Desarrollo lean

Índice

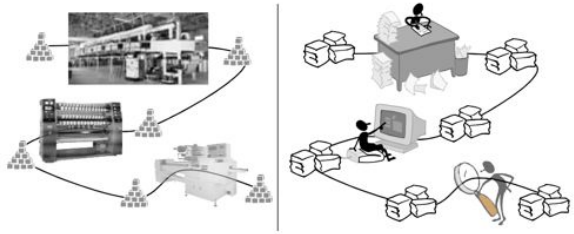
- Historia de la fabricación *lean*
- Prácticas de fabricación *lean*
  - Desperdicios
  - Sistemas pull
  - Kanban
- Pricipios de Mary Poppendieck del desarrollo lean de software

El software es un producto distinto

- Ya lo vimos la semana pasada: el **software y su desarrollo es único** y tiene características propias que lo hace distinto de cualquier otro proceso. No es lo mismo fabricar software que fabricar bicicletas.
- El **proceso de desarrollo** es distinto (hacer una aplicación es distinto que hacer una bicicleta):
  - El desarrollo es el proceso de **transformar ideas en productos**. En el caso del software debe ser un proceso empírico, basado en la adaptación y la iteración. Es más un **proceso de diseño** que de fabricación.
  - La **variabilidad del software** desarrollado puede hacer complicado definir un proceso común para todos los tipos de software.
- El software como **producto desarrollado** (una aplicación en funcionamiento es distinto que una bicicleta en funcionamiento):
  - El software en funcionamiento crea un **sistema complejo** que articula y organiza la actividad de un gran número de elementos (personas, dispositivos, software de terceros, etc.). Por ejemplo: todo el software de gestión académica de una universidad (matriculación, cambios de grupo, recibos, actas, …).
  - Una parte importante de la detección de errores, bugs, fallos de diseño, etc. se realiza en el sistema en funcionamiento.

Pero… es posible aprender de los procesos de fabricación tradicionales

- Fabricación de un producto
  - Entradas: materias primas y componentes
  - Salida: producto terminado (automóvil, teléfono móvil, televisor, etc.)
  - Proceso: diferentes máquinas y pasos en la cadena de montaje
- Si vemos el desarrollo software como un proceso iterativo podemos definir un proceso de desarrollo general:
  - Entradas: software funcionando e ideas de nuevas **características** (***features***) en forma de casos de uso, historias de usuario, etc.
  - Salida: software funcionando al que se le ha añadido las nuevas características.
  - Proceso: cada característica debe ser analizada, desarrollada, probada, añadida y entregada.
  - 2 ejes de calidad: elegir qué característica (*right product*) y desarrollar correctamente la característica (*product right*).
- Cuando miramos el proceso de desarrollo de nuevas características como un proceso de fabricación, podemos aplicar al desarrollo de software muchas ideas aprendidas en los procesos de fabricación. Sobre todo las técnicas de **fabricación lean**.



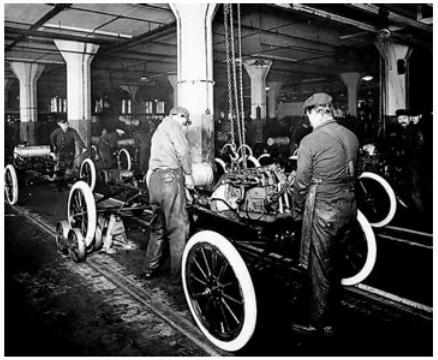
Mary Poppendieck - Lean Software Development, Tutorial 2007

Henrik Kniberg - Agile Lean Slides

1910: Cadena de producción

- Ejemplo típico de un proceso de fabricación: fabricación de automóviles
- En las fábricas de Henry Ford se desarrolla en EEUU en 1910 el proceso de fabricación en cadena que permite acometer una producción en gran escala
  - Descomponer la fabricación en pequeños pasos especializados
  - El producto que se va construyendo se va moviendo de un proceso a otro (cadena que fluye)
  - Trabajadores fáciles de formar y de reemplazar: se especializan en hacer poco y hacerlo bien
- Problemas:
  - Falta de motivación de los trabajadores
  - Sistema muy útil para la fabricación en escala pero muy rígido: se tarda mucho en responder a los cambios que piden los consumidores

