

DevOps Foundations: Core Concepts and Fundamentals

Understanding Lean Software Development



Chris B. Behrens
Senior Software Developer

@chrisbbehrrens



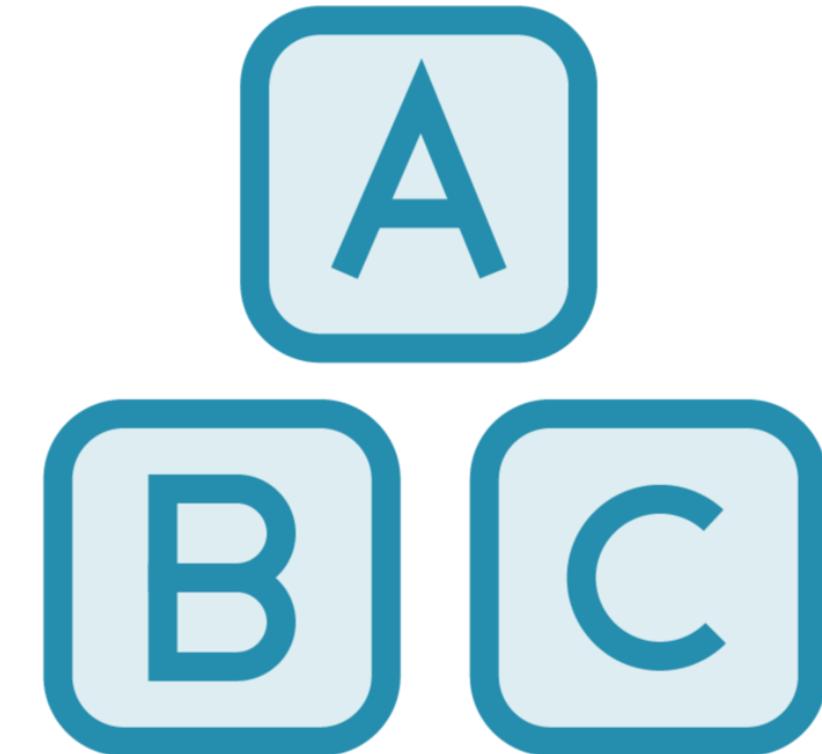
The Fundamental Truth of DevOps



**Upon reflection, some
big ideas**



**Big picture stuff that
would have been nice
to know from the
beginning**



**So that what you
learned thereafter
was placed in context**



In science and technology, we
grossly underestimate the
value of certainty.



Getting Started

Lean Development

Epistemology – “how do we know?”



Where Lean Comes From

[https://app.pluralsight.com/library/
courses/exploring-lean-principles](https://app.pluralsight.com/library/courses/exploring-lean-principles)



The Toyoda Family



This system is primarily the contribution of a single family

The story of a little boy and his mother

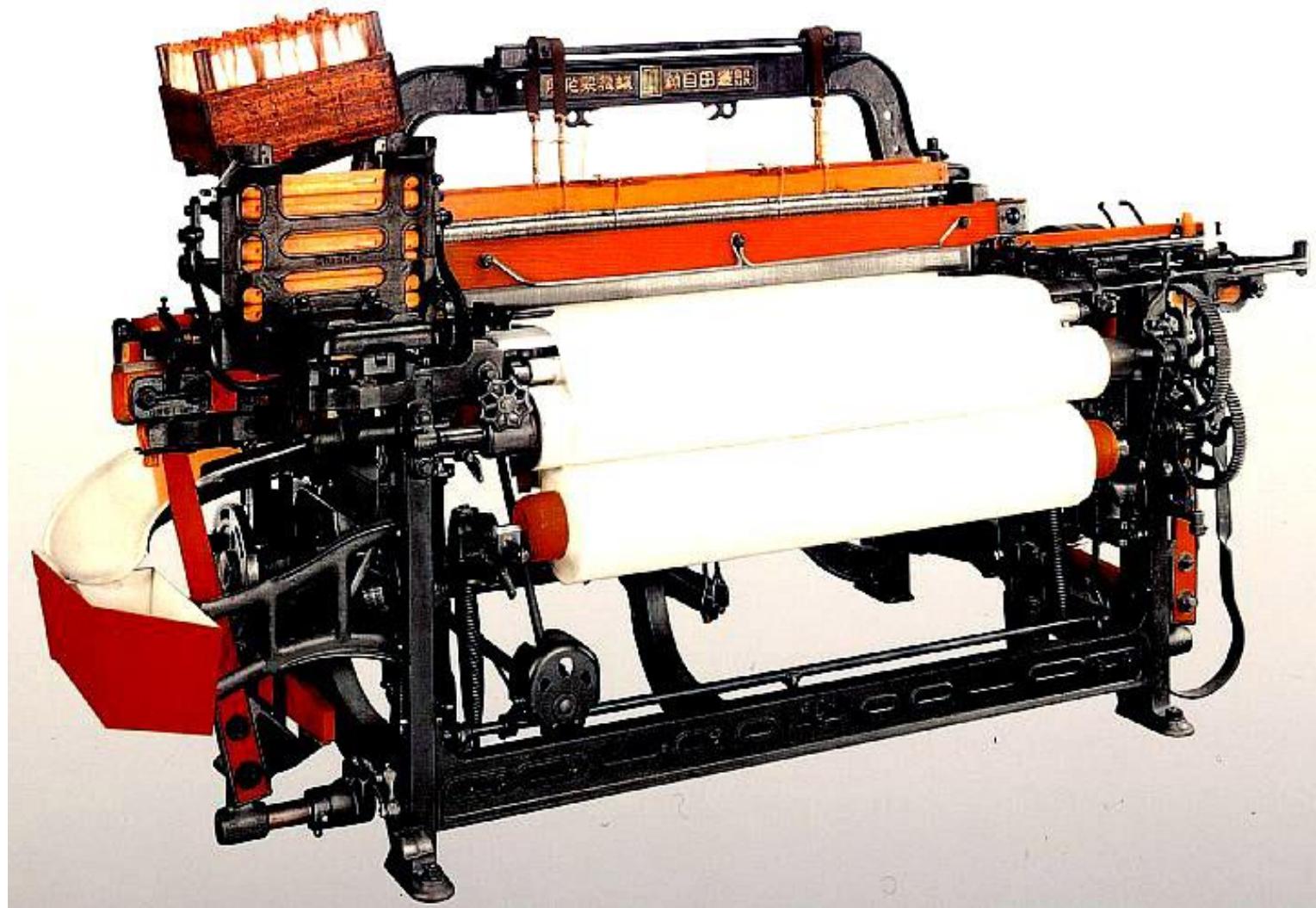
In Japan, in the era of the Old West

A carpenter father and a weaver mother

A boy that saw the repetition and waste in motions his mother carried out over and over again

His love for his mother placed the human being at the center of the analysis





Toyoda created a steam-powered automatic loom

One which could run attended through the night

“The Father of the Japanese Industrial Revolution”

Eventually, the looms became the business itself

Kiichiro, the son, loved engines, so the company pivoted to automobiles



A Quick Aside

丰田
トヨタ

“Toyoda” (Kanji)
“Toyota” (Katakana)



The Obstacle Is the Way

Japan was in ruins

Partly because of
mass production

Japan could not
mass produce (yet)

“Just-In-Time”

