

MADS

S11: Lean Software Development

(Bloque 4 - Lean y Kanban)

El tweet de la semana

Kevlin Henney
@KevlinHenney

consultant · father · he/him · human (very) · husband · itinerant · programmer ·
keynote speaker · technologist · trainer · writer

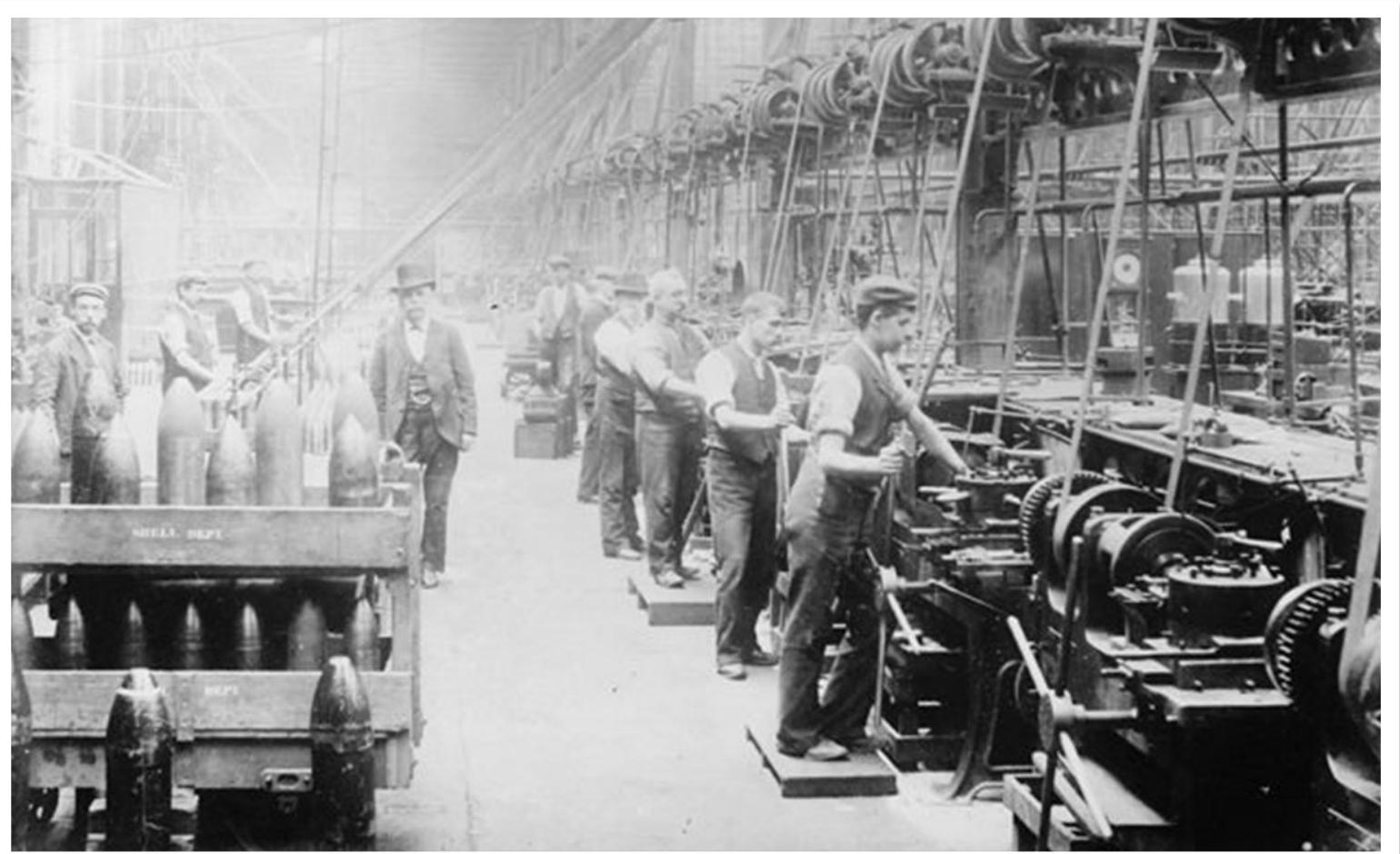
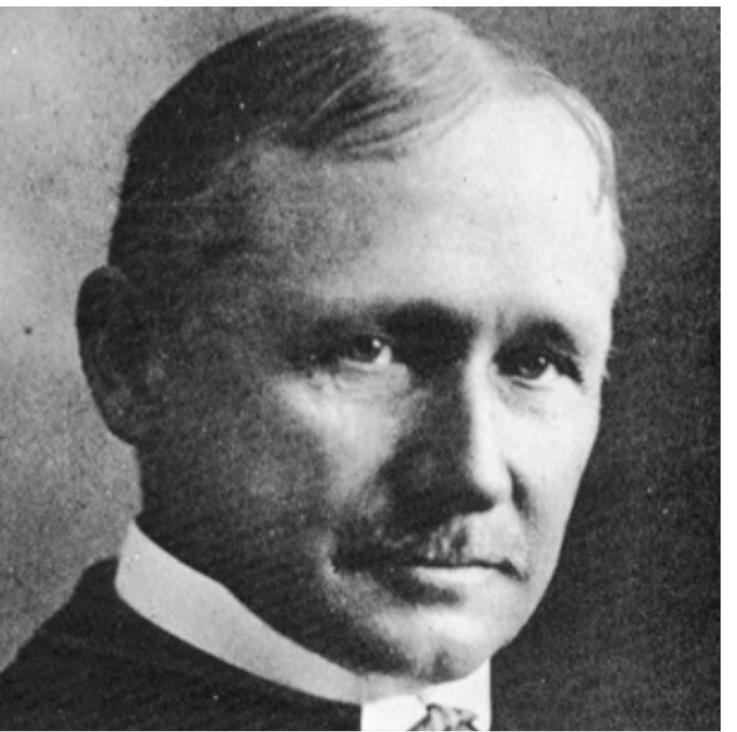
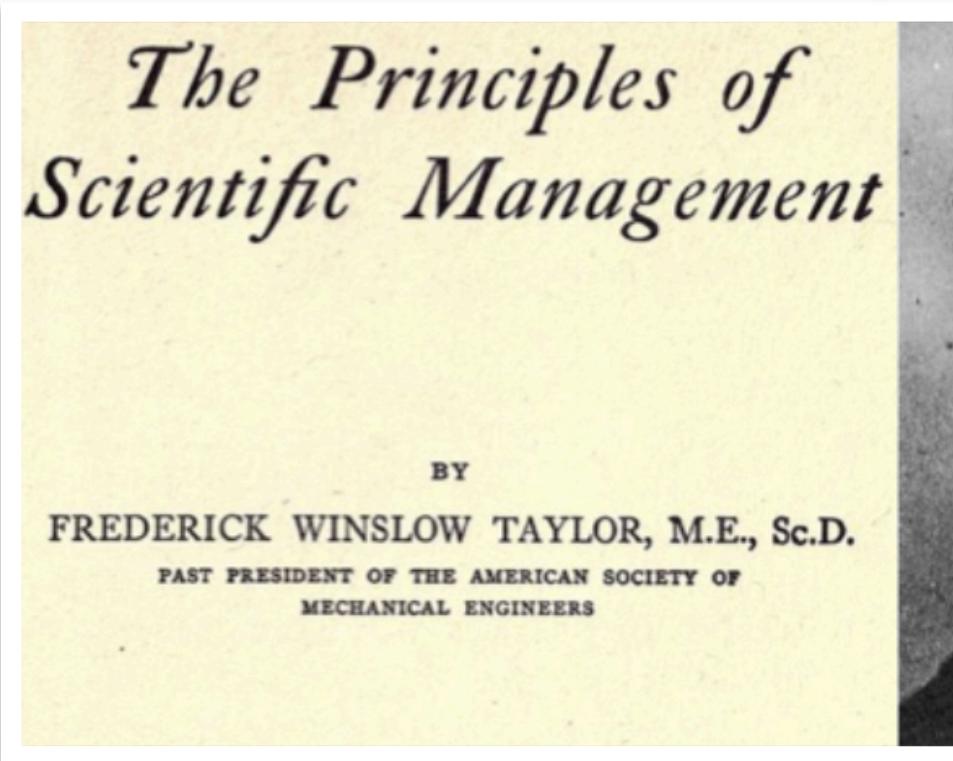
⌚ ①+~1au ⚡ mastodon.social/@kevlin 📅 Joined June 2009

1,441 Following 21.2K Followers

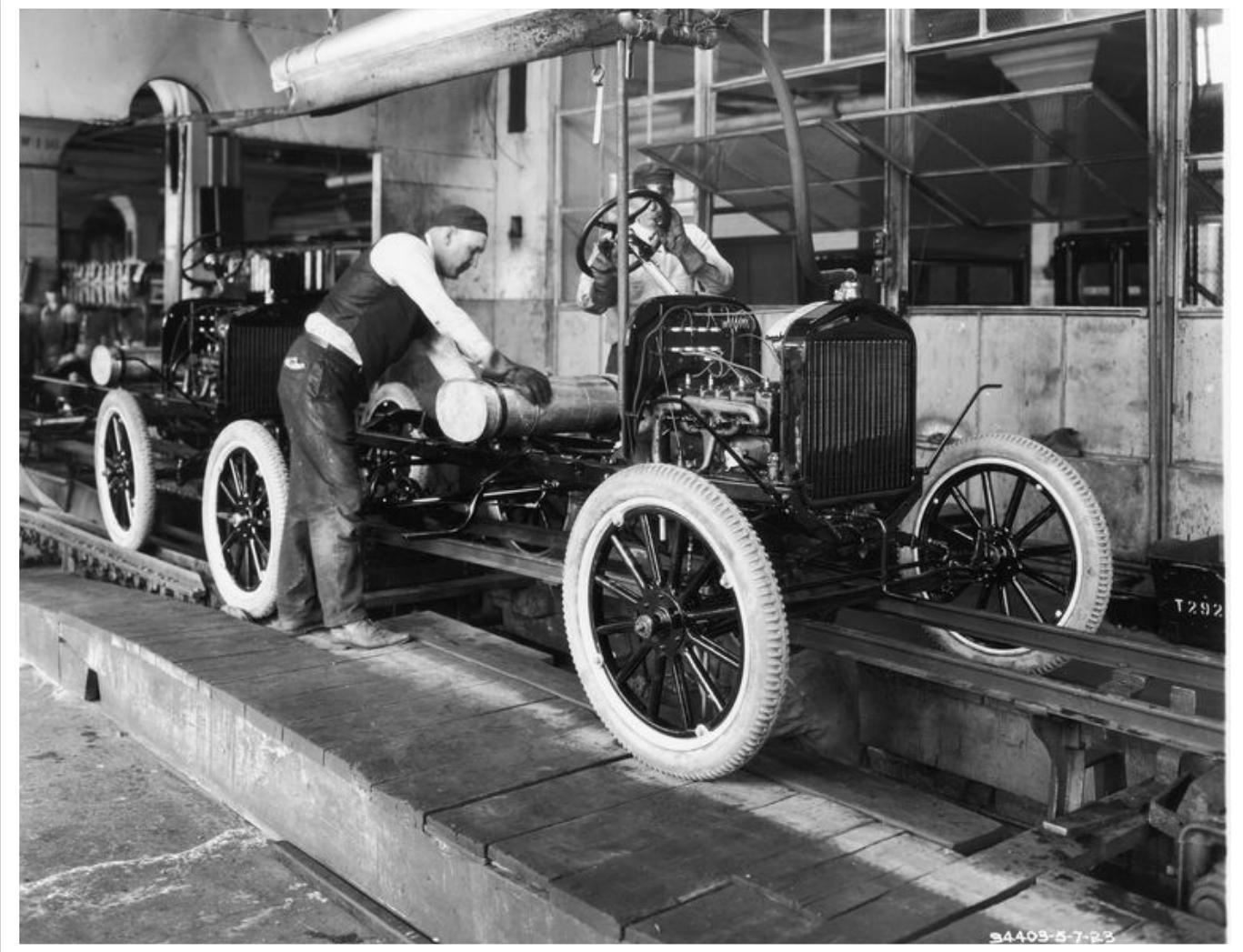


Orígenes de los sistemas de fabricación lean

Frederick Taylor - Taylorismo (1880)



Henri Ford - Fordismo (1910)



Ford Modelo T



- 15 millones de unidades (1910-1927)
- 10.000 coches diarios en 1925
- En 1920 la mitad de coches del mundo eran Fords

Japón 1945



- Falta de efectivo para financiar los sistemas de producción habituales basados en la acumulación de grandes inventarios.
- Falta de espacio para construir fábricas cargadas de inventario.
- Falta de recursos con los que construir los productos.
- Alto desempleo y gran cantidad de mano de obra, que hacía innecesaria la optimización de los procesos de fabricación.