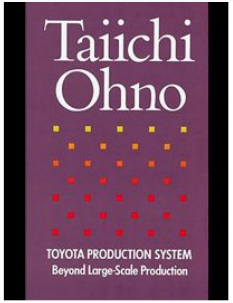
Henry Ford

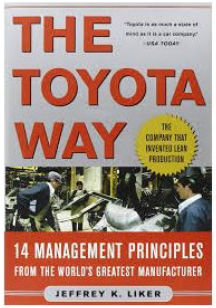


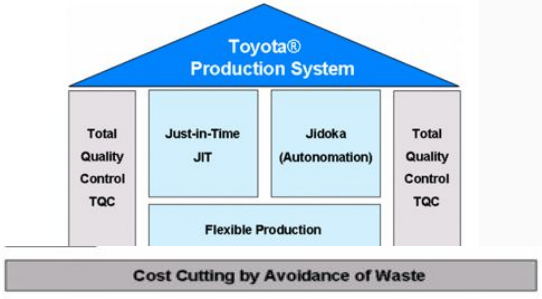
1950: Sistema de producción de Toyota

- Después de la Segunda Guerra Mundial, en el marco de un Japón empobrecido y sin apenas recursos, en la fabrica de automóviles Toyota **Taiichi Ohno** empezó a desarrollar una filosofía de gestión y producción que a la postre terminaría aupando a Toyota (y al resto de fabricantes japoneses) a dominar la producción de automóviles, superando a los gigantes americanos Ford y General Motors.
- TPS (Toyota Production System), también denominado “producción just-in-time”:
  - **Eliminar los desperdicios** (*waste*).
  - Flujo de producción ***just-in-time***, evitando gastos de inventario. Ciclos de producción cortos.
  - Cultura de “**parar la cadena**” en el momento en que se detecta el mínimo error.
  - Cultura de **mejora continua** en todos los niveles: desde los trabajadores de la cadena hasta los directivos.
  - Pensar en el **conjunto**. Equipos multi-funcionales.
  - Herramientas para **visualizar** el proceso.
  - La esencia de TPS es una **mentalidad** (*mindset*) o cultura de empresa basada en la mejora continua y el desarrollo de las capacidades de los empleados y los colaboradores.

Taiichi Ohno



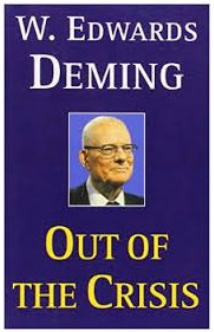




1950: Mejora continua y calidad

- **Edwards Deming**: ingeniero electrónico y físico americano, asesoró a los líderes de las industrias japonesas después de la segunda guerra mundial.
- Entre junio y agosto de 1950 formó a cientos de ingenieros japoneses, impartiendo seminarios sobre lo que denominó Administración Estadística de la Calidad del Producto.
- Uno de los padres del milagro económico japonés en los años 50-60, cuando Japón se transformó en la segunda potencia económica mundial.
- Algunas ideas principales y citas:
  - Buscar siempre la **mejora continua de la calidad** como el medio de reducir gastos y de aumentar la productividad y la cuota de mercado. “Mejora la calidad, y automáticamente mejorarás la productividad.”
  - “No es necesario cambiar. La supervivencia no es obligatoria.” “El aprendizaje no es obligatorio… tampoco lo es la supervivencia.”
  - “No es suficiente dar lo mejor de ti; debes saber qué hacer y después dar lo mejor de ti.” “Un gran esfuerzo nunca sustituirá al conocimiento.”
  - “Si no puedes describir lo que estás haciendo como un proceso, no sabes lo que estás haciendo.”
  - “¿La experiencia ayuda? ¡No! No, si estás haciendo lo equivocado.”
  - “Deberíamos trabajar en nuestros procesos, no en los resultados de nuestros procesos.”