Rather than aiming
for speed, focus on
eliminating waste

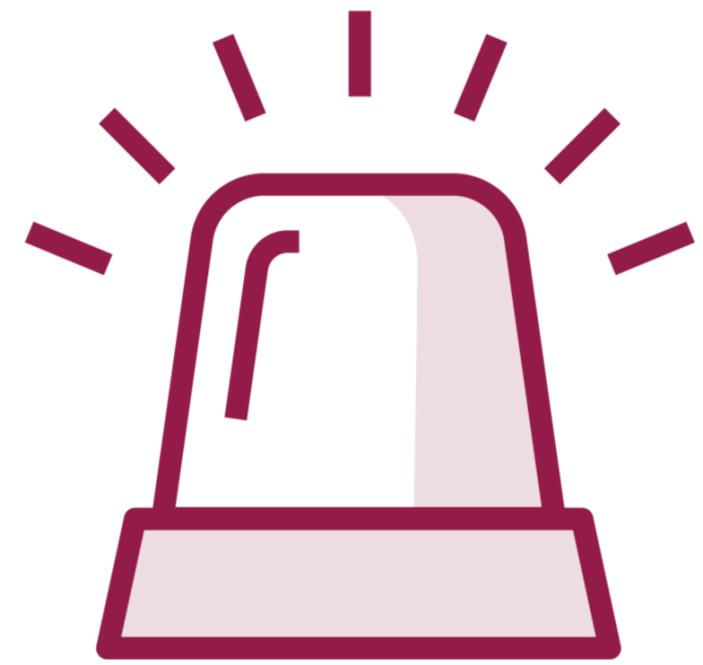
The Toyota
Production System



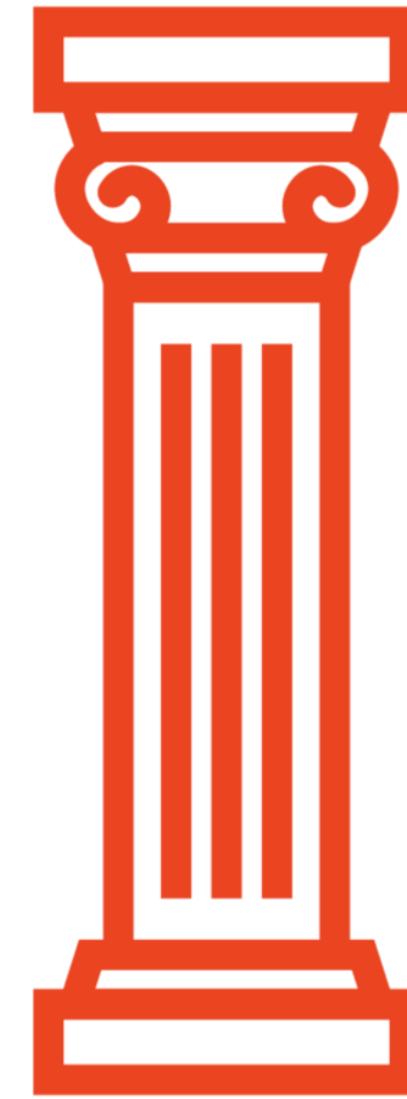
The Two Pillars of the TPS



Just-In-Time



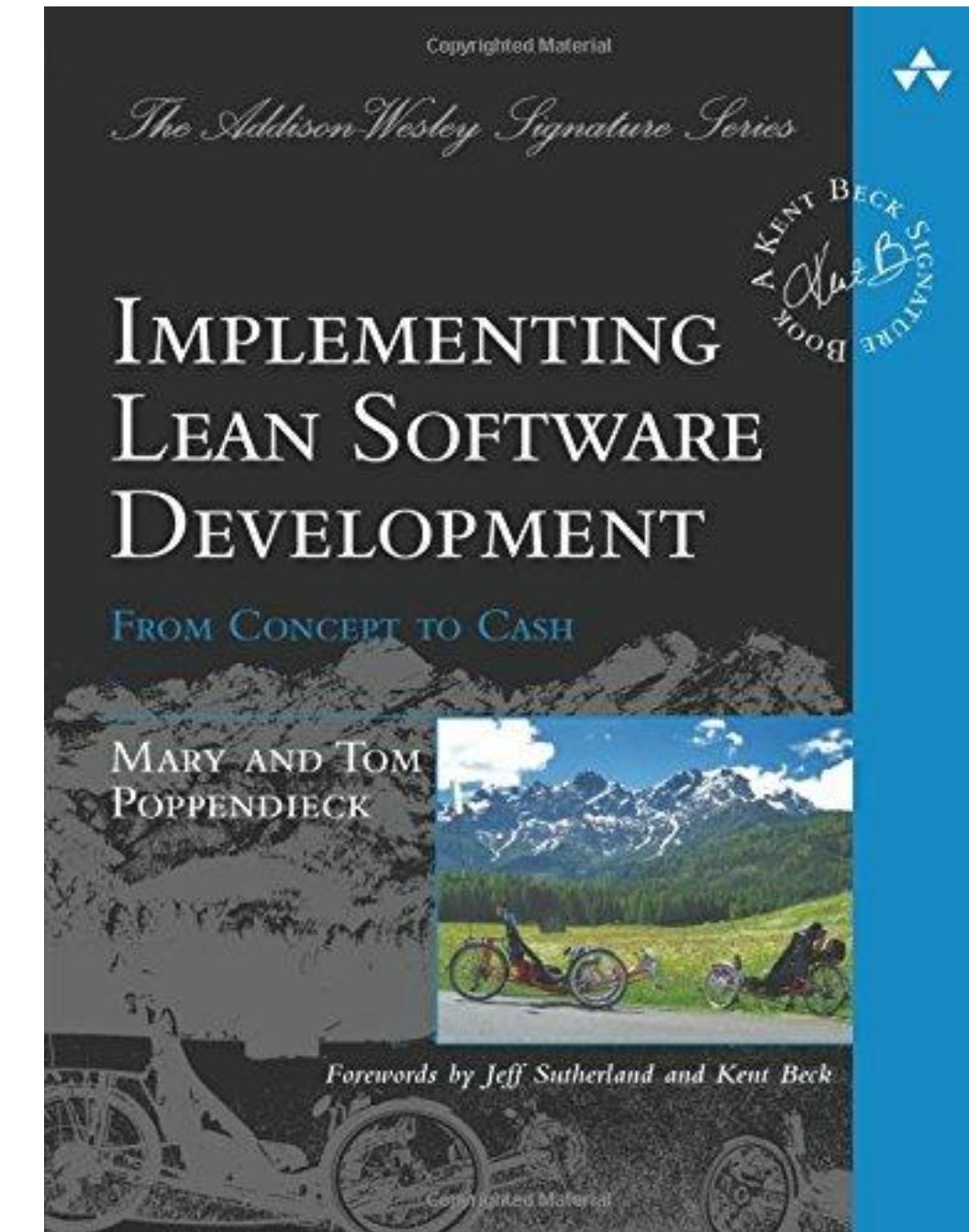
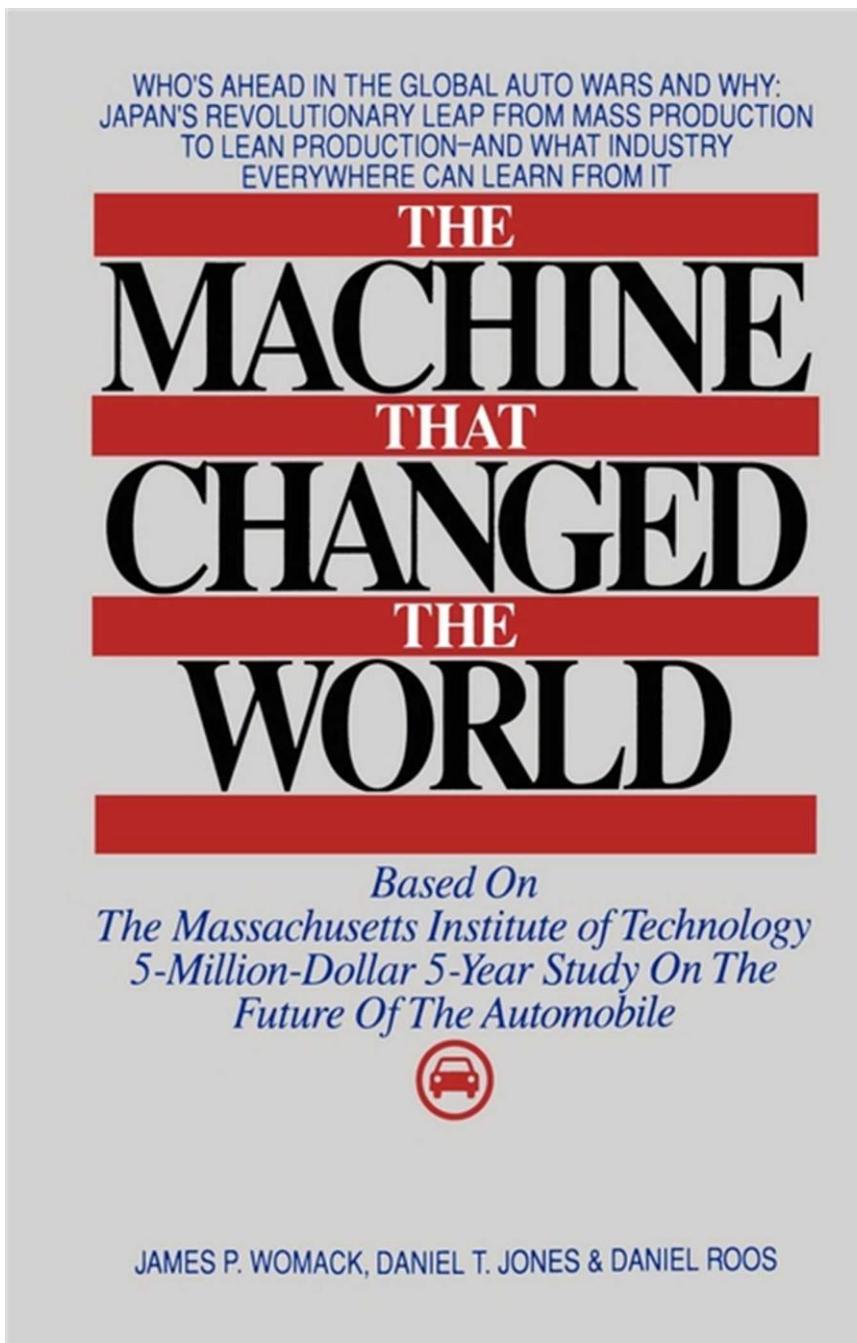
Andon