Taiichi Ohno - Toyota [1950]



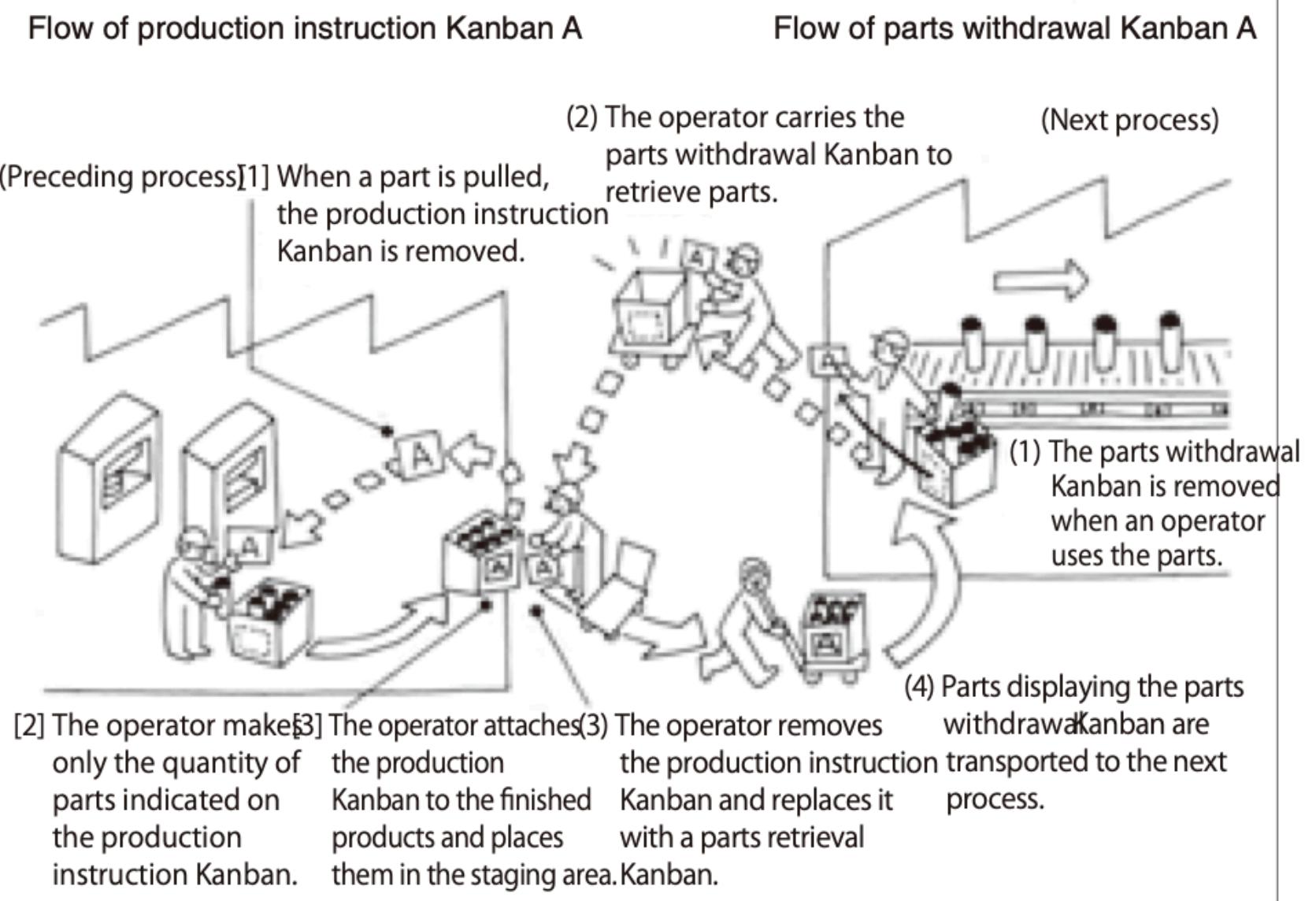
Toyota Production System
(TPS)

Pull system of production adopted (1948)



Flow of production instruction Kanban A

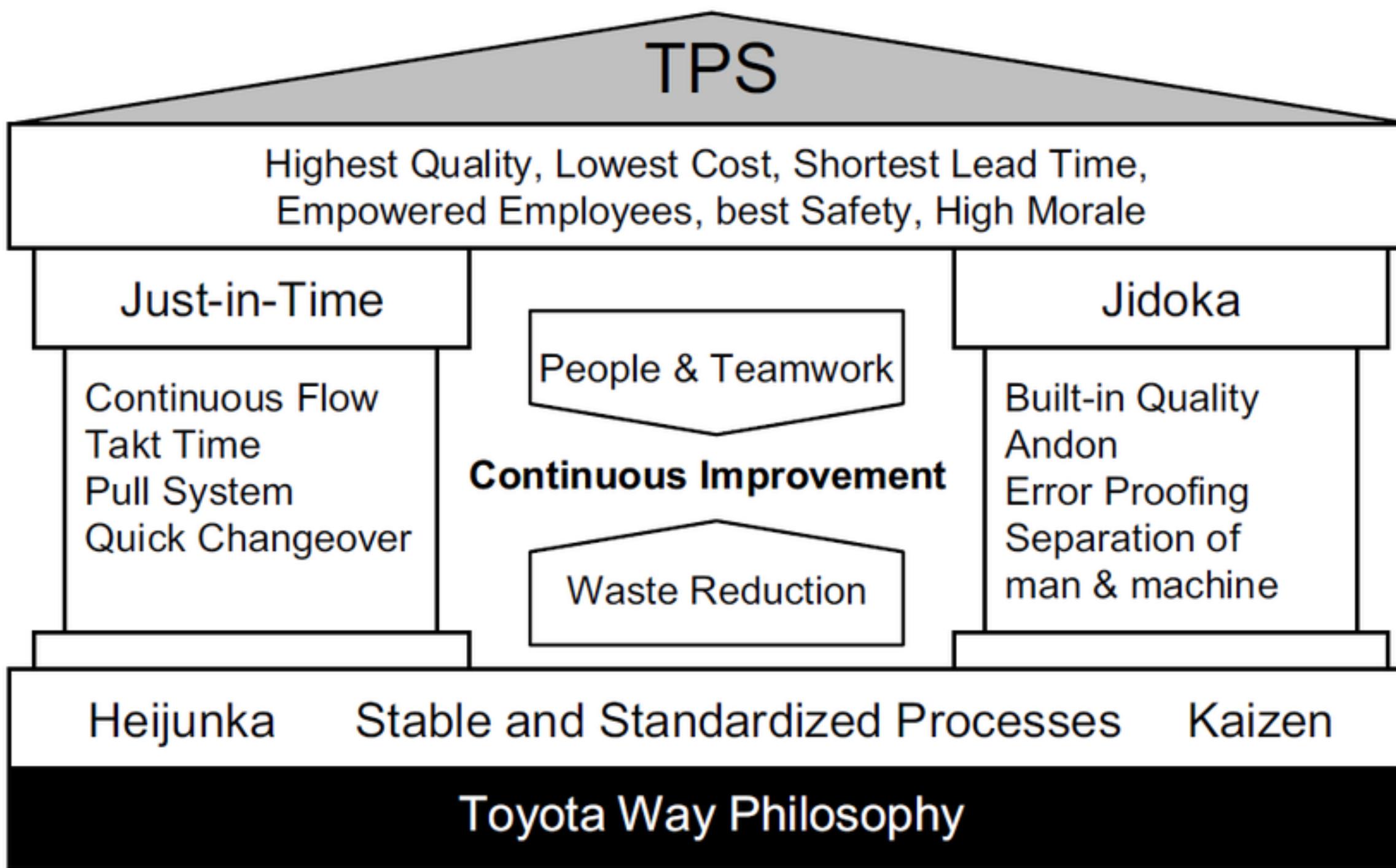
(Preceding process)[1] When a part is pulled, the production instruction Kanban is removed.



Toyota Production System

The TPS is a framework for conserving resources by eliminating waste. People who participate in the system learn to identify expenditures of material, effort and time that do not generate value for customers.

