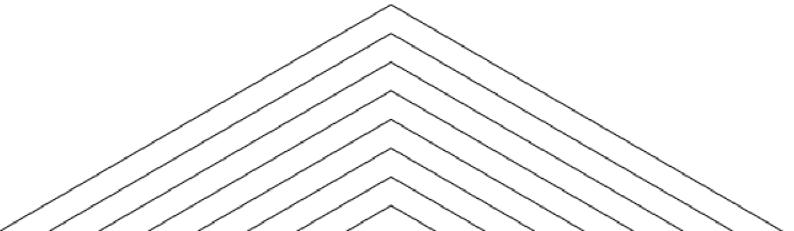
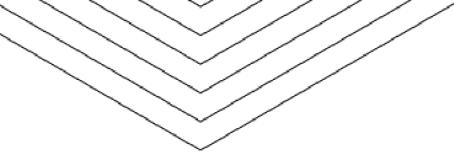
"Minimize Waste"

	Muda (無駄)	"Waste"
	Mura (斑)	"Unevenness"
	Muri (無理)	"Overburden"



Lean Manufacturing identifies 7 forms of Waste:

1. waste of *Inventory*
2. waste of *Over-Production*
3. waste of *Extra Processing*
4. waste of *Transportation*
5. waste of *Waiting*
6. waste of *Motion*
7. waste of *Defects*

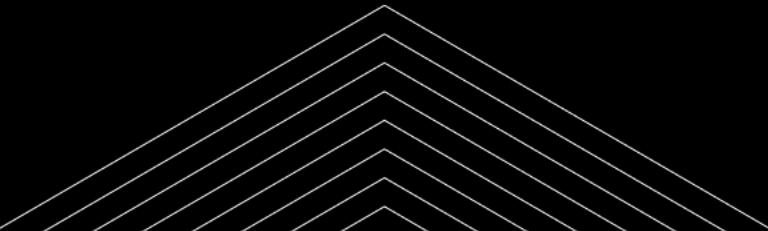


~20 years later

Lean Software Development: An Agile Toolkit

by Mary & Tom Poppendieck

Manufacturing Waste of...	Maps to Software waste as...
<i>Inventory</i>	 Partially Done Work (or Work In Process)
<i>Over Production</i>	 Delivering Extra or Unneeded Features
<i>Extra Processing</i>	 Relearning
<i>Transportation</i>	 Handoffs
<i>Waiting</i>	 Delays
<i>Motion</i>	 Context Switching
<i>Defects</i>	 Defects



Any of this sounding familiar?

I. The Stakes

**The Appchain
thesis is better**

**The Appchain
thesis is losing**

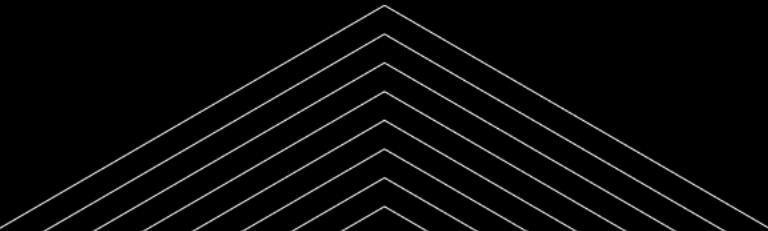




BUT! If we get this right

IT'S A COMPETITIVE ADVANTAGE

(versus the rest of Web 3)