Jidoka



Lean Production Becomes Lean Software Development



The Principles of Lean Development



If you aim at speed, you may
get speed, but you'll get waste.
If you aim at the elimination of
waste, you'll eliminate waste
AND get speed.



Shift Left

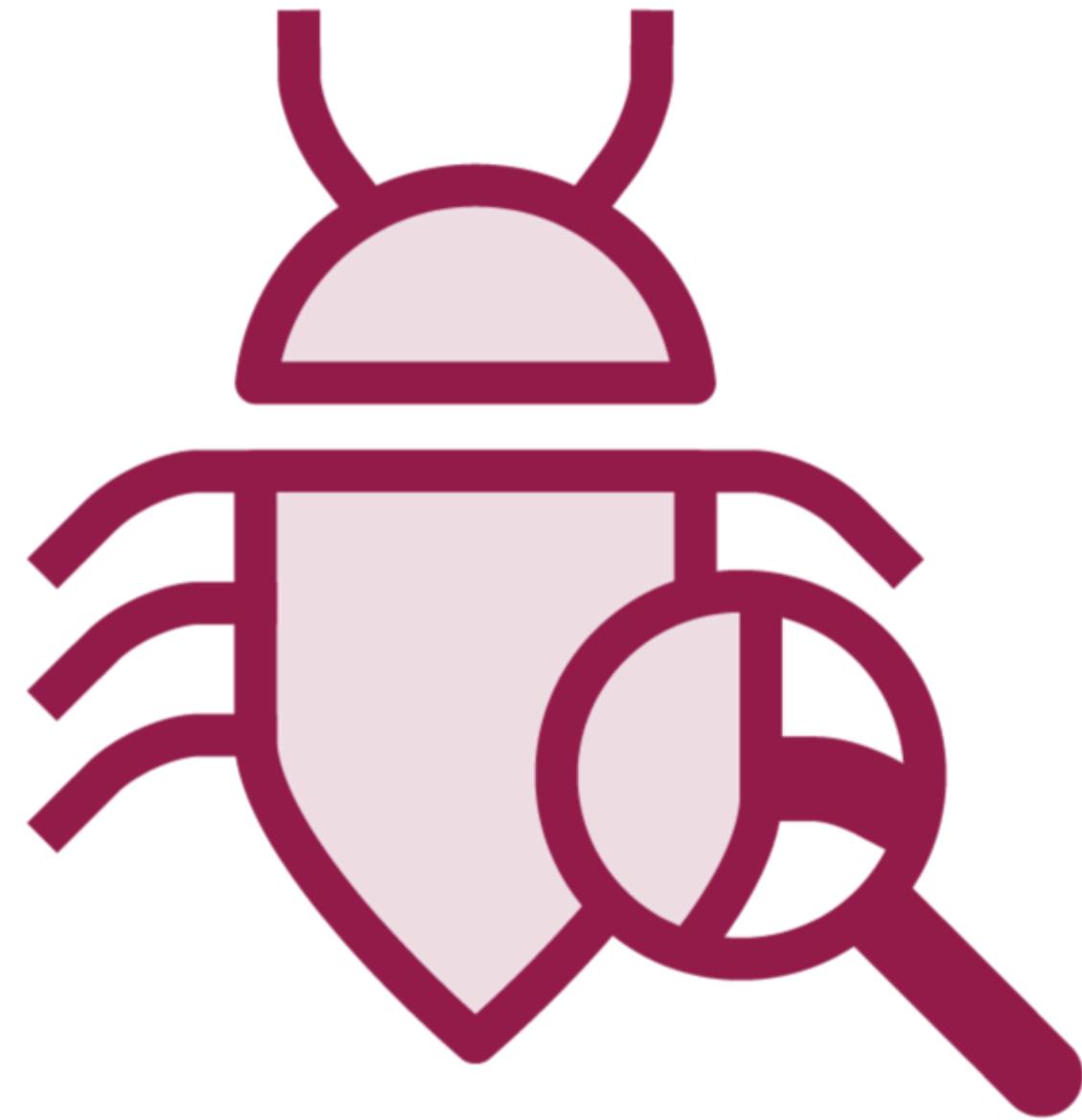
Bug caught by the customer

Bug caught by QA

Bug caught by code review

Bug caught by developer





The final level

Write the test first

Nothing is error-free

And you can only anticipate what you can anticipate

Test-first makes the code more testable (duh) and makes you focus on what you can know for sure



The Seven Principles



Eliminate Waste

What is “waste”?