Toyota *Toyota Production System*

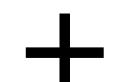


Automatización con supervisión humana

自働化

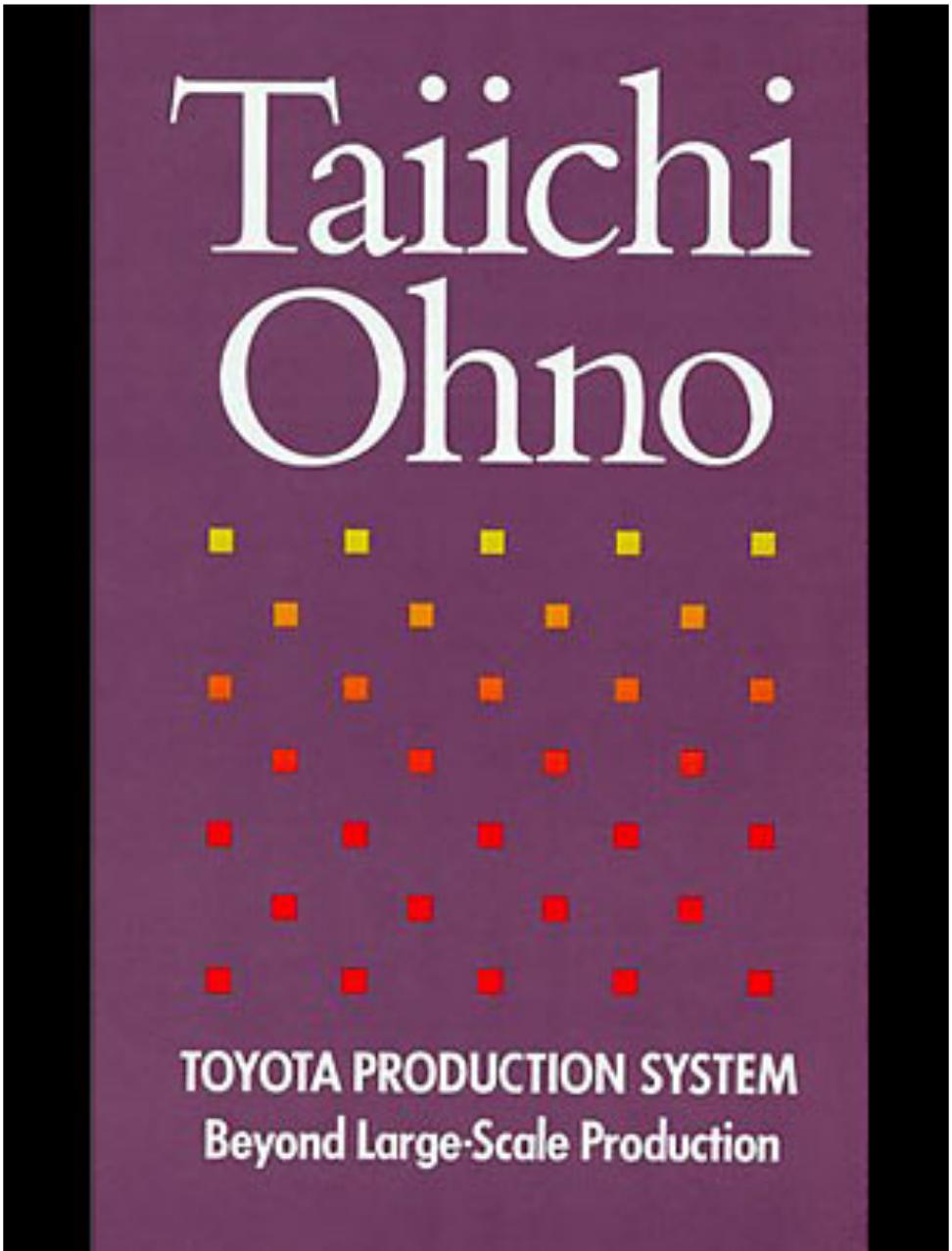


自動化
Automatización

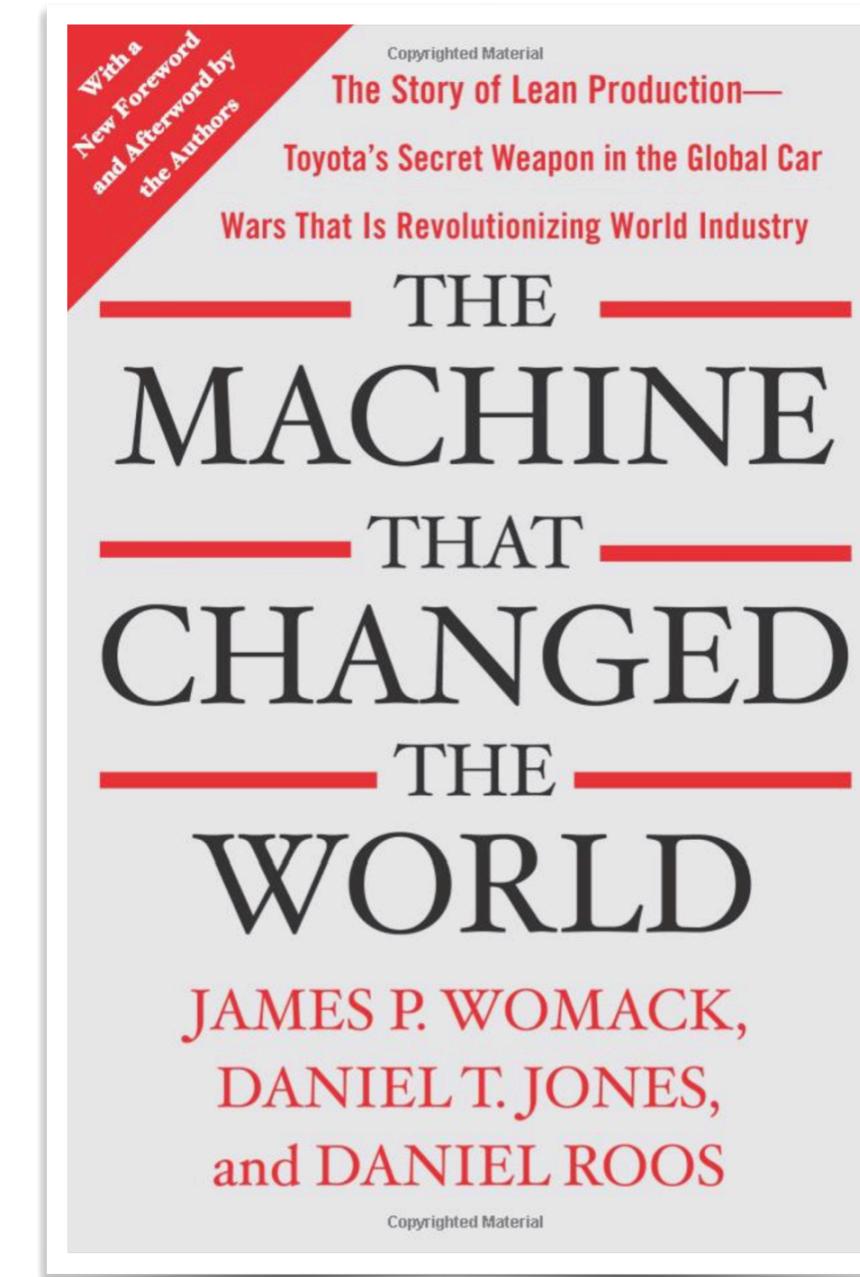


人
Persona

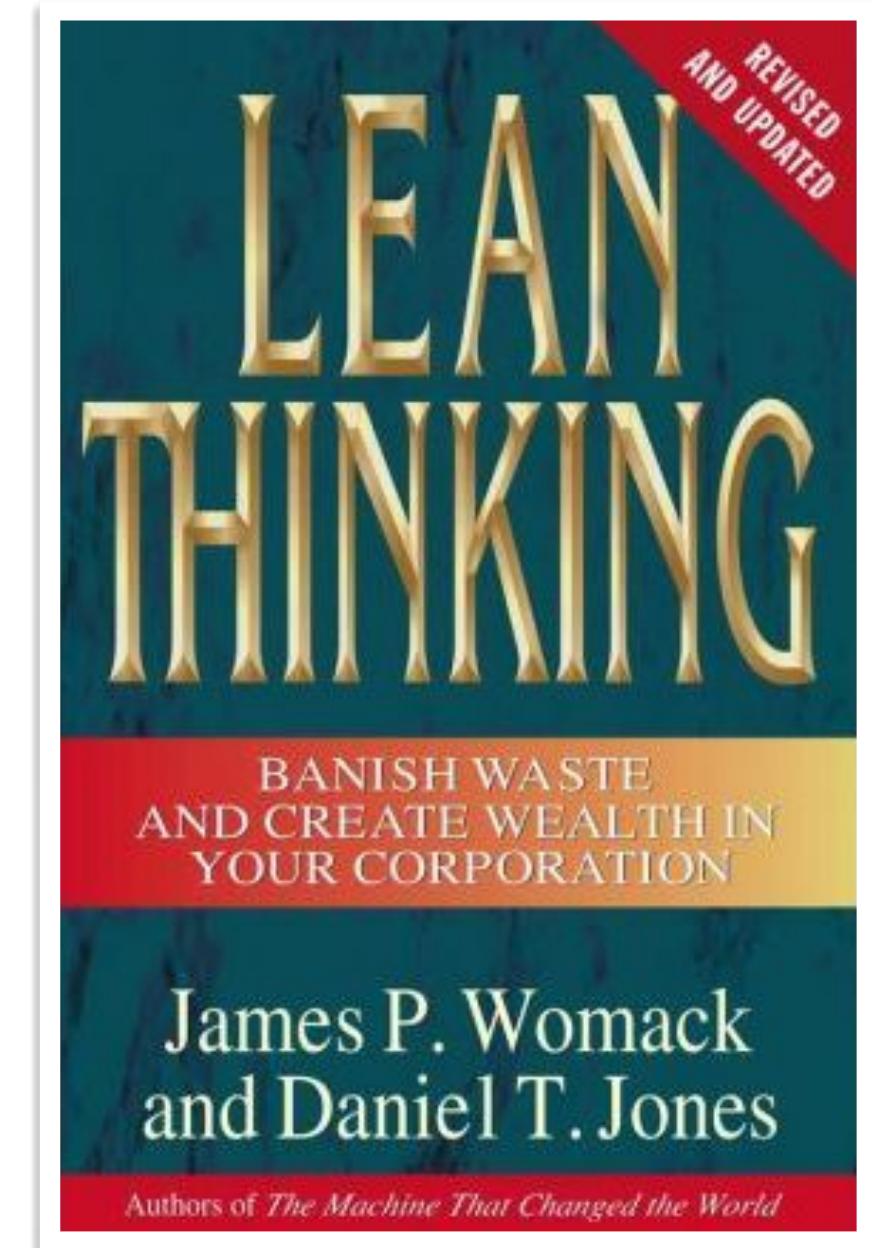
Popularización TPS [1990]