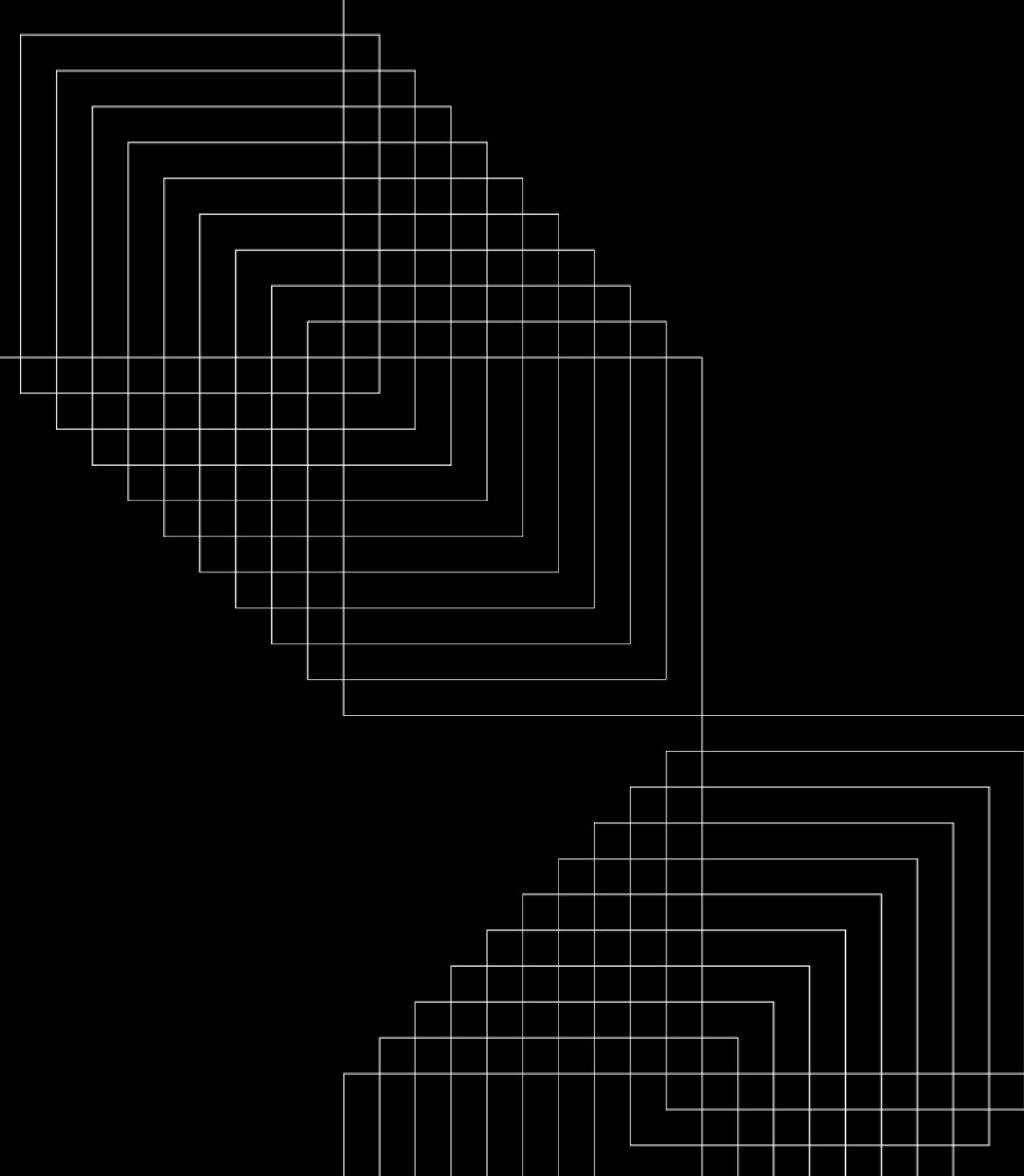


What does failure look like?

- ETH / SOL / L2s press their TVL / penetration gains
- They catch up w/ our PoS-in-production lead
- crypto becomes another Inferior Tech Wins story,
(e.g. *VHS-vs-Beta* or *Qwerty-vs-Dvorak*)

What does success look like?

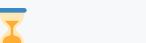
- ETH / rollup thesis has peaked
- significant incumbent investment in Appchains
- clear path to appchain-TVL-surpasses-rollup-TVL



Why are we at risk?

- Unreliable Delivery
- Unsustainable Builds
- Poor Lotto Count
- High Cost Of Change

Do Cosmos Pains map to Lean waste?

恼怒的脸 "self-eating dependency graph"	
恼怒的脸 forks & cost upstream maintenance	
恼怒的脸 direct correlation bw size of project and how bespoke it's become?	
恼怒的脸 timelines are slow and unpredictable; cycle time is a killer	
恼怒的脸 requirements aren't ACTUALLY what the user needs	
恼怒的脸 (or are obsolete by the time they get to market)	