**The time spent
fixing a bug after
the fact is waste**

**Human repetition
is waste**



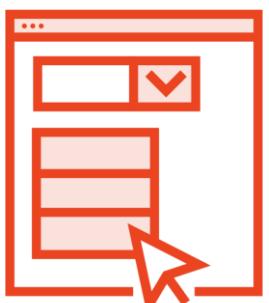
Build Quality In



Inspection to FIND defects

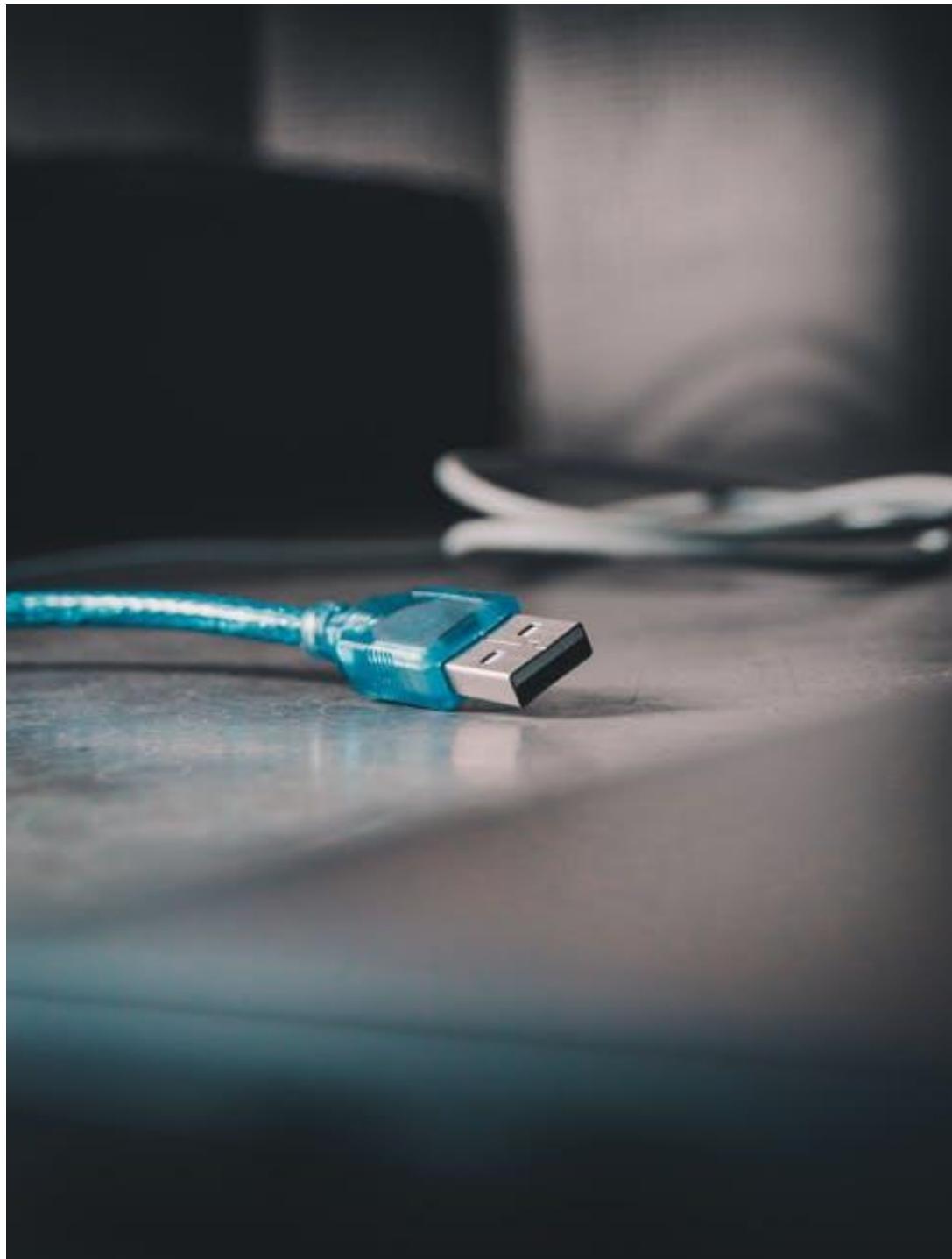


Inspection to PREVENT defects



Online forms use this approach extensively





Poka-Yoke

“error avoidance”

Selecting your choice from a limited UI domain

Poka-yoke is present everywhere

DON’T STICK A FORK IN THE POWER SOCKET

But if you do, there’s a good chance that a GFCI will break the circuit before it kills you

Manual transmissions make you press in the clutch before you start the car

And you can’t plug the USB connector in the wrong way



What This Means for DevOps



Least Privilege Principle



**What you can't do, you can't do
mistakenly (or maliciously)**



Poka-Yoke in Version Control



Quality and Testing

Code has premises

**Tests ARE the
explicit premises**

**This prevents
defects now and
forever**



In science and technology, we grossly underestimate the value of certainty.

“The sky is blue”

“It is certain that this is true”



Knowing Whether a Release Is Ready



“The release is ready”

“It is certain that this is true”



The Automation of Knowledge Creation



Human testing is of limited usefulness

People are not the problem, software is

“Software performance is discontinuous across a given input domain”

Change the software, and ALL tests generally need to be re-run

Software is better at doing everything over and over again than people are



Predictably Unpredictable

“We shouldn’t be surprised that we’re surprised”

**Human beings are
really bad at
accepting
predictable
unpredictability**

**“Do better next
time”**



DevOps, Lean, and Agile in the
broad sense are all just
systems to force you to stop
pretending that you know more
than you really do.