1988

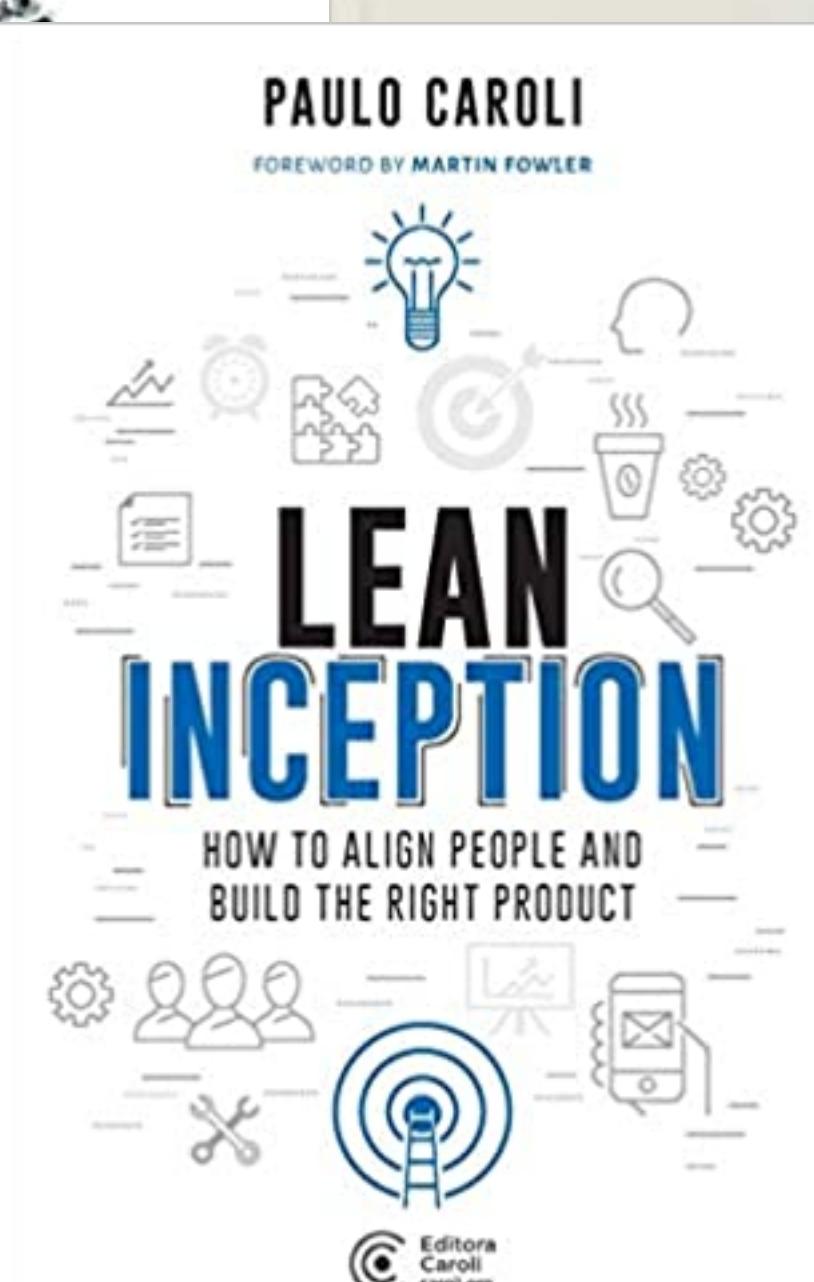
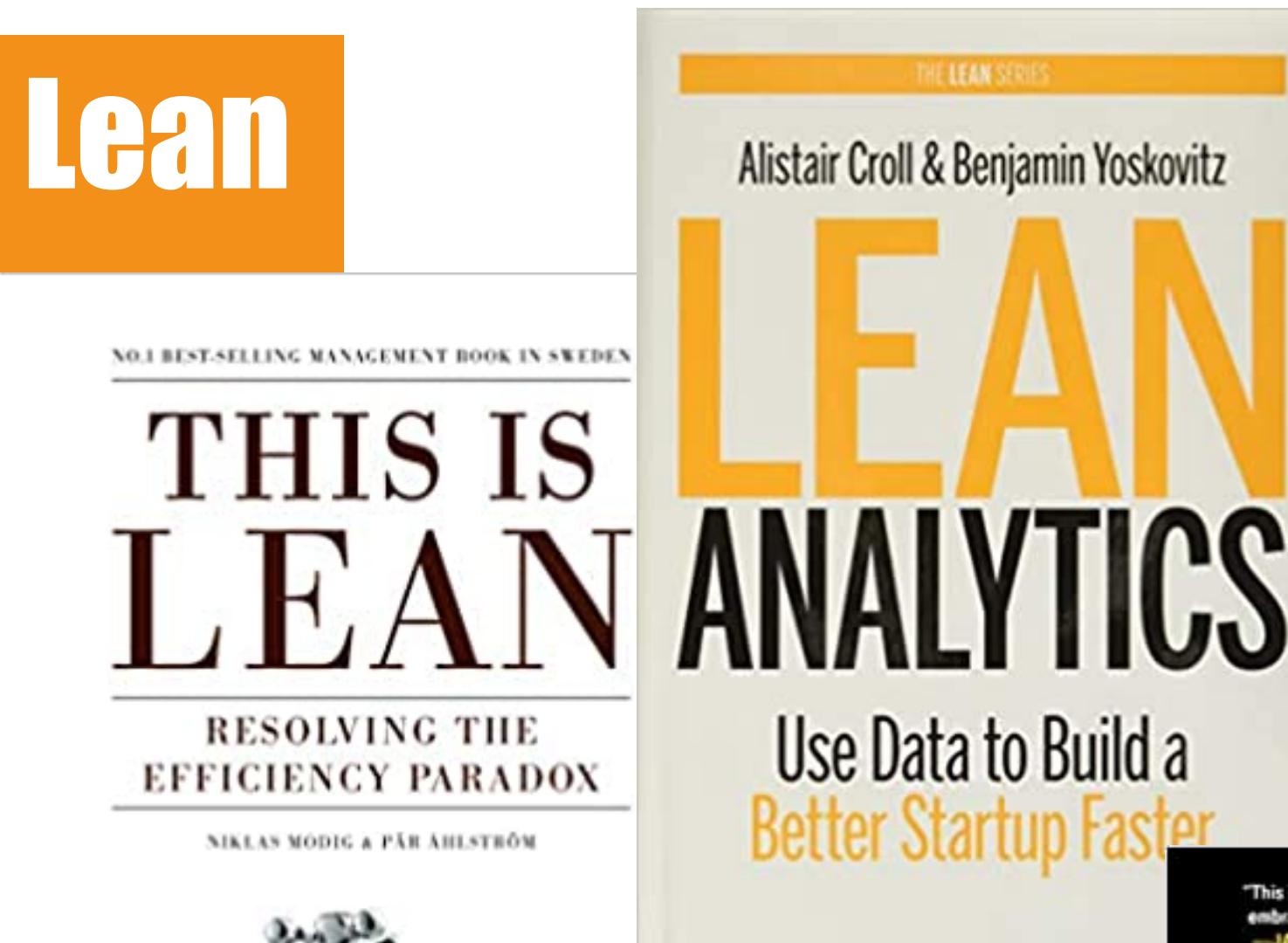
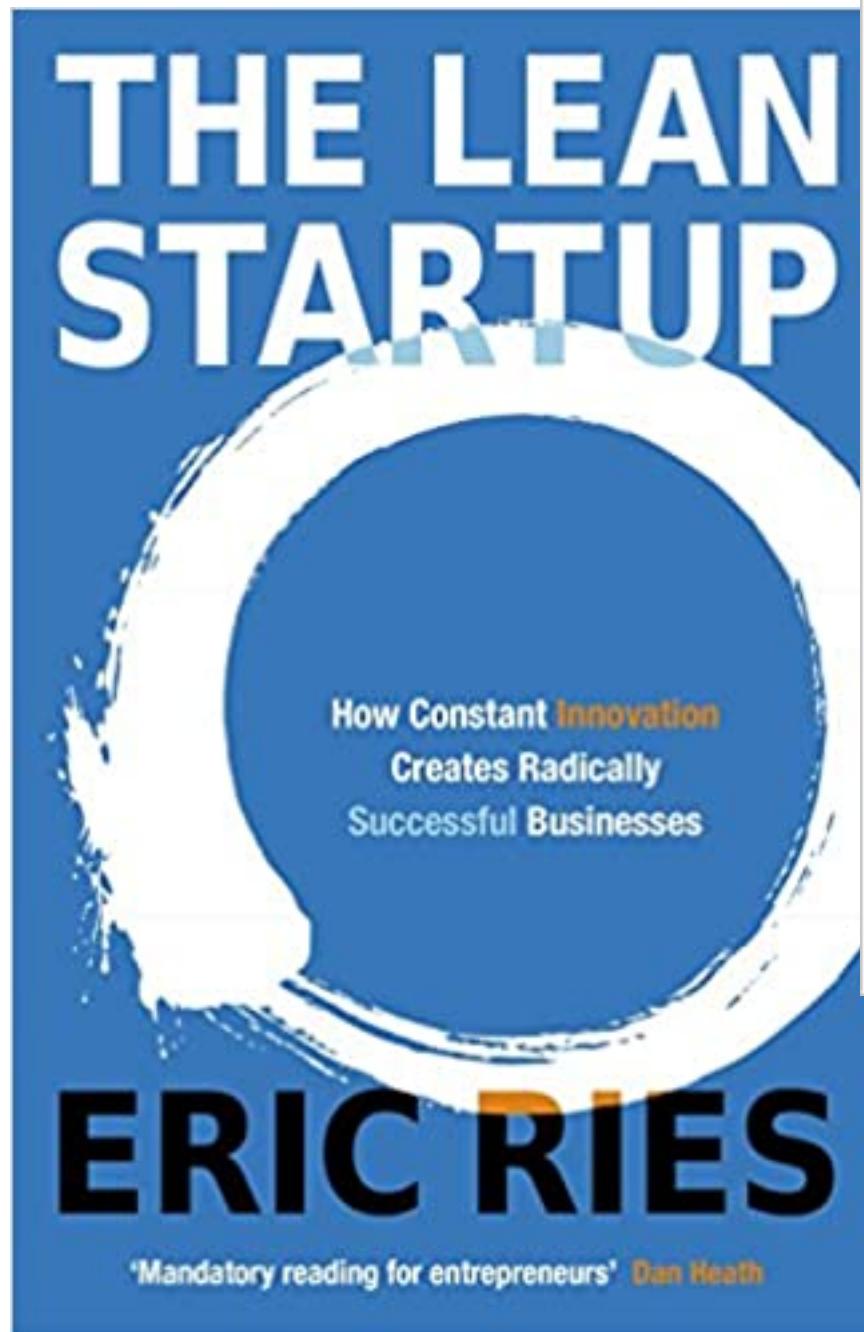


1991



1996

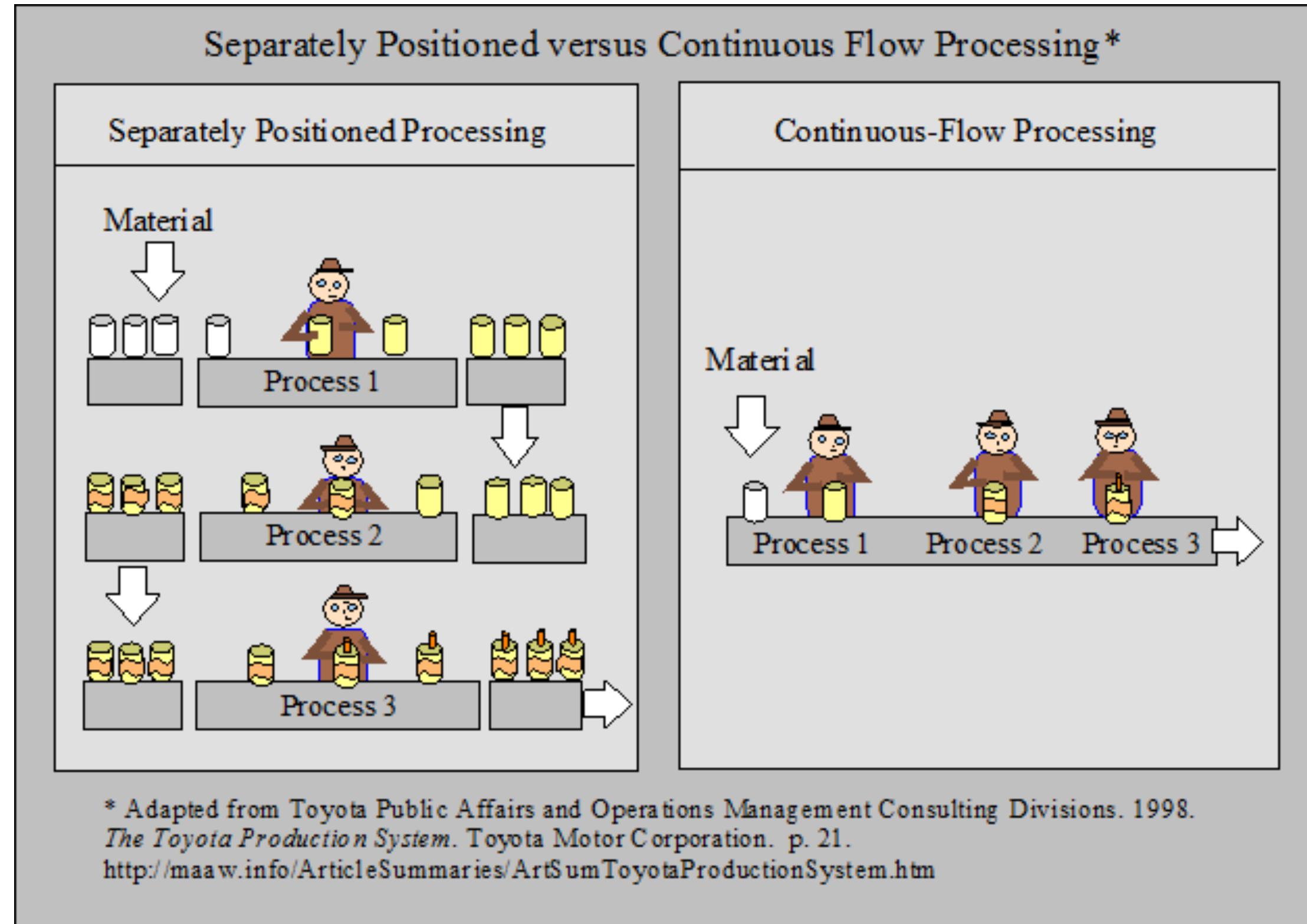
Popularización Lean (2010)