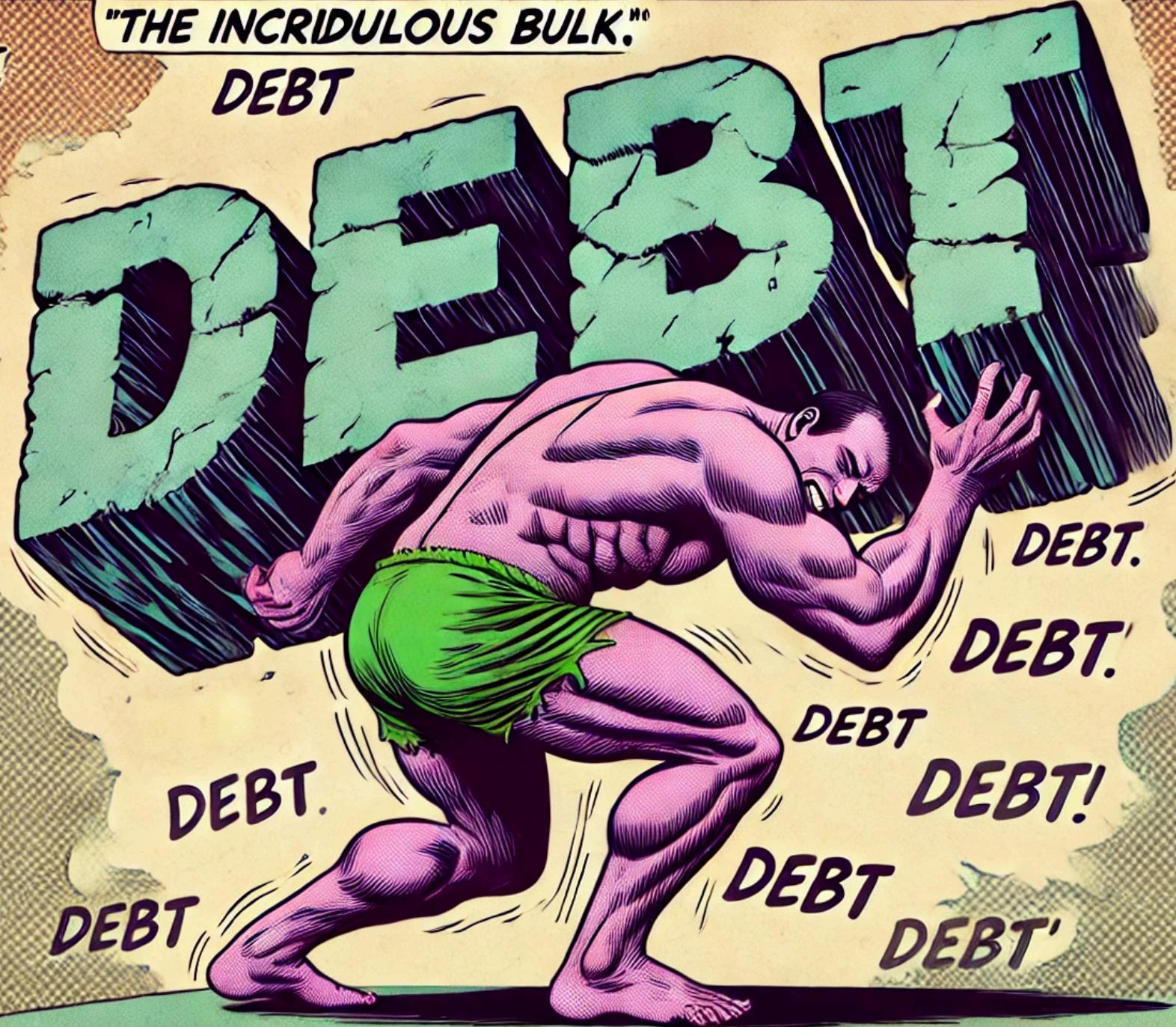
What's missing from this description of "waste"?

"THE INCREDULOUS BULK."

DEBT.

DEBT

DEBT!



DEBT.

DEBT

DEBT.

DEBT'