Edwards Deming



Sistemas de fabricación lean

- Estas ideas dan origen a los denominados sistemas de fabricación lean (lean = austero, flaco)
  - El **proceso** siempre puede ser mejorado y los trabajadores son los que mejor pueden proponer estas mejores.
  - **Método científico**: los trabajadores aprenden a crear hipótesis, probarlas, analizar los resultados y, si los datos confirman la hipótesis, hacer el cambio permanente.
  - Identificar los distintos pasos del proceso de producción (el ***value stream***) cadena de valor del proceso.
  - Una idea central es la continua búsqueda y eliminación de los **desperdicios** (*waste*) generados por el proceso. Simplificar.
  - Cuando se eliminan los desperdicios la calidad mejora, y el tiempo de producción y los costes se reducen y la producción se vuelve **fluida** (*flow*).
  - Consecuencias: alta disciplina y alta respuesta al cambio.
  - Es una **mentalidad** (*mindset*), no un conjunto prescrito de reglas.
- El término lean se ha popularizado: lean startups, lean thinking

Buscar y eliminar desperdicios (*waste*)

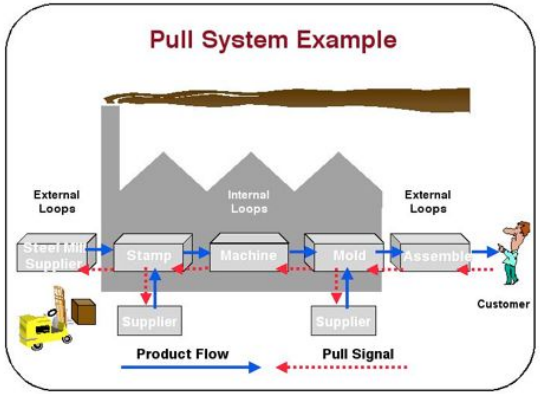
- Todo aquellos elementos que no añaden valor al producto. Si minimizamos los desperdicios maximizaremos la cantidad de trabajo útil, que realmente da valor.
- Ejemplos de desperdicios en procesos de fabricación y servicios:
  - **Espera**: personas o hitos del proceso esperando que termine otro proceso o que llegue cierta información
  - **Movimiento**: movimiento físico o mental que no añade valor
  - **Inventario**: almacenar servicios y componentes extra que el cliente no ha pedido
  - **Defectos**: errores que hay que corregir
  - **Sobre-procesamiento**: excesiva documentación, informes excesivos, partes no necesarias

Meaning of 5S

	Principles	General Description
1S	**Sort**	Remove what is not needed and keep what is needed
2S	**Set in Order**	Arrange essential items in order for easy access
3S	**Shine**	Keep things clean and tidy; no trash or dirt in the workplace
4S	**Standardize**	Establish standards and guidelines to maintain an organized workplace
5S	**Sustain**	Make 5S a habit and teach others to adhere to established standards

Sistemas de fabricación *Pull*

- Uno de los pilares de la fabricación *lean*
- El proceso de fabricación se divide en un conjunto de pasos (*celdas*) que necesitan recursos y consumen los resultados de procesos anteriores (*upstream*)
- Sistema **push**: se planifica a priori la cantidad de trabajo a comenzar. Cuando un proceso upstream produce un componente se empuja (push) a la siguiente celda para que continue procesándolo. Muchas veces se provoca sobrecarga en las celdas.