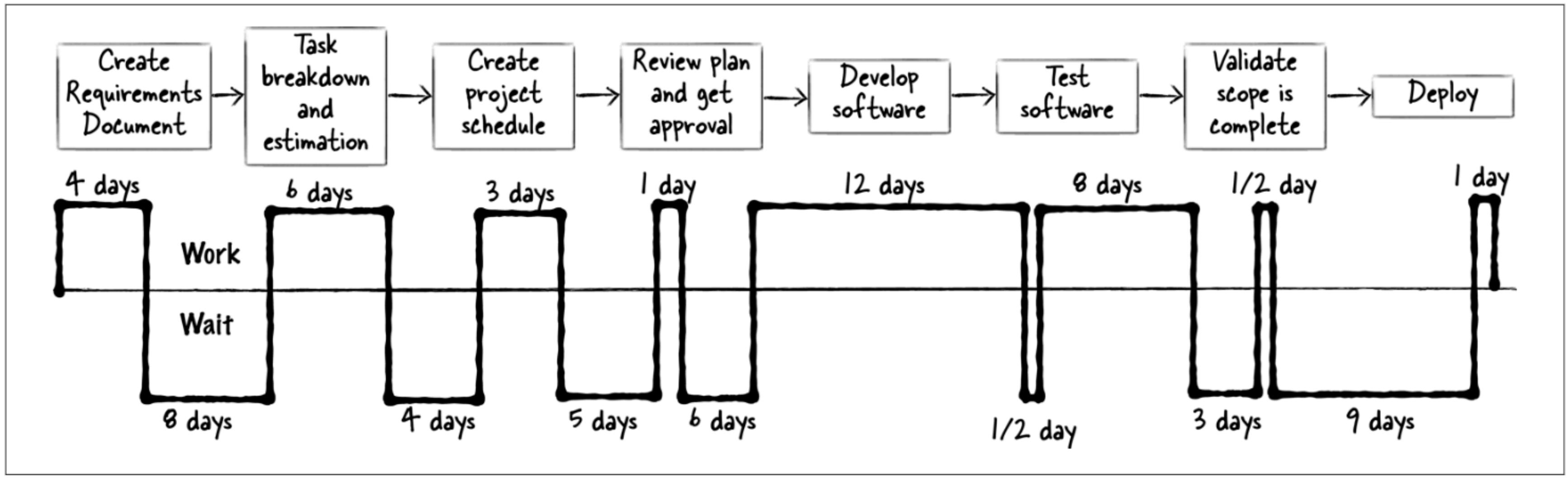
Algunas técnicas lean

Flujo de trabajo

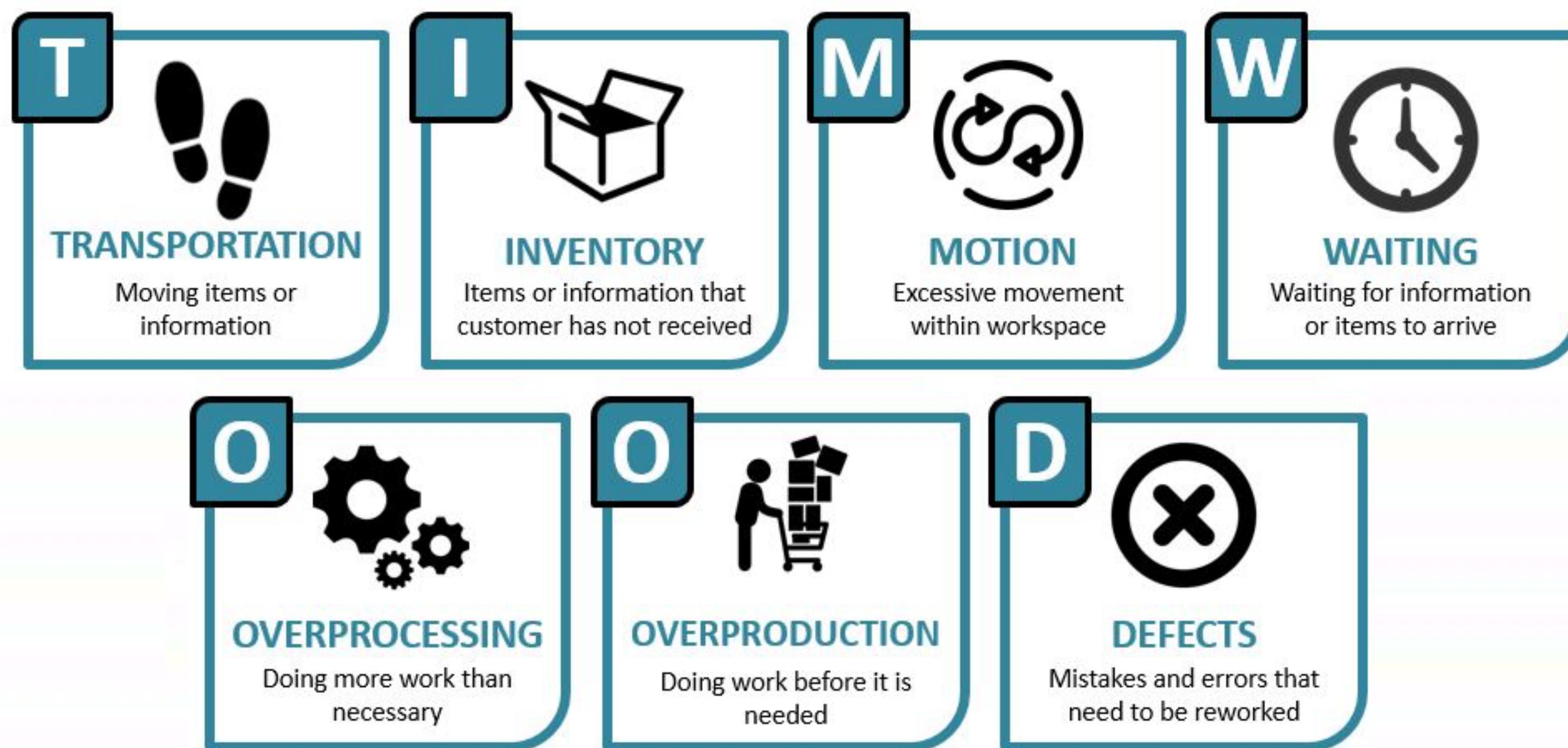
- Se construye un flujo de proceso y en cada paso se le añade valor al producto.
- Hay que optimizar el flujo: hacerlo lo más continuo, predecible y corto posible.



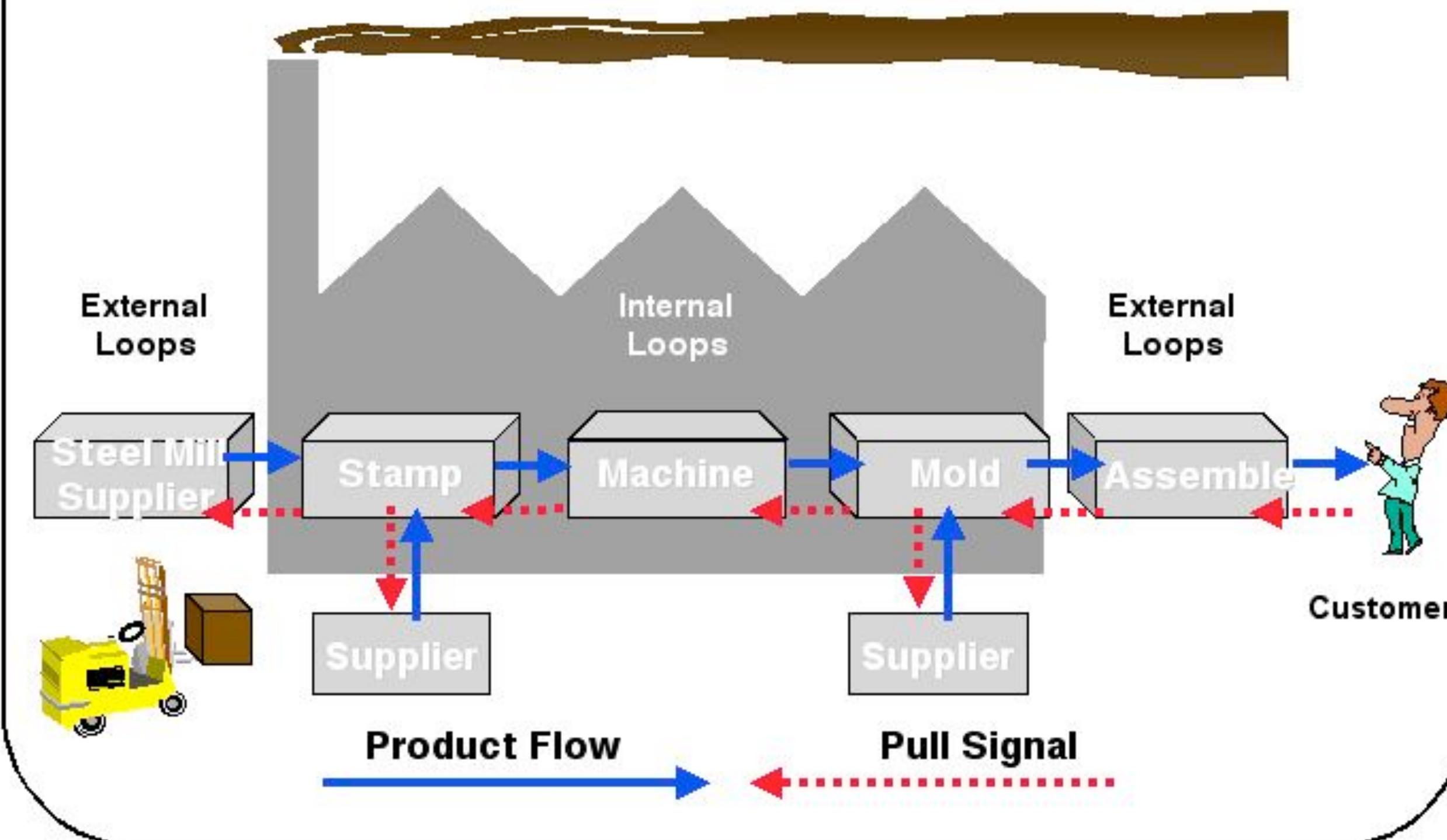
Value Stream Mapping



無駄 muda (waste, desperdicio)