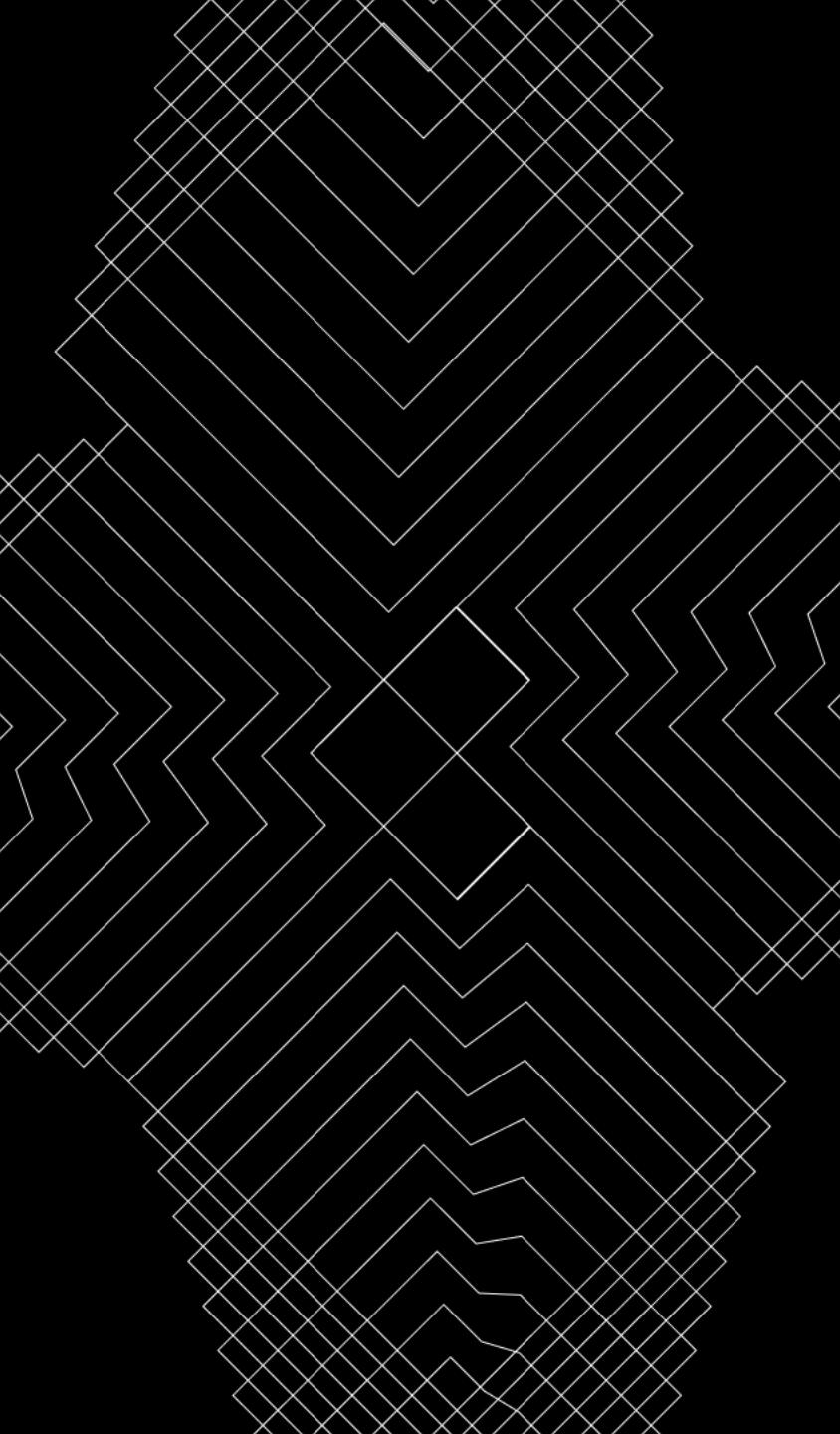
DEBT

DEBT

DEBT'

DEBT

DEBT



 **Waste != "Debt"**
(though we often conflate the two)

Tech Debt != 🧑‍💩 Bad Code 🧑‍💩

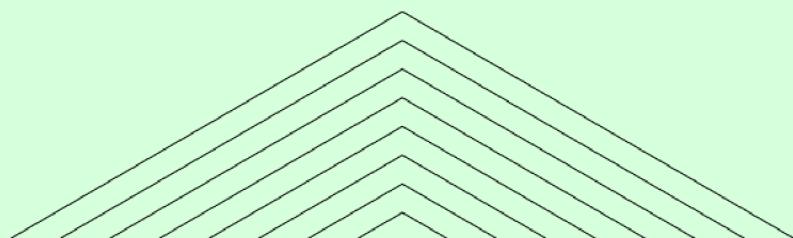
**Tech Debt == the freedom to choose
when to spend work effort optimizing
code**

In software
development,
 *Premature*
Optimization 
is the mind killer





HMW escape this {debt, waste} trap?



II. The Way Out

**What if we stopped
forgetting what worked
well last time?**

 $f(x)$

 $f^{-1}(x)$ 

What might translate from Web 2..3?

	Lean software Muda	Web 2 (XP) remedy	Works for Web 3?
	<i>Partially Done Work (WIP)</i>	WIP Limits	
	<i>Delivering Unneeded Features</i>	Validating Assumptions	
	<i>Relearning</i>	strong Lotto Count, pair-programming, CoC	
	<i>Handoffs</i>	autonomuous, x-fct'l teams	Is w3 too specialized for x-fct'l?
	<i>Delays</i>	reduce deps, bring related work together	Is decen inimical to "bring related work together"?
	<i>Context Switching</i>	WIP Limits	
	<i>Defects</i>	BDD, TDD	



Atoms vs. Bits

The Old Way

optimize for predictable timelines.

The Right Way

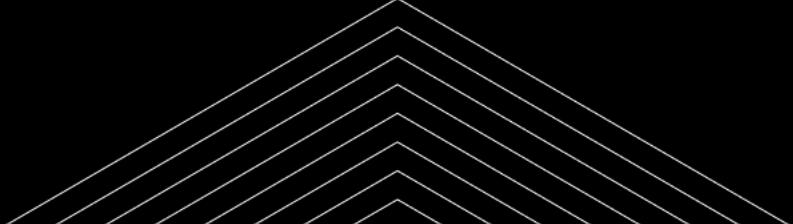
optimize for low cost-of-change.