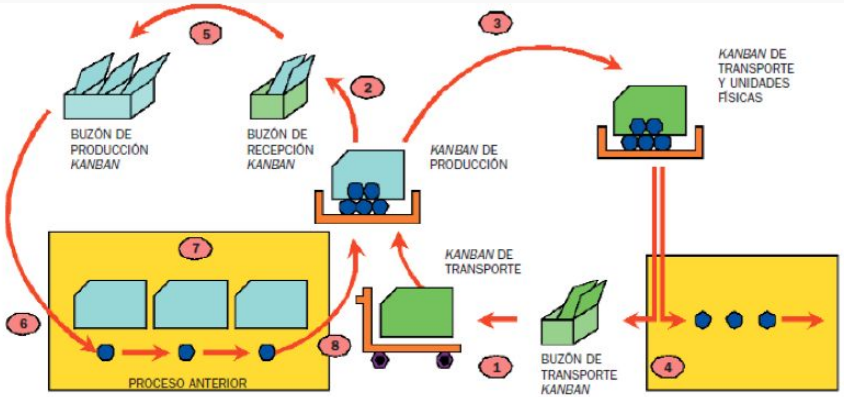


- Sistema **pull**: el origen del flujo de trabajo está al final de la cadena, al entregar el producto final a los clientes. Cada celda tiene un número máximo de productos (**WIP**), cuando se consume uno se envía una señal (pull signal) a la celda anterior de que se necesita recibir un nuevo componente.
- El flujo de trabajo se regula tirando (***pull***) de los materiales a transformar con una cadencia constante
- Un sistema pull regula el flujo de los recursos mediante un proceso de fabricación reemplazando solo lo que ha sido consumido y lo que es inmediatamente entregable

Sistema *pull*

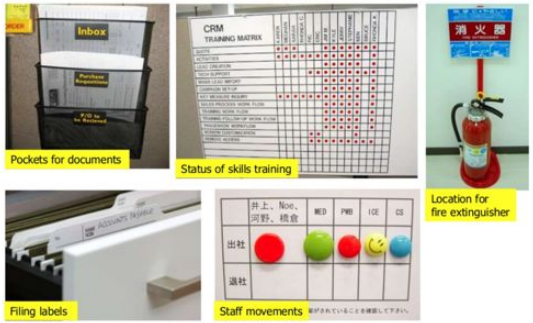
Kanban

- Una de las herramientas más importantes para organizar el proceso de producción son las **kanban** (del japonés, kan=visual y ban = tablero o tarjeta), **señales visuales** que implementan el sistema pull.
- **Enfoque visual** para el control de la producción, usando herramientas sencillas como contenedores retornables, tarjetas o incluso espacios vacíos para “tirar” de los productos desde los centros de producción hacia los centros de consumo o transformación.
- Una kanban es una señal o ayuda visual que indica que un centro de trabajo ha finalizado un proceso, necesita trabajo o necesita más materiales.
- Los tableros kanban permiten que los centros de trabajo hagan un seguimiento de las necesidades de los clientes o de los proveedores y que respondan rápida y adecuadamente.

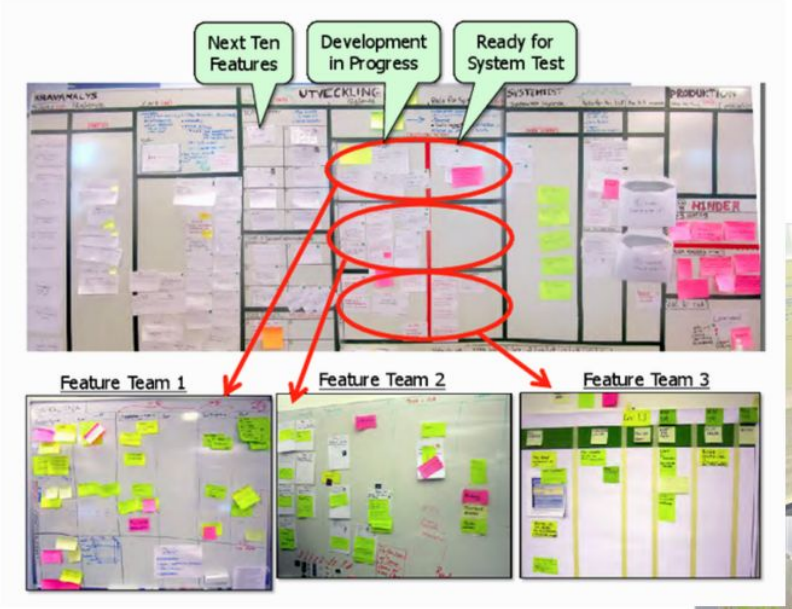


Toyota Just-in-Time Kanban System

Examples of Visual Display



Tableros *kanban* en el desarrollo de software

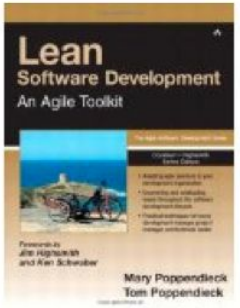


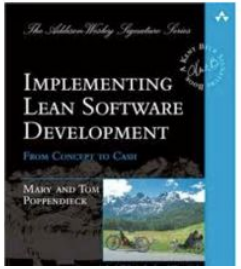
Demo: flow mediante descomposición de tareas