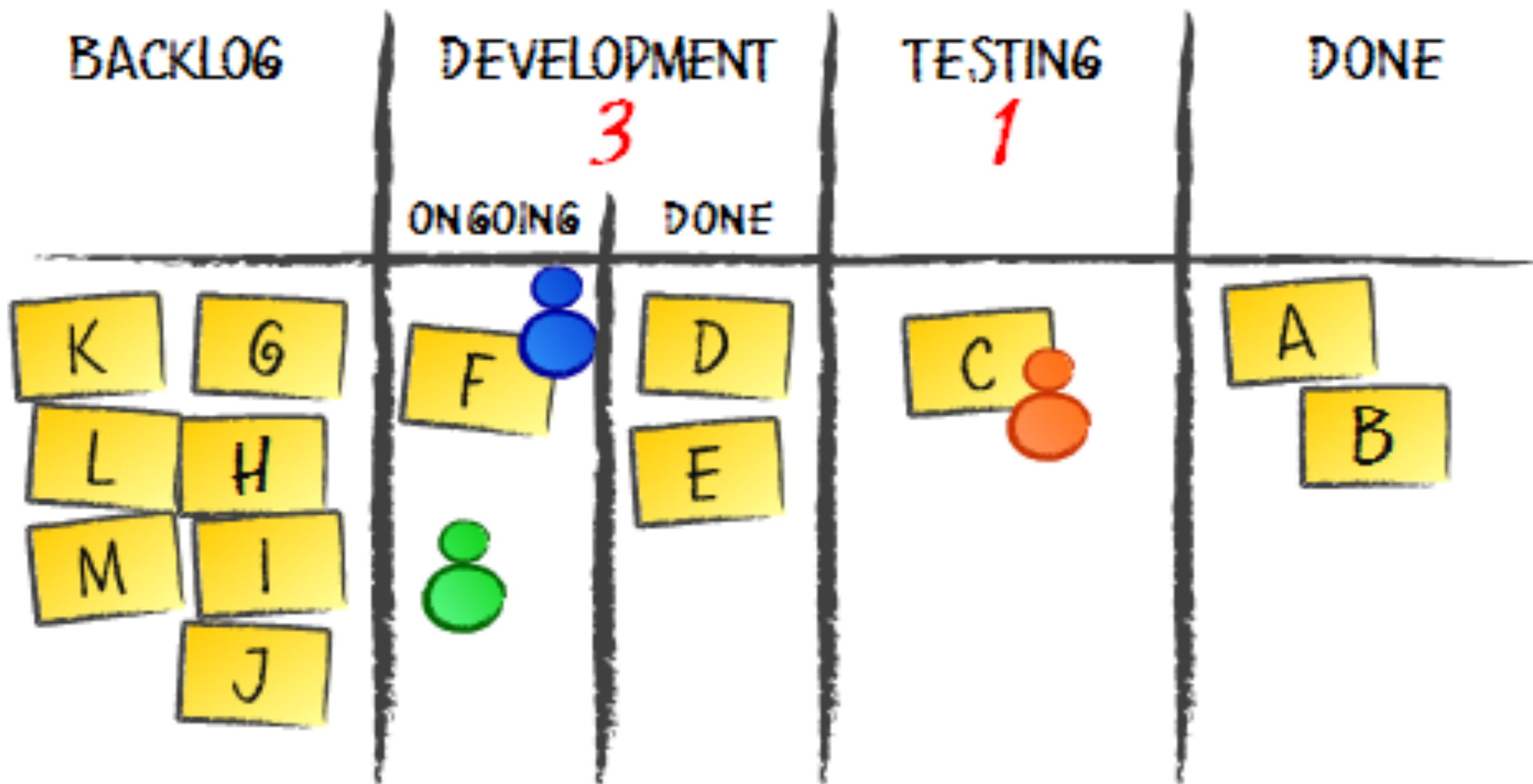
Pull System Example



Kanban

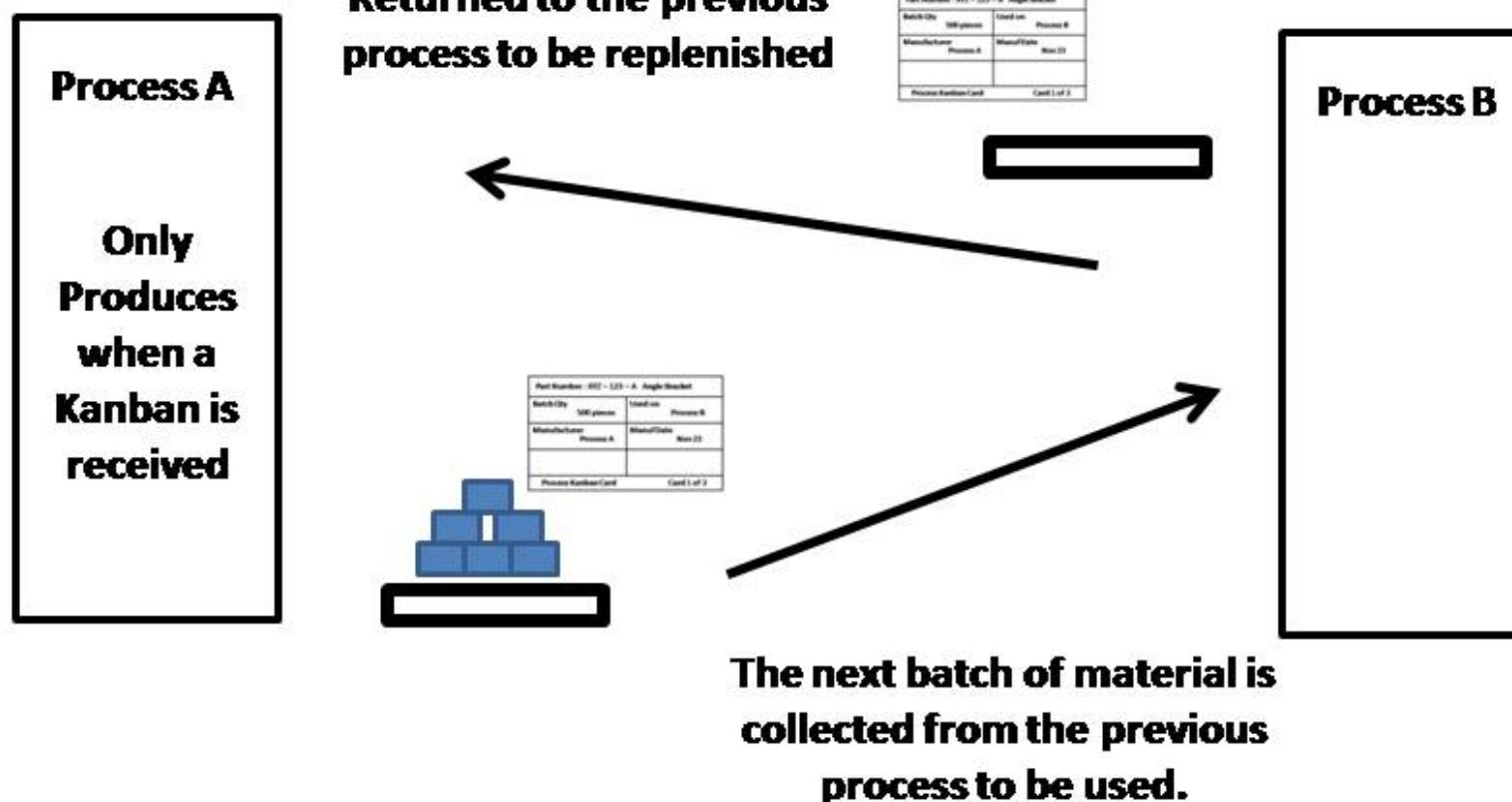


WIP Limit



看板 Kanban

Basic Kanban System



Tableros Kanban en desarrollo de software



Juego: Pasar las monedas



Henrik Kniberg

crisp

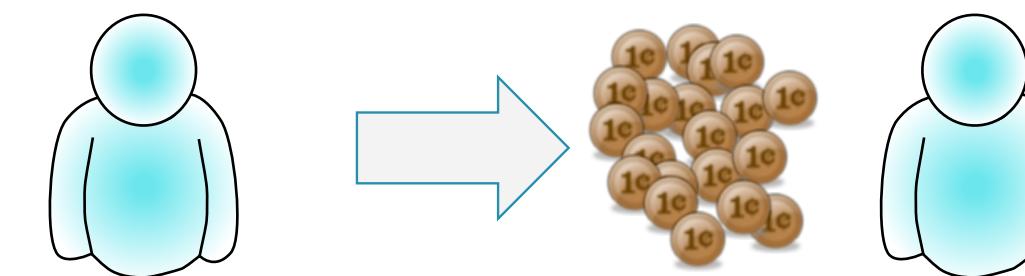
3 rondas

	Ronda 1	Ronda 2	Ronda 3
¿Cuánto tarda cada trabajador?			
Lisa			
David			
Martín			
María			
¿Cuándo recibe el cliente la primera moneda?			
¿Cuándo recibe el cliente la última moneda?			

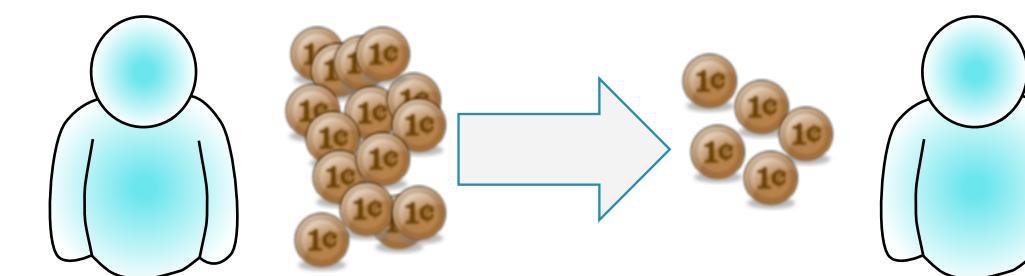
Cada encargado:

- Pone en marcha el cronómetro cuando su trabajador recibe la moneda #1
- Para el cronómetro cuando el trabajador envía su moneda #20

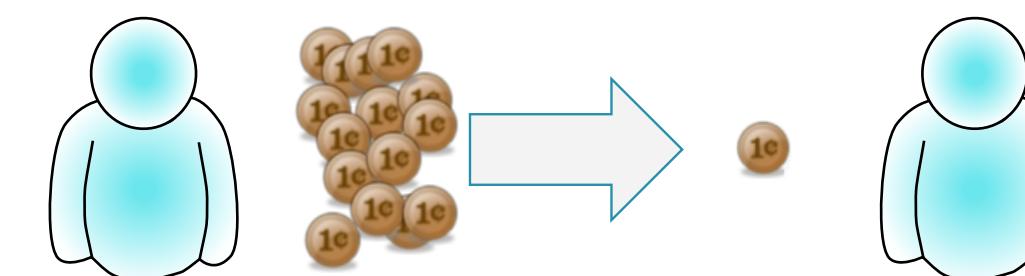
Ronda 1: tamaño del batch 20



Ronda 2: tamaño del batch 5



Ronda 3: tamaño del batch 1



Resultado típico

20 5 1

Worker 1	20.6	25.5	21.2
Worker 2	10.5	23.5	21.7
Worker 3	13.4	26.4	24.6
Worker 4	25	24.7	32.6
First	1.18.0	18.9	4.8
Total	1.18.0	40	37

Algunas preguntas

- ¿Qué ha pasado con el tiempo total? ¿Por qué?
- ¿Qué ha pasado con el tiempo de cada trabajador individual? ¿Por qué?
- ¿Cómo se sentían en cada ronda? ¿Ha sido estresante? ¿Cuándo ha sido más calmado?
- ¿Se puede trasladar esto al desarrollo de software?
- ¿Qué representan las monedas?
- ¿Qué no es aplicable en el desarrollo de software?