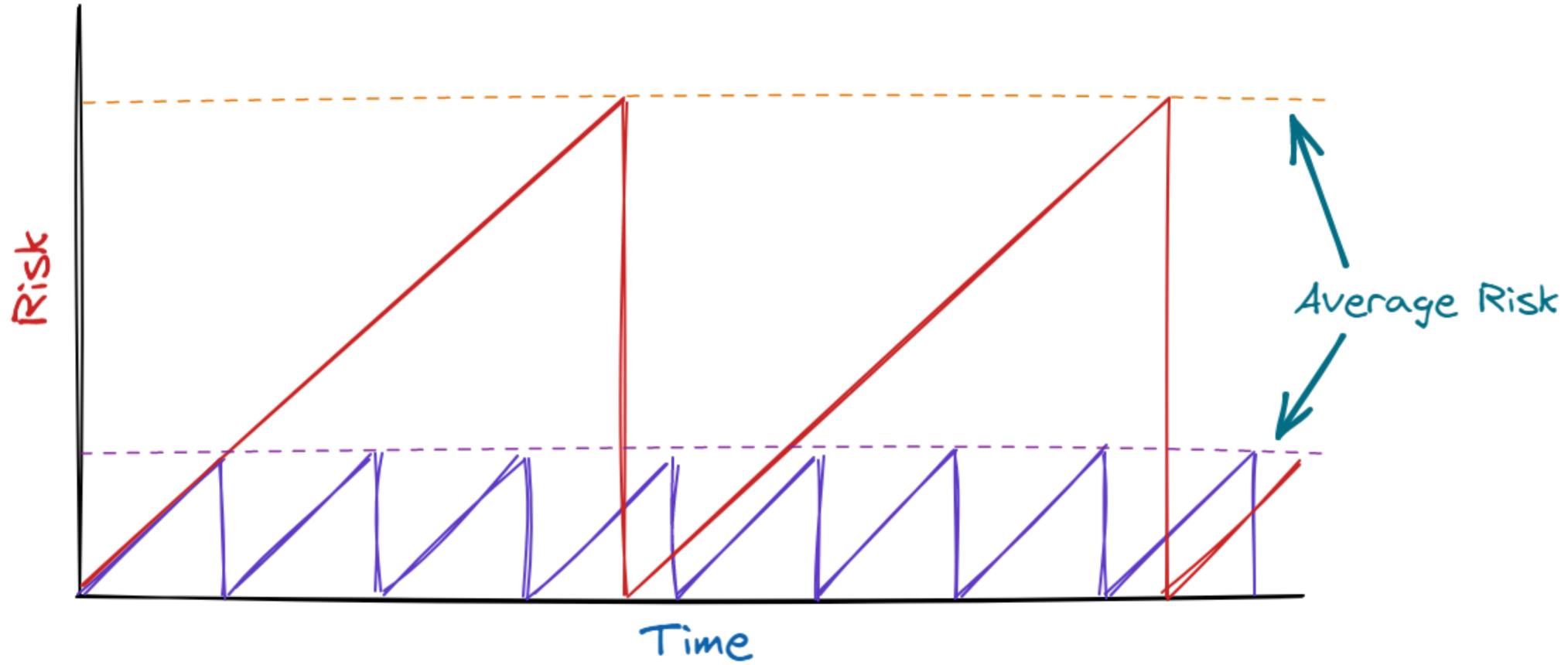


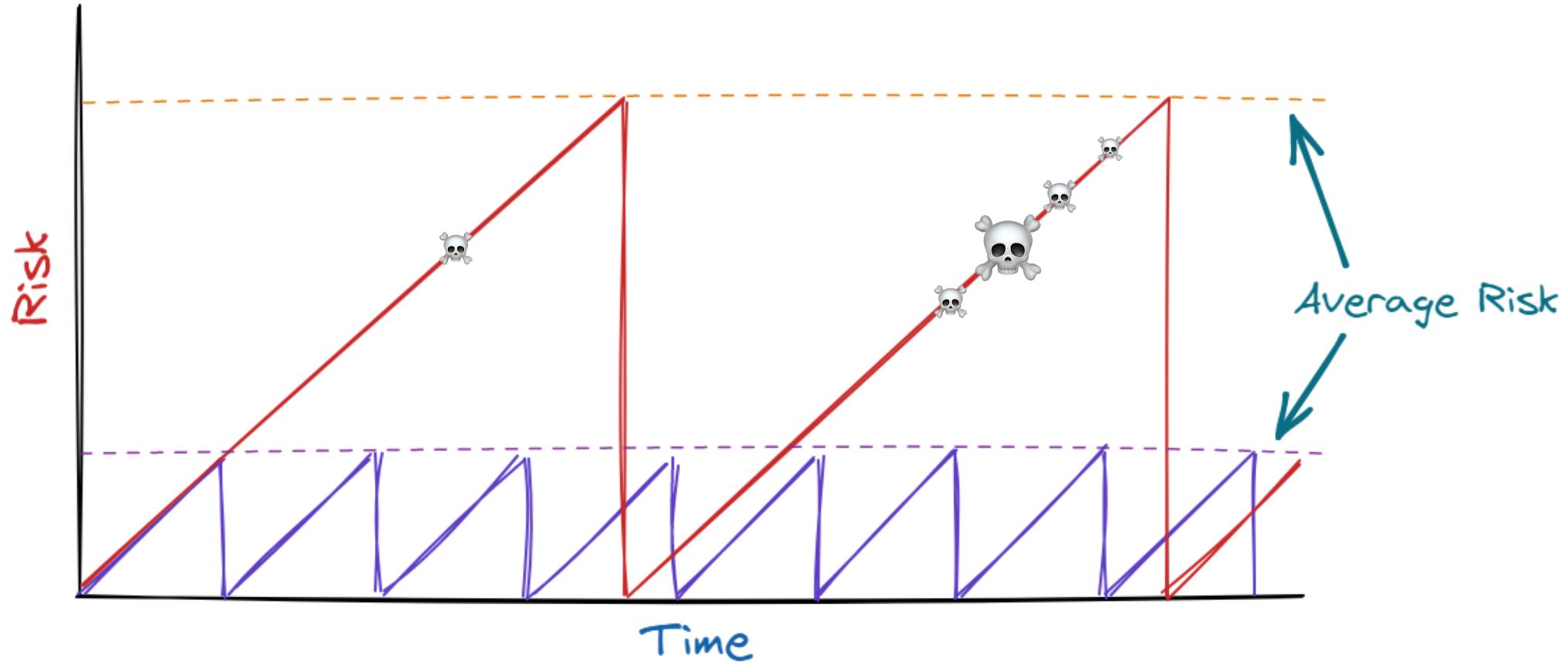
The "Standing on one foot" version:

Optimize for Short Feedback Loops

(...all the rest is commentary...)