Creating Knowledge by Creating Software



Embrace uncertainty

“Agile is Utopian”

Agile was created by those of us who were bitter and disappointed and were ready to accept a hard reality

The schedule is only clear in retrospect, or when the project is 75% done

SOFTWARE IS RESEARCH

A problem that can only be wholly defined after it has been partially attempted



“Epistemic Humility”

The quality of our knowledge
is poor

So, we need to plan with that
in mind



Defer Commitment



Big Design Up Front – BDUF

Favors early commitment at the expense of predictability

Because information increases the further you go

Predictions reduce predictability

Irreversible and reversible decisions

Irreversible decisions commit to working with their consequences

Reversible decisions you can make whenever you want

“Decide early and often”



Change the Decision Type

**Choose a
reversible choice
over an irreversible
one**

**This is why travel
services offer
travel insurance**

**Amazon and other
online retailers
were built on easy
return**



Deliver Fast



Two different kinds of “fast”

Move quickly

Deliver early

Facilitate feedback

Deliver often

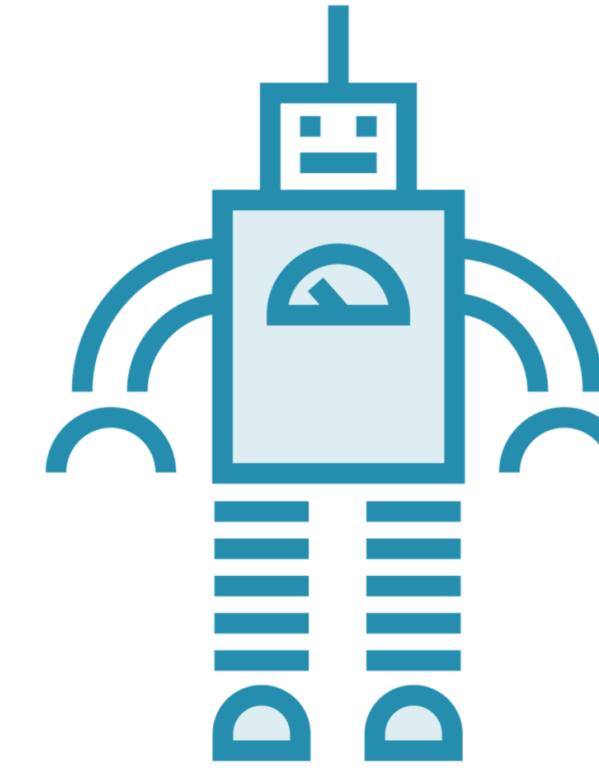
Compare a quarterly release schedule to a weekly one



The Reality of Weekly Releases



**Weekly means doing
different things**



**Things that didn't
make sense to
automate will now be
automated**



If it hurts, do it often



Respect People

The centrality of the human being

Because the human being was Mom

Other process mavens at the time were less worker-oriented

But the TPS has “Respect People” as a primary principle



“Top managers typically possess superficial, casual definitions of “Respect for People” such as fairness, civility, or listening...This is a severe misjudgment.”

“It is not a conveyer that operates men, while it is men that operate a conveyer, which is the first step to respect for human independence.”



The Case for “Respect People”

Reduces turnover

Treats people with
dignity and
decency

The numbers just
say that this works
better



Optimize the Whole



I was pitching test-driven development
“Our release cycles already take too long; this would add so much time to development”.

Doing TDD would take longer
But this was an illusion
Automated testing would let us do more and faster for less money
But the manager was focused on optimizing one part of the system





“The number one mistake of star engineers is optimizing a thing that shouldn’t exist”.

“The best part is no part. The best process is no process. It weighs nothing. Costs nothing. Can't go wrong.”



The Seven Wastes



Partially Done Work

“It’s 90% done”

Which means that
half of the work
remains

The developer is
not lying

But he's thinking
only of the code

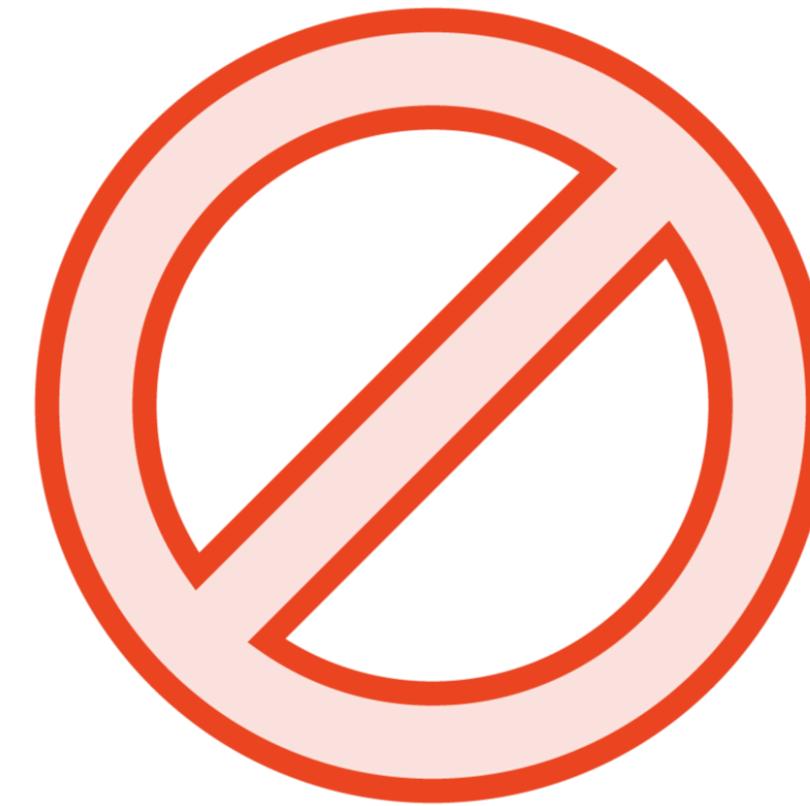
Code without tests
(and other stuff) is
incomplete