El tweet de la semana

The image is a screenshot of a tweet from Sam Altman (@sama). The tweet was posted on November 30. It features a profile picture of Sam Altman and the text: "today we launched ChatGPT. try talking with it here: chat.openai.com". Below the tweet are engagement metrics: 1,006 replies, 6,175 retweets, and 24.3K likes. There is also a share icon.

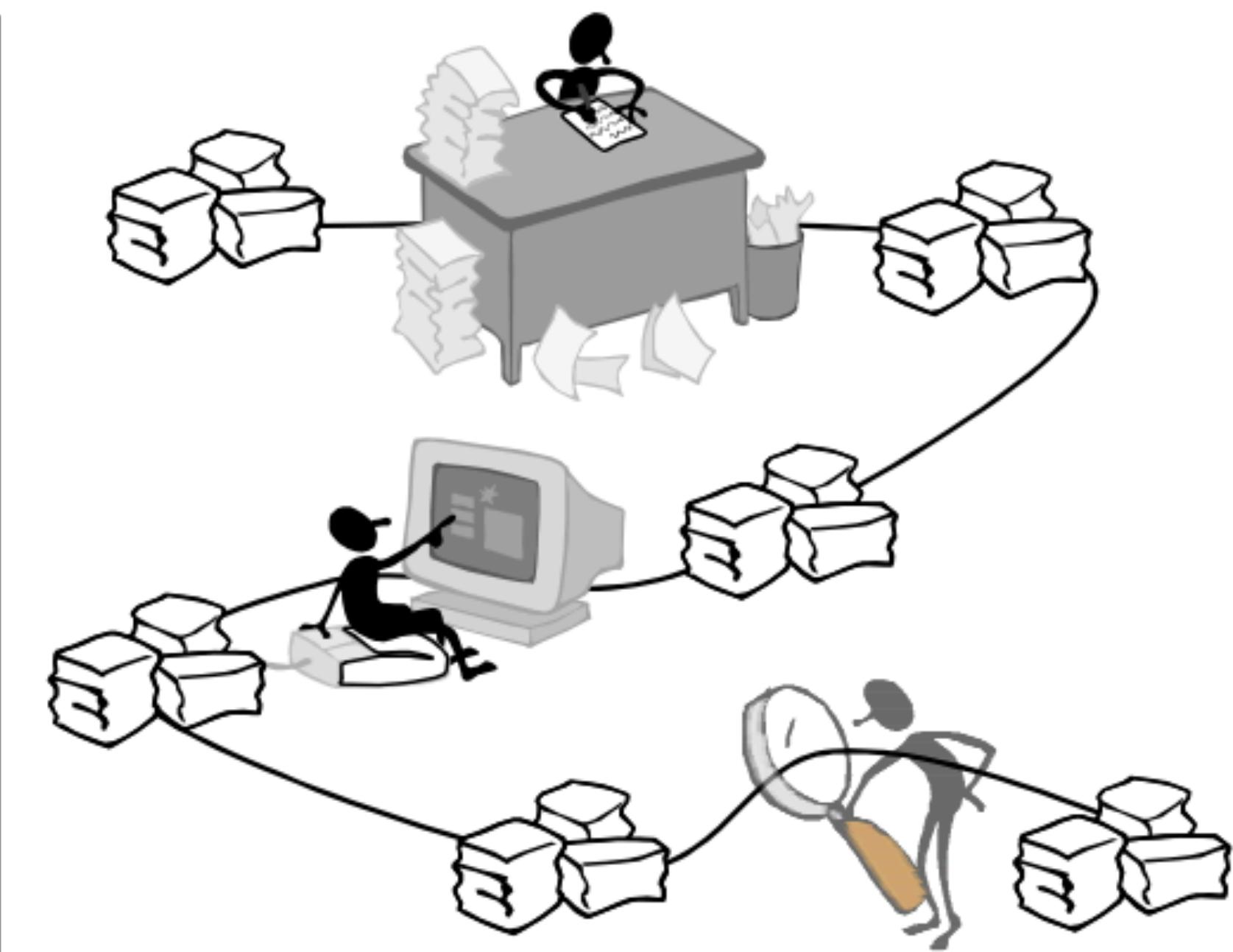
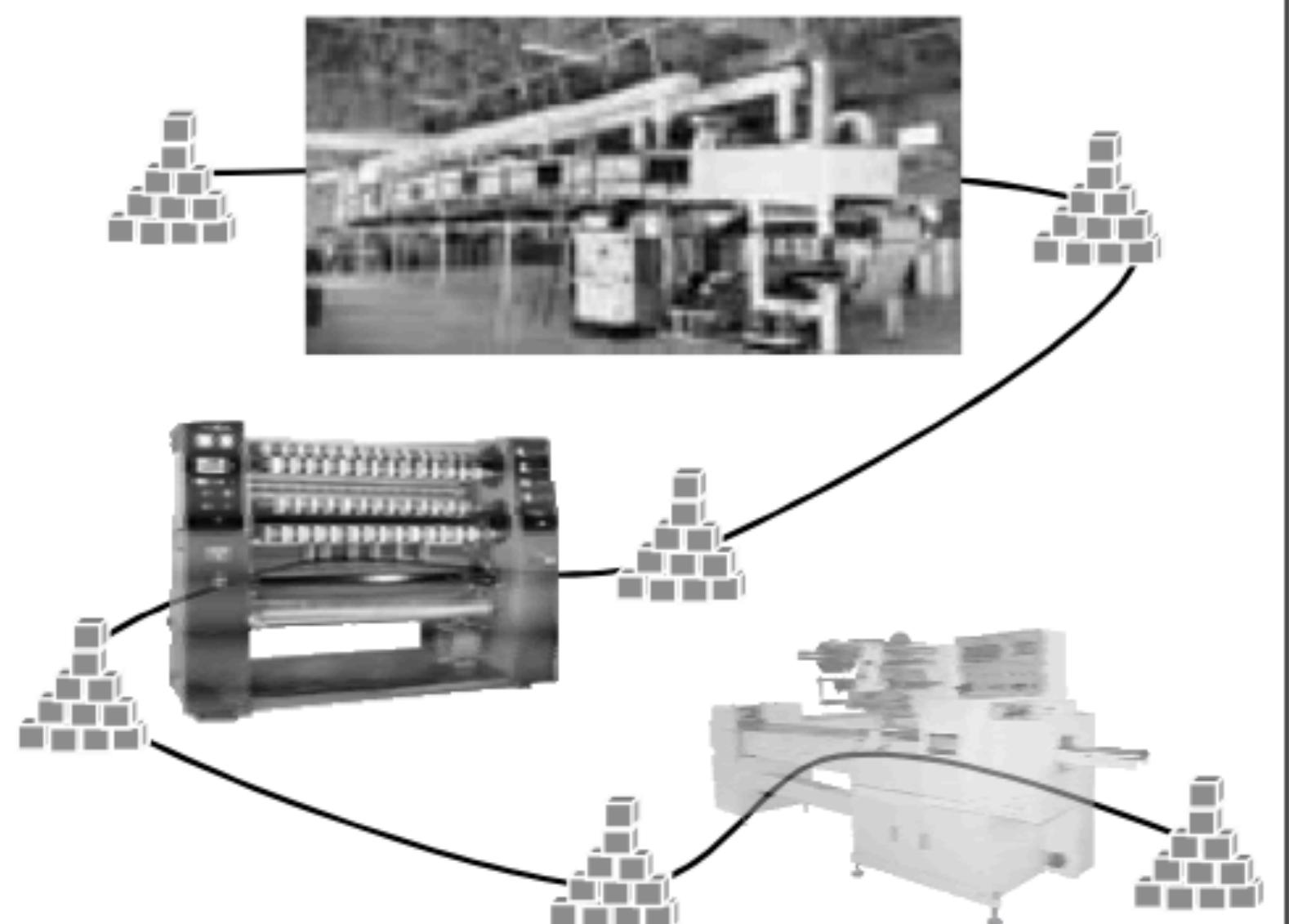
Sam Altman ✅ @sama · Nov 30

today we launched ChatGPT. try talking with it here:

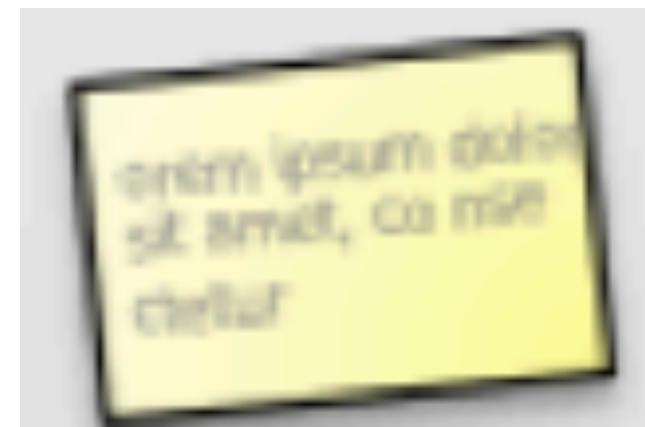
chat.openai.com

1,006 6,175 24.3K

El desarrollo de software como un proceso de fabricación



- Proceso de toma de decisión (¿deberíamos implementar esta feature?)
- Proceso de diseño (especificaciones, pizarras, mockups, etc.)
- Proceso de implementación (escribir código)
- Proceso de prueba (encontrar bugs)
- Proceso de depuración (arreglar bugs)
- Proceso de despliegue (enviar el código a los clientes, ponerlo en el servidor web, etc.)



Historia de usuario



Inventario en software

The screenshot shows a blog post titled 'Software Inventory' by Joel Spolsky. The post is dated July 9, 2012, and is categorized under 'NEWS, PROGRAM MANAGER'. It features a black and white photo of Joel Spolsky holding a ping pong paddle. Below the photo, there is a bio: 'I'm Joel Spolsky, a software developer in New York City. [More about me.](#)'

- **Backlog de features:** el 90% de las *features* en el backlog no llega a implementarse. Podemos minimizarlo limitando el backlog a 1 o 2 meses. Una vez que esté lleno no se introducirán nuevos ítems si no se quita alguno.
- **Base de datos de bugs:** algunas empresas mantienen bases de datos con cientos de bugs que nunca llegan a corregirse. Podemos minimizarlo implementando un sistema de triaje que indique si un bug debe corregirse o marcarse como cerrado. No hay que preocuparse en equivocarse, los bugs importantes reaparecerán.
- **Features no desplegada:** características implementadas pero no puestas en producción por ser el proceso de despliegue muy lento. Para minimizar el número debemos mejorar el proceso de despliegue y utilizar entrega continua.