Fear leads to Anger

Anger leads to Hate

Hate leads to Suffering

Continuous Iterative Development

Why?

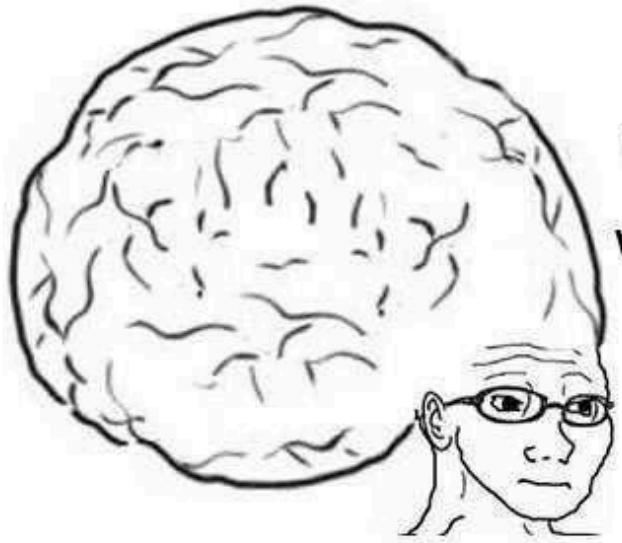
- Premature Optimization is the Mind Killer
- Learning Requires Mistakes

How?

- Optimize for **Small Feedback Loops**
- By default, *Show Your Work*
- Fidelity Of Planning Must Be Appropriate ("Flag Down The Beach")
- Resist BDUF ("Big Design Up Front")



Agile then



"Let's get a customer to sit with the team while we iterate on a solution"

Agile now

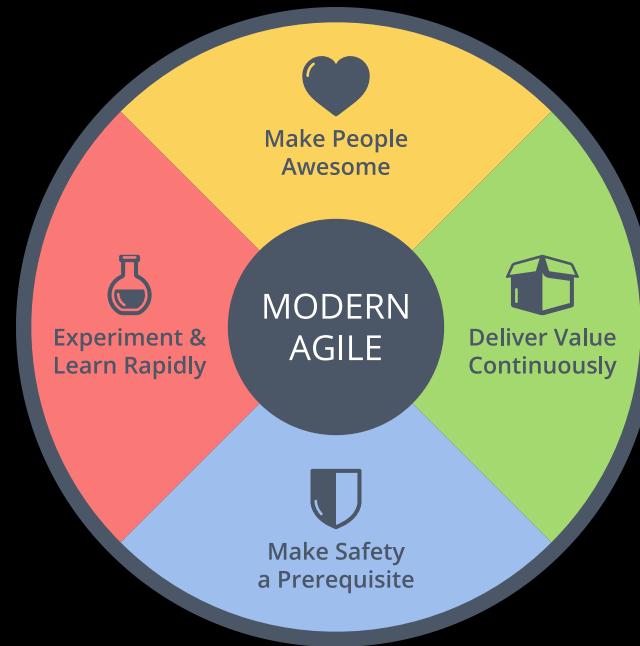
"Your card is in the wrong Jira template"

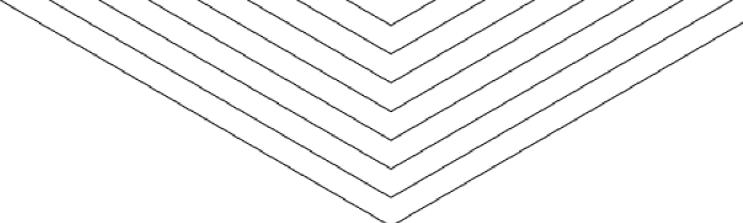


⚠️ SPOILER ALERT ⚠️

**"Agile" is the old word for
"Decentralization"**

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





Hypothesis:

- 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 this by **revisiting those Wheels and Solutions**
(which are feasible and cheap to implement), 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.)

So how exactly
does this work?



III. The Nitty-Gritties

THE PROBLEM

 Doing more work than we need to

THE TOOL

Convention over Configuration

- Always reserve the ability to configure based on local conditions, but otherwise
- Agree on conventions for the 80% of every project which is boilerplate

THE PROMISE

Minimize energy spent making trivial choices

THE PROBLEM

 **Unclear Definition of "Done" at the ~Epic level**

THE TOOL

Lean Hypotheses

We believe [TYPE OF USER]
has a problem [DOING THING].

We can help them with [OUR SOLUTION].

We'll know we're right if [CHANGE IN METRIC].

THE PROMISE

A shared, falsifiable understanding of "MVP"

THE PROBLEM

 **Unclear Definition of "Done" at a Story level**

THE TOOL

Well-Formed Stories

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*"

THE PROMISE

Cheap, simple confidence in shared expectations

THE PROBLEM

⟳ No sane defaults for team communications

THE TOOL

{FUTURE, PRESENT, PAST, CADENCE}.