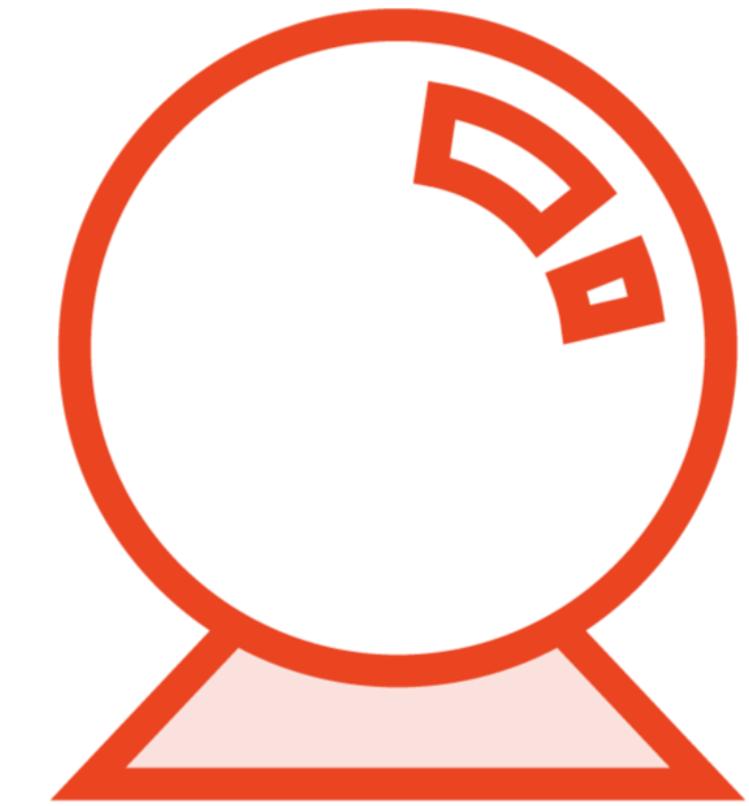
Extra Features



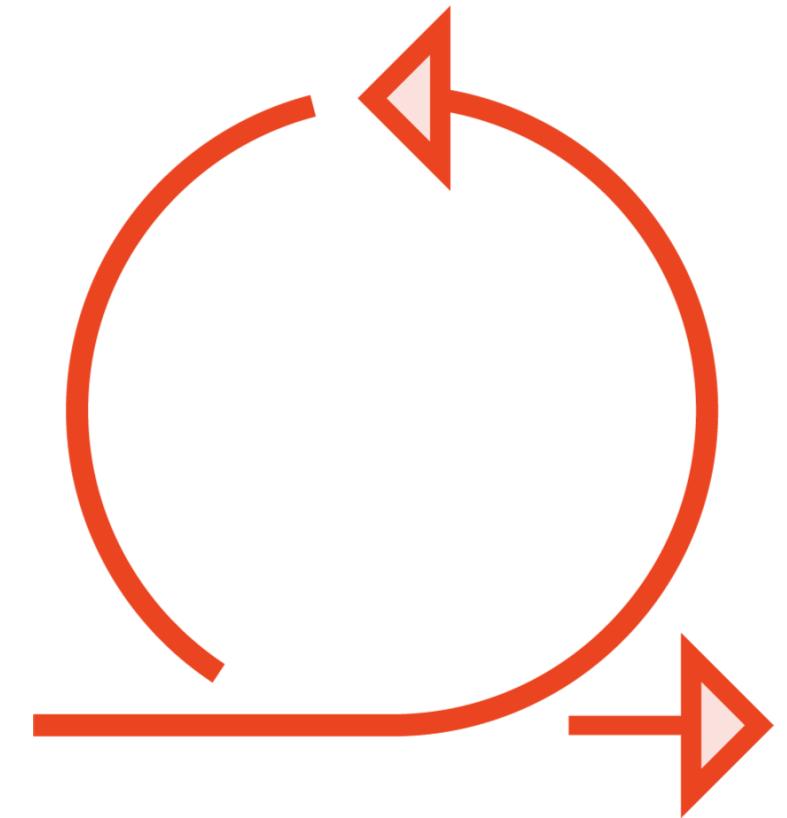
A feature produced at the wrong time



The right time might be “never”



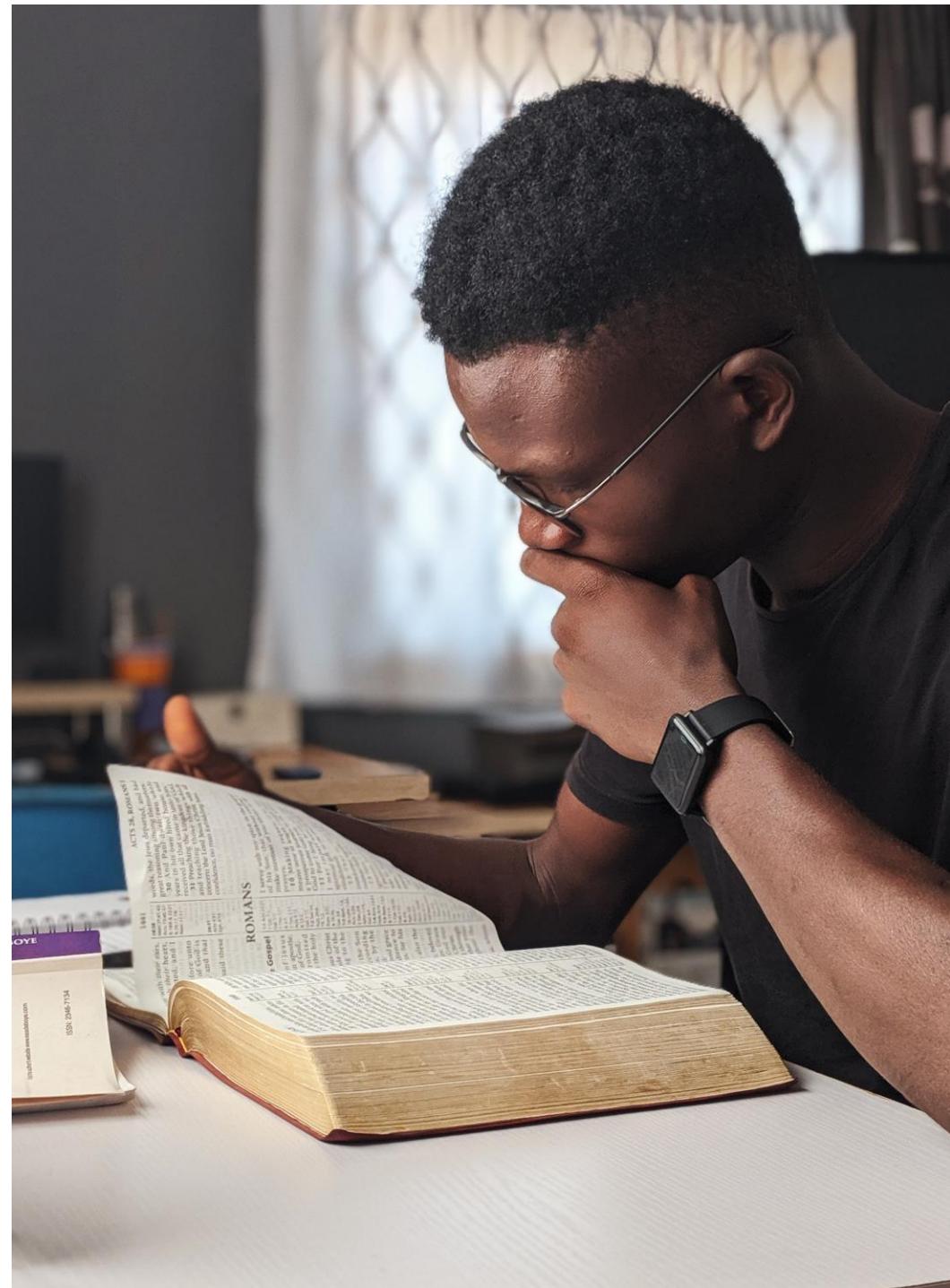
Avoid creating features “just-in-case”



Focus that effort on making your commitments deferrable



Relearning



The acquisition of knowledge which has happened before

Something YOU learned before...