Mary Poppendieck

Mary Poppendieck

- Mary Poppendieck fue una de las primeras personas en proponer aplicar la filosofía de la **fabricación lean** al desarrollo del software
- Conoció los sistemas de fabricación lean a mitad de los años 80, cuando implantó un sistema lean en una fábrica de cintas de vídeo para competir con la producción japonesa
  - Formación a los trabajadores en sistemas pull, producción just-in-time
  - Desaparición del inventario
  - Bajada del coste de producción
  - Respuesta al cliente en 1 semana en lugar de 1 mes
- Publicó uno de los primeros libros sobre cómo aplicar estas ideas al desarrollo de software: Lean Software Development: An Agile Toolkit (2003)
- Activa divulgadora de las metodologías ágiles y del enfoque lean, participa activamente en las más importantes conferencias ágiles en todo el mundo





Los 7 principios del desarrollo lean de software

- Definidos por Mary Poppendieck ("Lean Software Development: An Agile Toolkit", 2003 y “Implementing Lean Software Development”, 2006)

1. Eliminar los desperdicios (*Eliminate Waste*)
2. Fomentar la calidad (*Build Quality In*)
3. Crear conocimiento (*Create Knowledge*)
4. Decidir lo más tarde posible (*Defer Commitment*)
5. Entregar rápido (*Deliver Fast*)
6. Respetar a la gente, potenciar el equipo (*Respect People, empower the team)*
7. Optimizar el conjunto (*Optimize the Whole*)

1. Eliminar los desperdicios (*waste*)

- Desperdicio: cualquier cosa que no se utiliza y que no añade **valor** al producto final:
  - Añadir valor: cualquier actividad que incrementa la funcionalidad el producto o servicio
  - Algo por lo que el cliente está dispuesto a pagar
- Producir sólo lo necesario y en el momento en el que se necesita
  - Si algo no se utiliza, no podemos darnos cuenta de sus fallos
- Desperdicios en el proceso de desarrollo del software (*thing right*):
  - Demasiados requisitos para un *release*
  - Demasiada arquitectura para las necesidades actuales
  - Demasiado código para poder ser probado adecuadamente
- Desperdicios en el proceso de creación de producto (*right thing*):
  - Demasiadas funcionalidades para el usuario final
  - Tener que re-aprender por falta de comunicación: desperdicio de tiempo y energía. Cuando alguien en el equipo aprende algo nuevo sobre el producto, ese conocimiento se debe extender a todo el equipo rápidamente.

Paper “Software Development Waste”

- Publicado en 2017, en la International Conference on Software Engineering (ICSE 2017)
  - Estudio dirigido por Todd Sedano (Pivotal) durante 2 años, en los que se entrevistó a 8 equipos formados en total por 33 ingenieros, diseñadores de interacción y gestores de producto.
  - Todos los equipos usan metodologías ágiles, principalmente XP
- Identifican 9 tipos de waste:

1. Building the wrong feature or product
2. Mismanaging the backlog
3. Rework
4. Unnecessarily complex solutions
5. Extraneous cognitive load
6. Psychological distress
7. Waiting/multitasking
8. Knowledge loss
9. Ineffective communication