Every team ought to agree on the *Canonical Source Of Truth* for:

FUTURE plans	Where do we keep them?	Usually a backlog. GH Projects? Jira? A legal contract?
Conversations in the PRESENT	How do we have them?	Slack? TG? Respond to emails that same day?
Work that's been done in the PAST	What does it look like?	Code repos? gDocs? ADRs?
CADENCE	What's our regular rhythm of ...	Responsible Recipe

THE PROMISE

{Minimal, necessary, sufficient} expectations for team communications

THE PROBLEM

 Not enough time for real work

THE TOOL

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.

THE PROMISE

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

THE PROBLEM

 Too many meetings

THE TOOL

MEMENTO MORI MEETING TYPES

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".
DO NOT CALL A MEETING	Literally anything else	email, information radiator, etc.

THE PROMISE

A shared, actionable, falsifiable rubric for "this mtg could have been an email".

THE PROBLEM

Understanding the status of work

THE TOOL

Conventional story types

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

THE PROMISE

{Minimal, necessary, sufficient} vocabulary to talk about work being "done".

THE PROBLEM

Which work ought we prioritize?

THE TOOL

Conventional story states

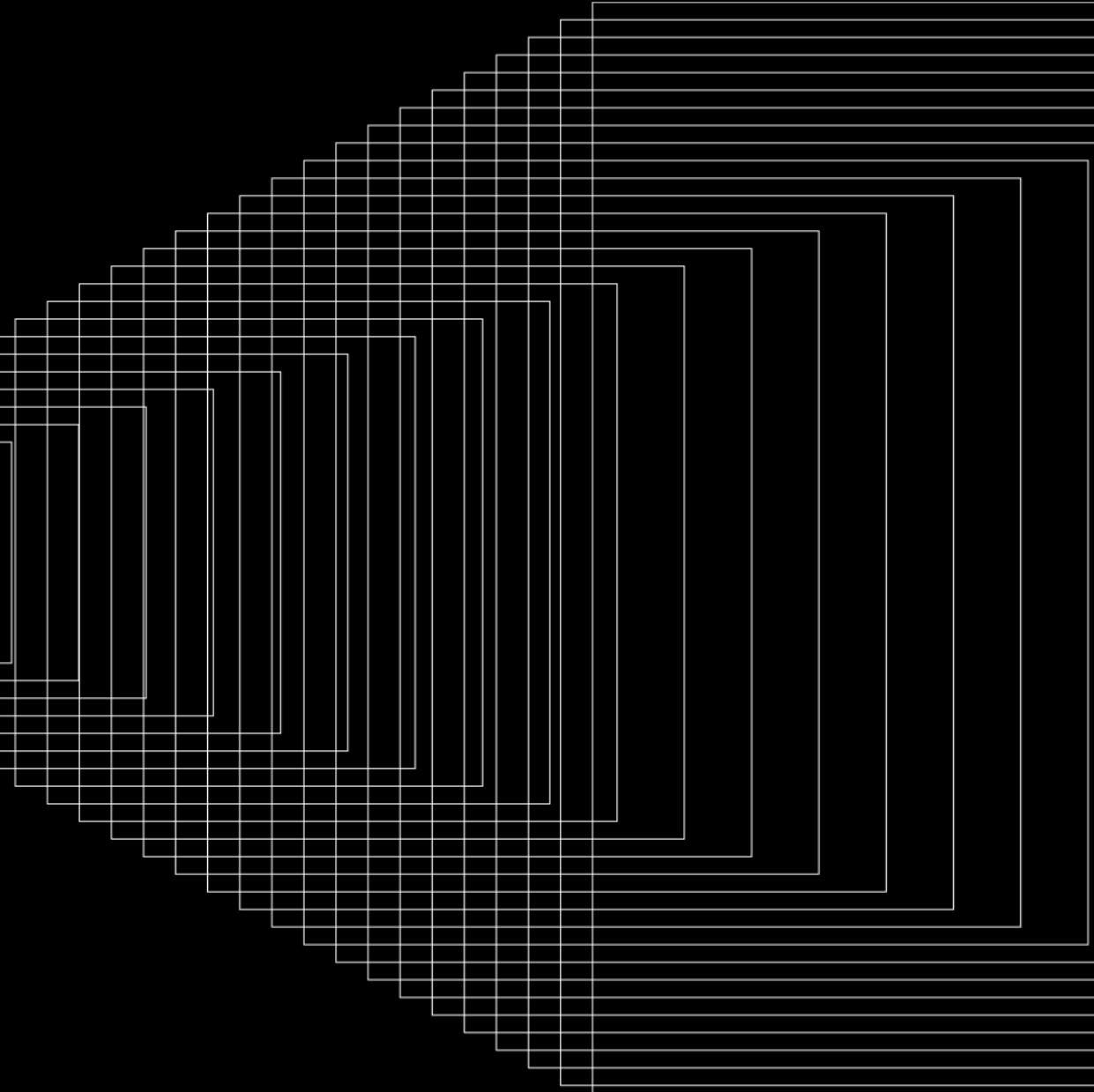
★ User story	Least amount of work w/ User-facing value
🐞 Bug	Regression (<i>not</i> new requirements)
⚙️ Chore	Anything w/o user-facing value
👑 Epic	Anything bigger than a story
🏁 Release Marker	Anything w/ a fixed time (scope or cost <i>MUST</i> float)