Or a knowledge transfer

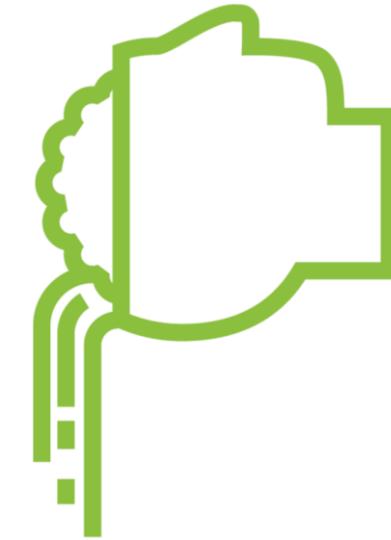
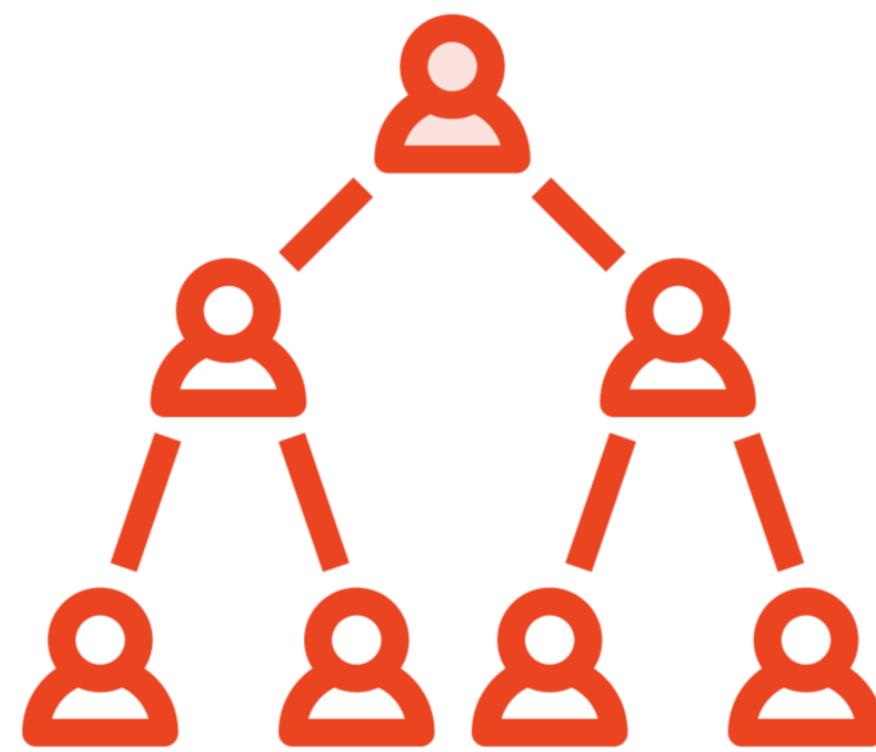
Some turnover is inevitable

The solution is to effectively Create Knowledge

This can take many forms



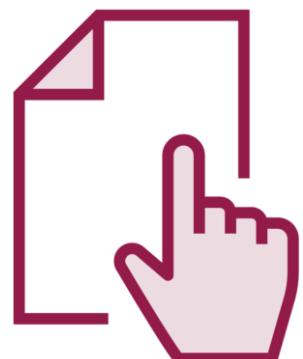
Handoffs



Handoffs



1. We could have recorded the hand-off sessions



2. We could have been more deliberate about cross-training



3. The company could have worked harder to hold on to us



Task Switching

People think they can multitask

They can't

**Things just don't work they way
people imagine they do**

The penalty is (at least) 40%



Delays

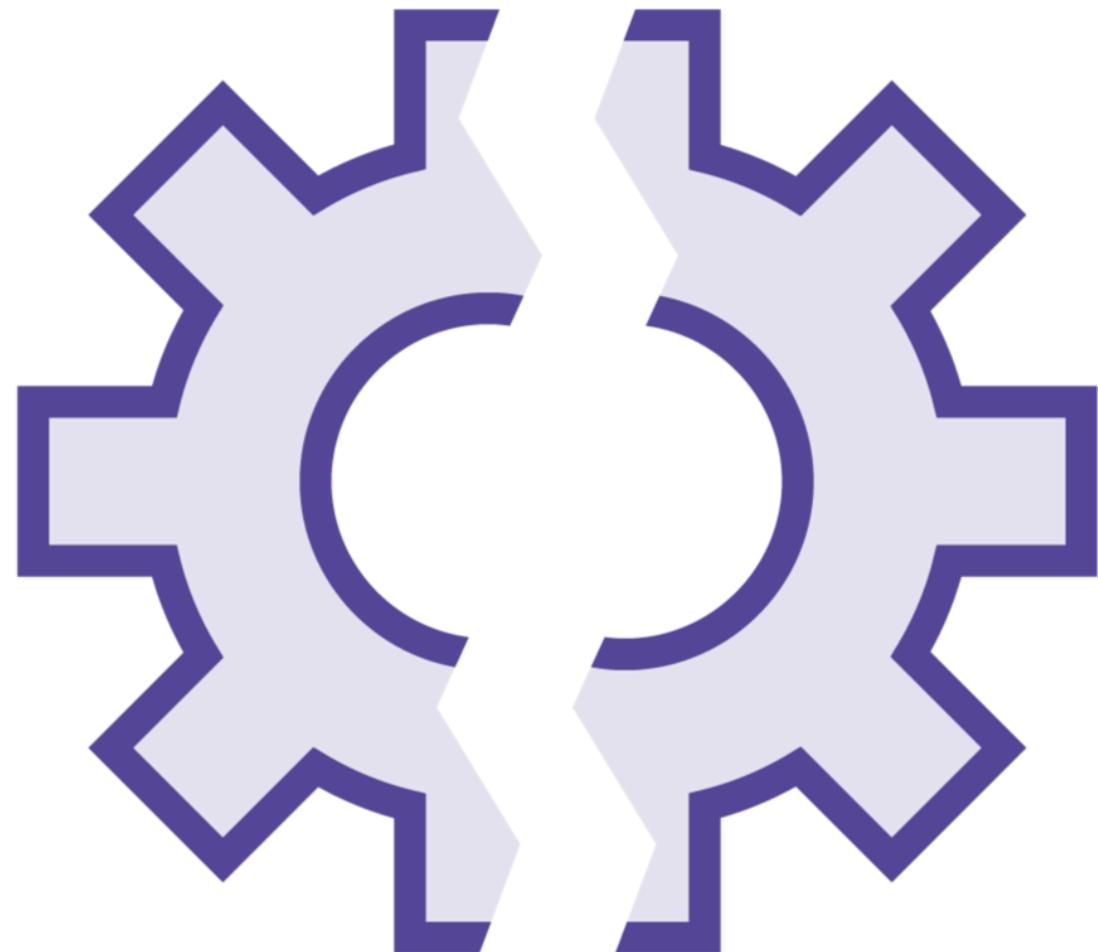
Partially because authority is invested at too high a level

Also because of siloed communication

Or because commitment was not adequately deferred



Defects



Like nothing else, defects derail your process

They impose task switching penalties

Assuming that the original developer is available, otherwise there's relearning or handoff waste