May 17, 2011

The Carrying-Cost of Code: Taking Lean Seriously

I've spent the past 8 years or so looking at ugly code. This isn't uncommon in software development but in my case, I've been looking at different ugly code developed by different teams every couple of weeks. One question that people often have is whether to refactor or rewrite. It's never a simple call. Usually,

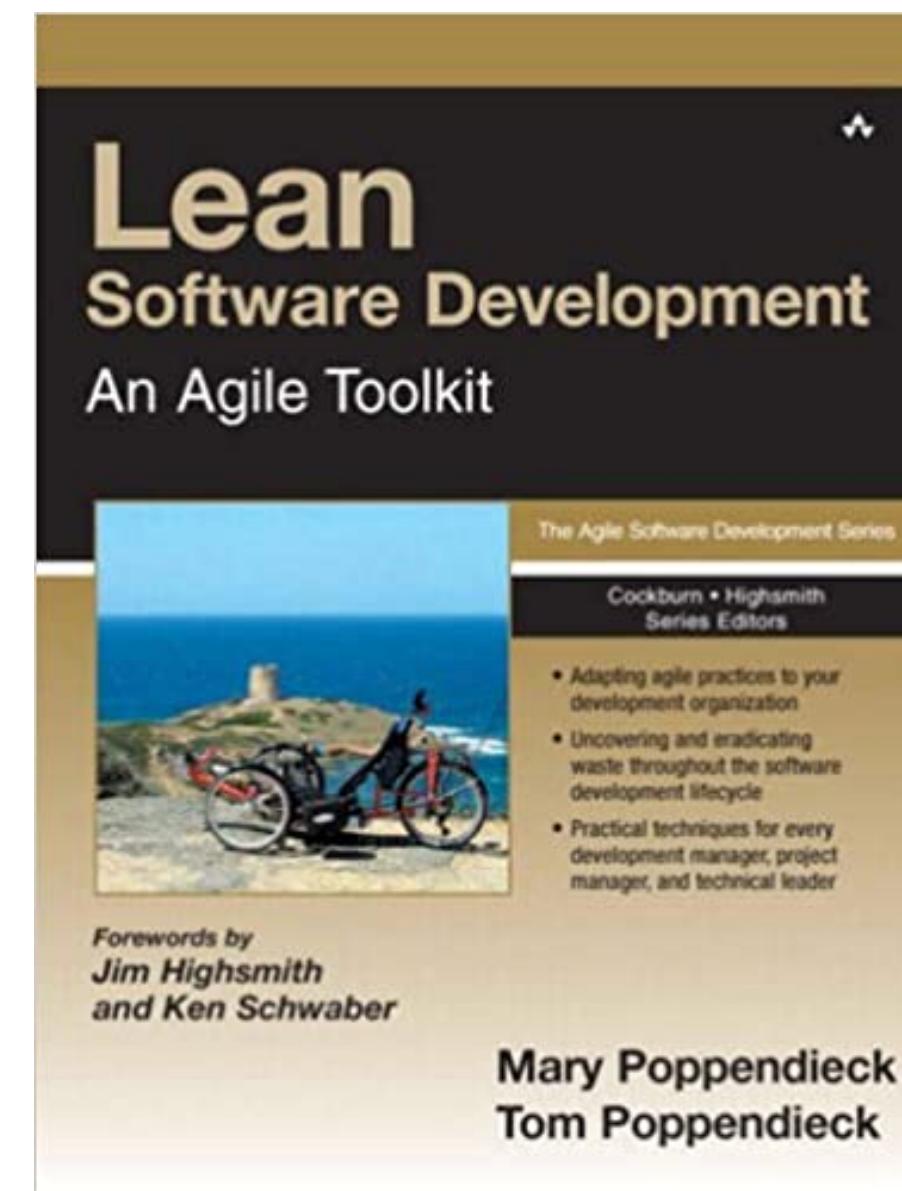
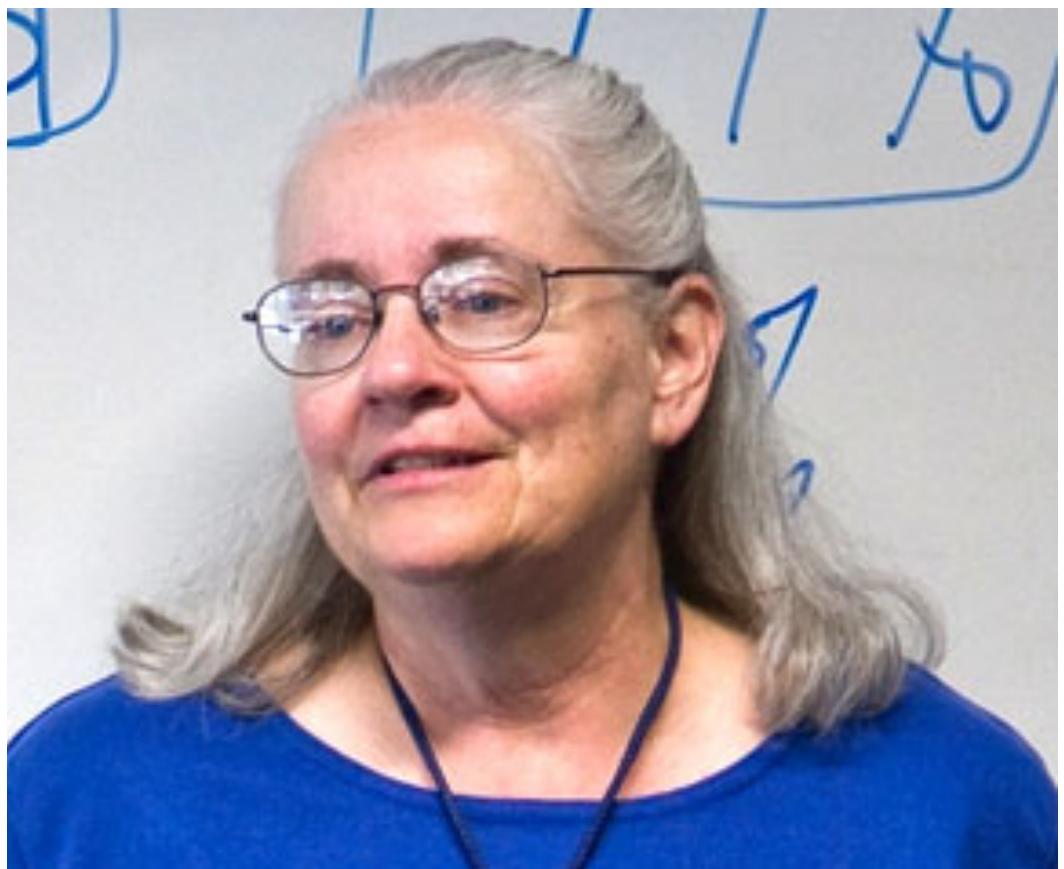
A few years ago, I was telling some friends about an experiment I would love to run. I'd like to have code base where every line of code written disappears exactly three months after it is written. If you were able to get past people gaming the system by copying code someplace else and then copying it back in when it was deleted, you'd have a very interesting set of constraints. Developers would be rewriting code constantly and they'd develop insights into ways to rewrite the code more compactly and (hopefully) more understandably. Perhaps more importantly, the business would have to make very serious tradeoffs about the features. If you are limited to a smaller number of features (because code keeps disappearing), you have to make sure that the ones you keep are really the ones which are making you money. I have the suspicion that a company could actually do better over the long term doing that, and the reason is because the costs of carrying code are real, but no one accounts for them.

to me, **code is inventory**. It is stuff lying around and it has substantial cost of ownership. It might do us good to consider what we can do to minimize it.

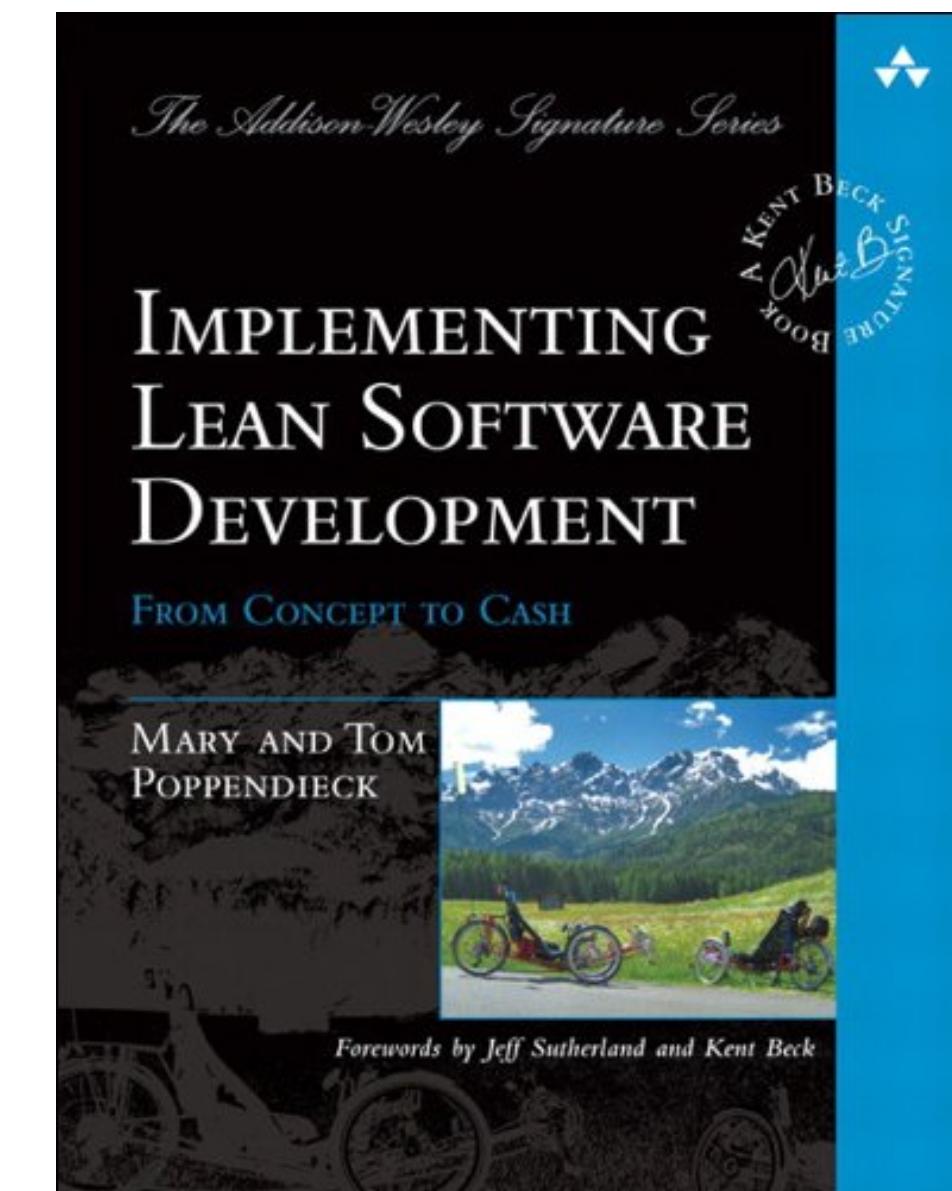
I think that the future belongs to organizations that learn how to strategically **delete** code. Many companies are getting better at cutting unprofitable features in their products, but the next step is to pull those features out by the root: the code. Carrying costs are larger than we think. There's competitive advantage for companies that recognize this.

Lean Software Development

Mary Poppendieck



2003



2006

Principios de lean software development

1. **Eliminar los desperdicios** (Eliminate Waste): ¿qué parte del trabajo no está relacionado directamente con crear valor para el usuario final? Debes encontrar los desperdicios y eliminarlos.
2. **Fomentar la calidad** (Build Quality In): debes fomentar tanto la calidad interna del software como la calidad final apreciada por el usuario.
3. **Crear conocimiento** (Create Knowledge): debes utilizar el feedback obtenido por las entregas para mejorar tu software.
4. **Decidir lo más tarde posible** (Defer Commitment): debes tomar las decisiones importantes para el proyecto cuando tengas la mayor cantidad posible de información, en el último momento responsable.
5. **Entregar rápido** (Deliver Fast): debes comprender el coste de los retrasos, usando un proceso pull.
6. **Respetar a la gente**, potenciar el equipo (Respect People, empower the team): debes crear un entorno de trabajo centrado y efectivo y construir un equipo de gente implicada.
7. **Optimizar el conjunto** (Optimize the Whole): debes entender el proceso de desarrollo y tomar todas las mediciones posibles para tener una idea clara de sus elementos y dinámicas.