THE PROMISE

Optimize for "progress is the primary measured in working software"

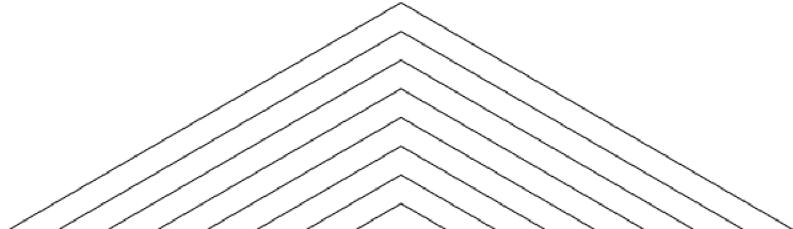
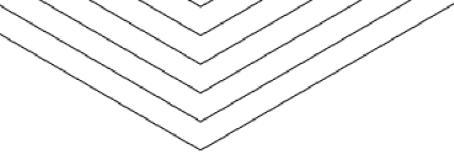
MORE TOOLS :

1. Software Craftsmanship
2. Y.A.G.N.I.
3. Continuous Delivery =>
FEEDBACK LOOPS
4. The Right Kind of Laziness
5. Focus on Business Value
6. Ubiquitous Language (Eric
Evans)



EVEN MORE TOOLS

"Done" means Done, Align the {Easy, Right} Thing to Do, Behavior-Driven Development (BDD), Big Design Up-Front ("BDUF"), Bikeshedding, Box' Law (_All Models Are Wrong; Some Are Useful_), Build ↴ Measure ↴ Learn ↴, Cargo Culting, Collective Ownership of Code, Comments Are Lies Waiting To Happen, Continuous Incremental Delivery, Convention Over Configuration, Definition of "Done", Deliver Value Continuously, Enablement && Execution, Escaping the Build Trap, Experiment & Learn Rapidly, Experiment Backlog, Form ↴ Storm ↴ Norm ↴ Perform ↴, FUEP (Functional – Usable – Emotional – Persuasive), FUTURE (Where do we keep plans?), Grow the product through continuous validation, Lean Hypothesis, Listen to yr users? Always. Believe them? Sometimes., Low Cost of Change, Make It {Work, Right, Fast}, Make People Awesome, Make Safety a Prerequisite, Minimal Standard of Care, Modern Agile, Outcomes vs. Outputs, PAST (Why did we do that?), Pivotal, PRESENT (How do we converse?), Progress Is Primarily Measured By Shipping {Code, Product}, R.A.C.I., Release early and often, Red ↴ Green ↴ Refactor ↴, Scrum, Short Feedback Loops, Small Batch Sizes, Software is a Team Sport, Story = Least Amt of Work w/ User-facing Value, Story = Placeholder for a Conversation, Strong Opinions Loosely Held, Sustainable Pace, Team != "Same ppl forever", Team == rotation && low-cost ramp-up, Team Matrix ratio, Technique: "Technique"!, Test-Driven Development (TDD), The Most Expensive Way To Learn Is By Building, Think ↴ Make ↴ Check ↴, Ubiquitous Language, Values vs. Recipes, W.A.Y.K., Well-Formed Stories, What's The Least We Could Do That Might Possibly Work?, XP (eXtreme Programming), Y.A.G.N.I. ("Ya Ain't Gonna Need It"), Yak Shaving, & Yesterday's Weather.



OH ONE MORE THING



GORDIAN is a brand new, TDD'd, modular, general consensus engine.

Drop-in Replacement for Comet BFT. Easily customize components such as consensus strategies, p2p networking algorithms, hashing schemes, and signature schemes. Supports multiple concurrent proposals and BLS signature aggregation. Addresses bottlenecks in mempool and peer-to-peer networking. Comprehensive documentation and sample implementations. Compatible with various back-end systems & optimizations, including different encoding formats and efficient hashing algorithms like Blake3. Implements ABCI++. Potential for 10-1000x performance improvements over existing solutions. Multiple concurrent proposers enable better scaling for blockchain networks. Easier to maintain than custom forks. BLS signature aggregation.

IV. The Recap

- 
1. As an ecosystem, we're being crushed by tech debt
 2. As an ecosystem, we're working harder than we need to
 3. As an ecosystem, there's competitive advantage in getting it right
 4. There are cheap, effective answers in the Old Ways
 5. Which of this makes sense to start trying on Monday?
 6. Follow up w/ Strangelove or the ICF to learn more or get help

Thanks!

- @strangelovelabs
- @sick_fade
- @reecepbcups' *Interchain Intro Hackmos Workshop*,
Friday, 10:30am