Defects are a sign that you're not managing the other wastes



Creating Quality in a Lean Context



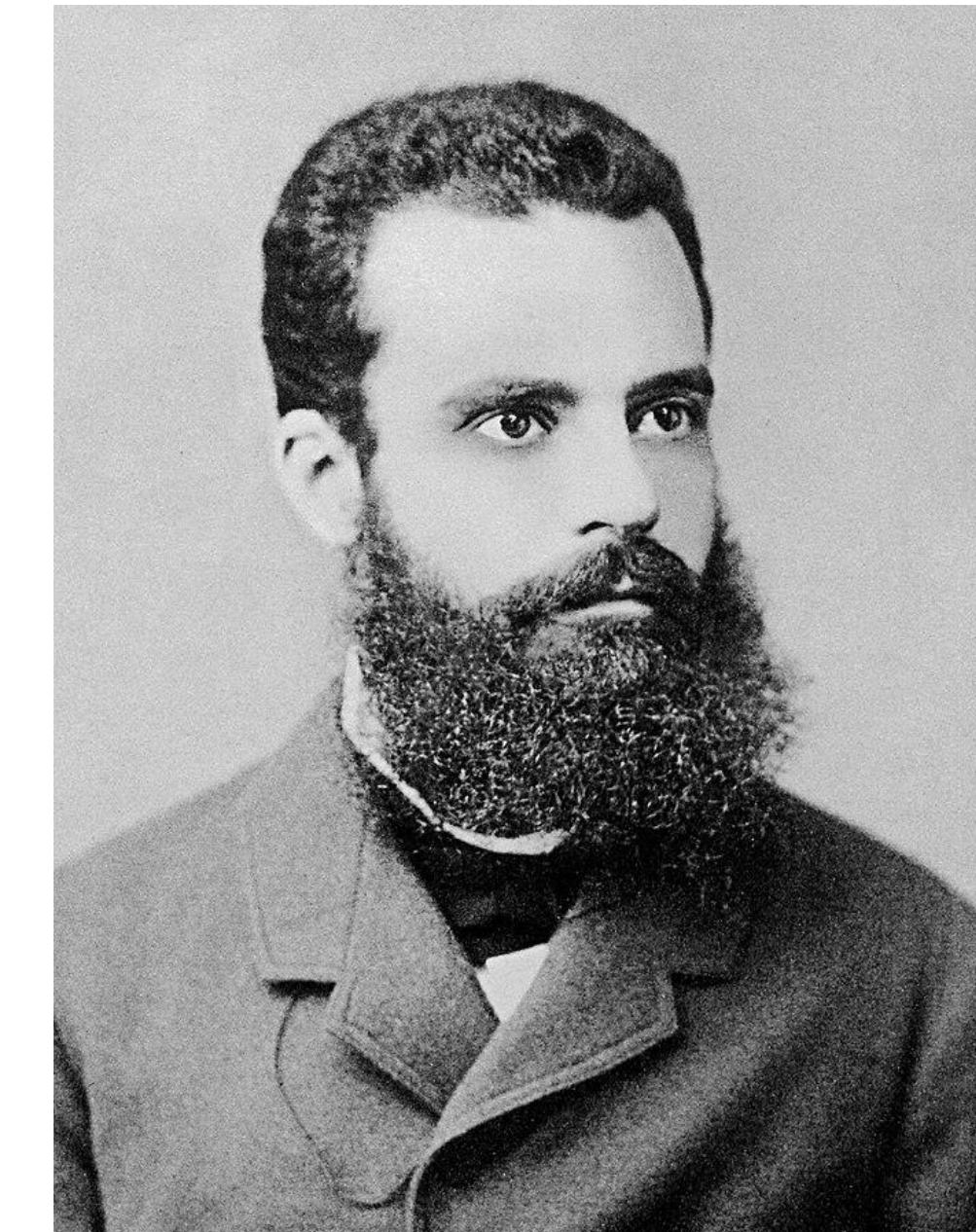
Pareto Analysis

“80% of the consequences come from 20% of the causes”

Reduce your defect rate using Pareto Analysis

Most problems in the code come from one “bad neighborhood” or a few

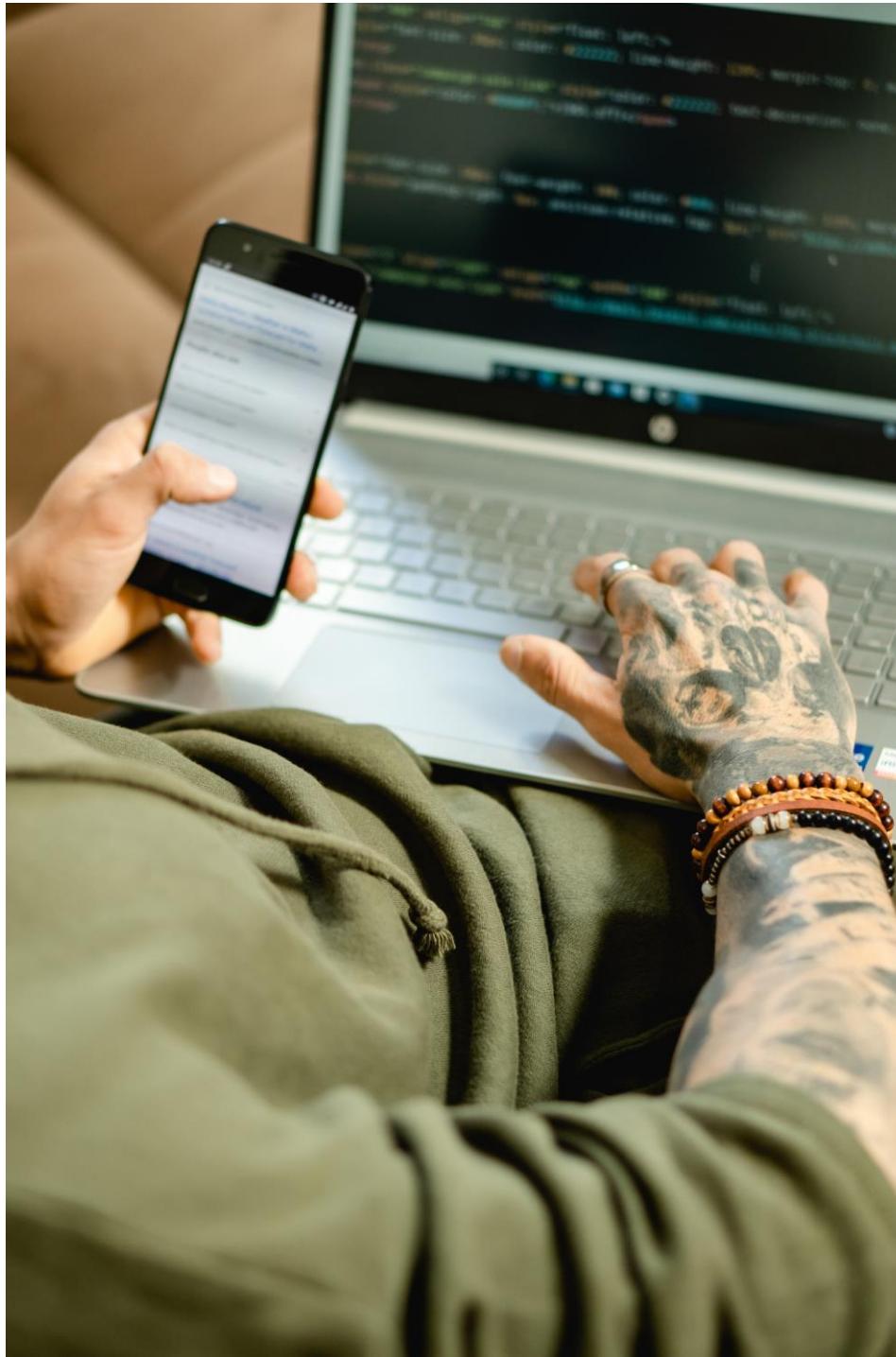
That bad neighborhood is where you need to go to move forward



Vilfredo Pareto



Optimize the Human Experience



Use human measures

A project where they wanted me to do a bunch of advanced stuff

When their developers couldn't even debug locally

Respect People would have told us to focus on that

Complaints from support engineers were made top priority

This meant that their support tools were always top quality



Create Interoperability

Similarity is the enemy

Compose the one, true build

**Reorganize the code to work
with it**

**Cross-train engineers to front-
load handoffs and minimize
relearning**



Summary



The Toyota Production System

Lean Manufacturing

Lean Software Development

The Seven Principles of Lean

Development The Seven Wastes

My Standard DevOps Triage

- For new customers

<https://app.pluralsight.com/library/courses/exploring-lean-principles>