2017 IEEE/ACM 39th International Conference on Software Engineering

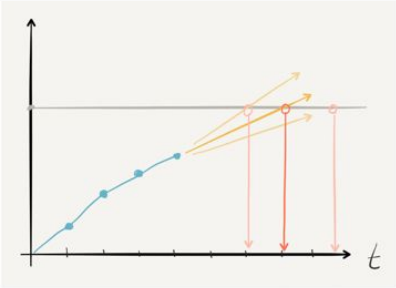
Software Development Waste

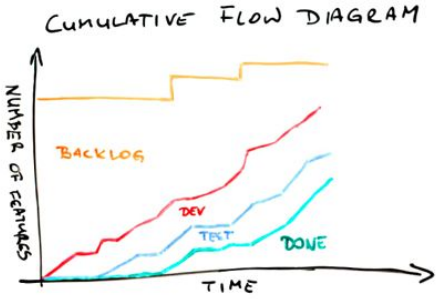
2. Fomentar la calidad

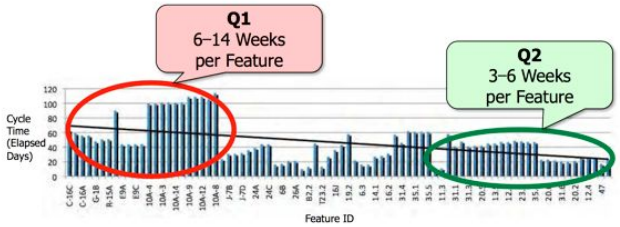
- Construye software que intuitivamente tenga sentido para los usuarios, y que forme un todo coherente.
- Muchas veces los clientes y los usuarios son distintos. El software debe satisfacer a los clientes porque satisface a sus usuarios.
- Minimiza la **deuda técnica** (***technical debt***)
  - Deuda técnica del software: problemas de diseño o implementación que terminarán apareciendo en forma de bugs o dificultades de ampliación y mantenimiento. “Parches” que se añaden al software para salir del paso rápidamente, en lugar de pensar en una solución más general.
  - Refactoriza el software siempre que tengas oportunidad.
  - Proceso de **revisión de código** (***code review***)
- Software sin defectos: énfasis en pruebas.
- Software que funciona:
  - Los clientes usan las **características** (***features***) del software
  - Software consistente con **integridad conceptual** (***conceptual integrity***): las características deben trabajar juntas para formar un producto único, integrado y fácil de usar.

3. Crear conocimiento

- Dos tipos de conocimiento:
  - Conocimiento sobre el producto que estamos desarrollando
  - Conocimiento sobre el proceso y la forma en la que desarrollamos productos
- Utiliza el feedback proporcionado por los proyectos para mejorar cómo construyes el software.
  - Representación gráfica de la evolución del desarrollo: las mediciones y los diagramas nos pueden dar pistas de posibles cuellos de botella, qué tipos de características son más complicadas.
  - Cosas que se pueden mejorar: metodología, arquitectura, tecnología.
  - Distintos tipos de representaciones gráficas nos permiten medir el proceso de desarrollo.





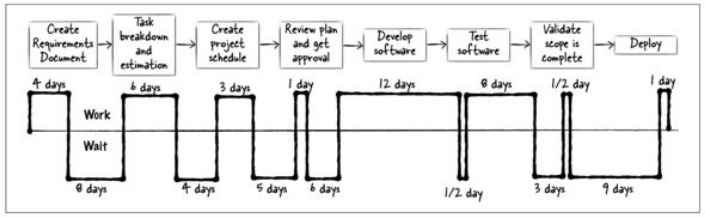


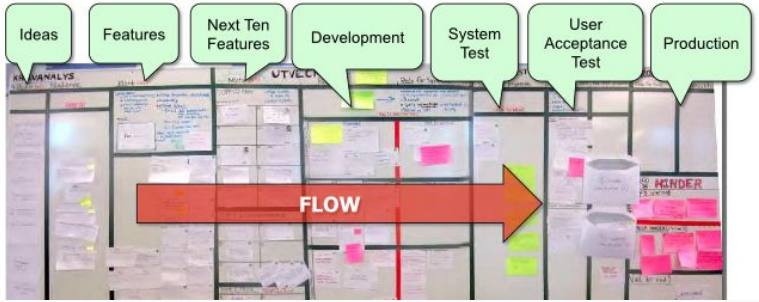
4. Decidir lo más tarde posible

- Toma las decisiones importantes sobre tu proyecto cuando tengas la mayor cantidad de información posible, en el último momento responsable.
- Diferencia entre **compromisos** y **opciones**. Intenta que la arquitectura y el desarrollo del proyecto permita maximizar las opciones.
- Se puede aplicar muy bien al desarrollo incremental: empezamos con un núcleo de desarrollo mínimo y se van añadiendo características no demasiado dependientes unas de otras. Hay que intentar que sea fácil **activar y desactivar** las características y que no haya demasiada interdependencias entre ellas.
- Pruebas A/B para comprobar una característica: los usuarios utilizan dos versiones del producto, una con la característica activada y otra desactivada. Se realizan estudios estadísticos para comprobar qué versión del producto proporciona más valor.
- Hay otra forma radical de tener más opciones: darte la posibilidad de deshacer decisiones. Ver el post de Kent Beck - *Taming Complexity with Reversibility*.

5. Entregar rápido

- Comprende el coste de los retrasos, y minimízalos utilizando sistemas pull con ciclos cortos.
- Las entregas y desarrollos rápidos pueden permitir proporcionar más características (o prototipos de las mismas) para que los clientes y PO puedan decidir con más información.
- Utiliza la técnica del **mapa de la cadena de valor** (***value stream mapping***) y la **característica mínima promocionable** (***minimal marketable feature***). El value stream mapping nos permite estudiar y mejorar el proceso de desarrollo y detectar los desperdicios.
- La cadena de valor nos permite definir las columnas del **tablero kanban** y comprobar cómo se mueven las tareas sobre ese tablero
  - Distintos tableros para representar distintos niveles: una característica se divide en varias tareas.
- Utiliza las técnicas de **entregas contínuas** (***continuous delivery***).





6. Respetar a la gente y potenciar el equipo

- Un equipo es un conjunto de personas comprometidas en conseguir un objetivo común.
- Establece un entorno de trabajo centrado y efectivo, y construye un equipo de personas con energía, que realizan un trabajo sostenible.
- No a los “héroes” que le echan un montón de horas y que sacan el trabajo adelante ellos solos. Eso no es potenciar el equipo.
- Respetar a los compañeros.

7. Optimizar el conjunto

- Todo el equipo debe tener claro el objetivo final del proyecto y cómo el software que estamos desarrollando va a añadir valor al usuario final.
- Una técnica útil es la técnica de los 5 por qués (5 whys): cuando quieras investigar la causa final de algo debes contestar a 5 por qués. El proceso de reflexión te permitirá ver cosas que mejorar. Un ejemplo:
  - *¿Por qué se producen tantos errores en el proceso de matrícula?* Porque mucha gente pulsa el botón atrás del navegador y nuestro programa no lo soporta.
  - *¿Por qué la gente pulsa el botón de atrás?* Porque quiere modificar alguna asignatura y no puede hacerlo de otra forma.
  - *¿Por qué no puede se puede modificar la matrícula de una asignatura?* Porque ya ha confirmado esa asignatura, ya se le ha asignado un turno y no se puede "desmatricular" de ese turno.
  - *¿Por qué no se puede desmatricular de un turno?* Porque no hay ninguna funcionalidad para que un estudiante pueda escoger el turno que le interese.
  - *¿Por qué no se ha añadido esa funcionalidad?* Porque no se nos había ocurrido. ¡¡Pues hay que modificar el backlog!!
- Contratos basados en la confianza. Por ejemplo, el cliente paga por características entregadas, hay un tope de gasto para el proyecto completo y la posibilidad de futuros contratos para ampliaciones de la aplicación.

Referencias y bibliografía de ampliación

- Mary Poppendieck, Implementing Lean Software Development, cap. 1 y 2
- Mary Poppendieck, Lean Programming, blog post 2001

Ejercicio 12

Utiliza los 7 principios del desarrollo leans para organizar el cursado y aprobación de este año.

Qué:

- Aplica a cada principio a las distintas etapas del proceso de cursado

Cómo:

- Para cada principio indique 3 ejemplos de aplicación que creas conveniente para tu situación con estudiante
- Agrega un ejemplo más que pueda servir para estudiantes en general

b) Utiliza un ML para realizar una tabla de relación de principios