Gestión de Proyectos Software

Principios ágiles en Scrum

Contenidos

- Desarrollo dirigido por planes
- Principios ágiles en Scrum
  - 1. Variabilidad e incertidumbre
  - 2. Predicción y adaptación
  - 3. Trabajo en marcha
  - 4. Progreso
  - 5. Rendimiento

Desarrollo dirigido por planes

Desarrollo dirigido por planes

- Tradicional, secuencial, predictivo, prescriptivo...
- Se planifica todo lo que el usuario querría en el producto final, y se determina cómo construirlo
- Cuanto mejor sea la planificación, mejor se entenderá el problema y por lo tanto mejor se podrá llevar a cabo
  - Esta es la idea principal
- Se adapta bien a problemas claramente definidos, predecibles y que no van a cambiar significativamente
  - Hay poco *feedback*, y llega tarde
- El problema es que la mayor parte de los desarrollos de productos son todo menos predecibles
  - Especialmente al principio

Desarrollo dirigido por planes

- Analiza (comprende) → diseña → "codifica" → prueba → despliega
- Parece que tiene sentido. **Debería** funcionar
  - Y si no funciona es porque no lo has analizado bien, o diseñado bien o...
  - Aunque no funcione, se sigue insistiendo. "Algo habremos hecho mal. Pero **debería** funcionar"
- Probablemente el problema no es que se haya hecho nada mal
- El problema es que este modelo **está basado en creencias, que no suelen ser ciertas** en el desarrollo de productos
  - Especialmente una: que podemos predecir y entender por anticipado todo lo que es relevante para el desarrollo del producto


YO SOY MUY LISTO
Y SI QUE PUEDO
memegenerator.es

¿De veras?

- ¿Ya sabe tu cliente (quizás eres tú mismo) todos los requisitos que va a tener tu producto?
  - ¿Y te los puede explicar con el detalle suficiente como para implementarlos? ¿El primer día/semana/mes?
  - ¿Y no va cambiar de opinión cuando vea el primer prototipo? Ah, no, que no le vas a enseñar ninguno... ¿Cuando vea un producto de la competencia, o la próxima versión de iOS, o un anuncio de detergente en TV o a su hijo jugar con la tablet?
- ¿Ya sabes cuáles van a ser tus usuarios?
- ¿Ya sabes cómo van a reaccionar a tu producto?
  - ¿Reaccionarán todos igual, o variará por grupos, conocimientos, mercados...?
  - Supongo que no quieres hacer un producto fantástico que nadie quiere

¿De veras?

- ¿Conoces todas las tecnologías que vas a usar? ¿Y cómo van a evolucionar (o a dejar de hacerlo) mientras tú desarrollas tu producto?
  - Lenguaje de programación (p.ej. Python), plataforma (p.ej. JVM), framework (p.ej. Ruby on Rails), SO (escritorio[s]/móvil[es]), navegador web, bibliotecas de software...
- ¿Ninguno de tus desarrolladores se va a ir a mitad del proyecto? ¿Tampoco vas a contratar a alguien nuevo?
- Y aunque supieras todo lo anterior, ¿eres capaz de predecir antes de empezar todas las posibles interacciones y problemas que pueden surgir?
  - Ni el conocimiento, ni la experiencia, ni herramientas, métodos o análisis de riesgos, te permiten anticipar los "*unknown unknowns*"

En resumen

Tu producto es más complejo de lo que puedes entender al principio

Tienes un entorno que no puedes controlar ni predecir, y que te influye mucho


UNO NO PUEDE VER MÁS ALLÁ...
...DE UNA ELECCIÓN QUE NO ENTIENDE

Principios ágiles en Scrum

1. Variabilidad e incertidumbre

Variabilidad e incertidumbre

- Con Scrum puedes aprovechar la variabilidad y la incertidumbre en el desarrollo de productos para crear soluciones innovadoras. Para ello:
  - Acepta la variabilidad útil
  - Usa un desarrollo iterativo e incremental
  - Aprovecha la variabilidad mediante inspección, adaptación y transparencia
  - Reduce las incertidumbres

Acepta la variabilidad útil

- **El desarrollo de productos no es como la fabricación de productos**
- En **fabricación** se toman unos requisitos fijos y se siguen unos pasos conocidos para fabricar productos que siempre son iguales (con cierta variedad limitada)
- En **desarrollo** el objetivo es crear una única instancia de un producto
- Esta instancia es análoga a una receta
  - Y no queremos inventar la misma receta dos veces
- La variabilidad aparece porque
  - Creas un producto nuevo cada vez
  - Cada característica de un producto es distinta a cualquier otra del mismo producto

Desarrollo iterativo e incremental

- **Iterativo**
  - Entenderemos las cosas mal antes de entenderlas bien, y las haremos mal antes de hacerlas bien
  - Por tanto planificamos múltiples pasadas para mejorar lo que hacemos hasta llegar a una buena solución
  - Prototipo → prototipo mejorado → … → primera versión → … → versión final
- **Incremental**
  - Dividimos el producto en partes pequeñas, y las vamos construyendo, y probando y encajando
  - Esto nos proporciona información muy valiosa sobre lo que estamos haciendo, para que podamos adaptarnos

Desarrollo iterativo e incremental en Scrum

- En cada sprint se crea un incremento de producto
  - Algo de análisis, diseño, implementación, integración y pruebas en cada sprint
- En el plazo (fijado y corto) de un sprint, comprobamos si nuestro análisis es correcto, si el diseño funciona, si la implementación es factible...
  - E intercambiamos *feedback* con el cliente

Aprovecha la variabilidad

- Mediante inspección, adaptación y transparencia
- Scrum acepta que el proceso necesario para crear algo nuevo es complejo y desafía una definición completa a priori
  - Por lo tanto, genera *feedback* pronto y frecuentemente para asegurarse de que se construye el producto correcto y que se construye correctamente
- Para ello se basa en la transparencia
  - La información importante está disponible para los que crean el producto
- La transparencia permite la inspección, que es un requisito para la adaptación
  - También lleva a más comunicación y confianza (en el proceso y entre los miembros del equipo)

Reduce la incertidumbre

- Incertidumbre de fines
  - Sobre las características del producto final
- Incertidumbre de medios
  - Sobre el proceso y las tecnologías usados para desarrollar el producto
- Incertidumbre de clientes
  - Sobre quién usará el producto (típica por ejemplo en *start-ups*)

Reduce la incertidumbre

- Las metodologías tradicionales se focalizan en eliminar la incertidumbre de fines definiendo completamente lo que se va a construir antes de empezar
  - Y luego ya la de medios
- El problema es que en el dominio complejo, ambas incertidumbres suelen estar estrechamente relacionadas
  - Y también la de clientes

Reduce la incertidumbre

- Scrum aborda todas las incertidumbres al mismo tiempo
- Gracias a la aproximación iterativa e incremental guiada por la inspección y adaptación continuas y a la transparencia
- Esto permite sondear el entorno para identificar y aprender sobre los "*unknown unknowns"* según van surgiendo

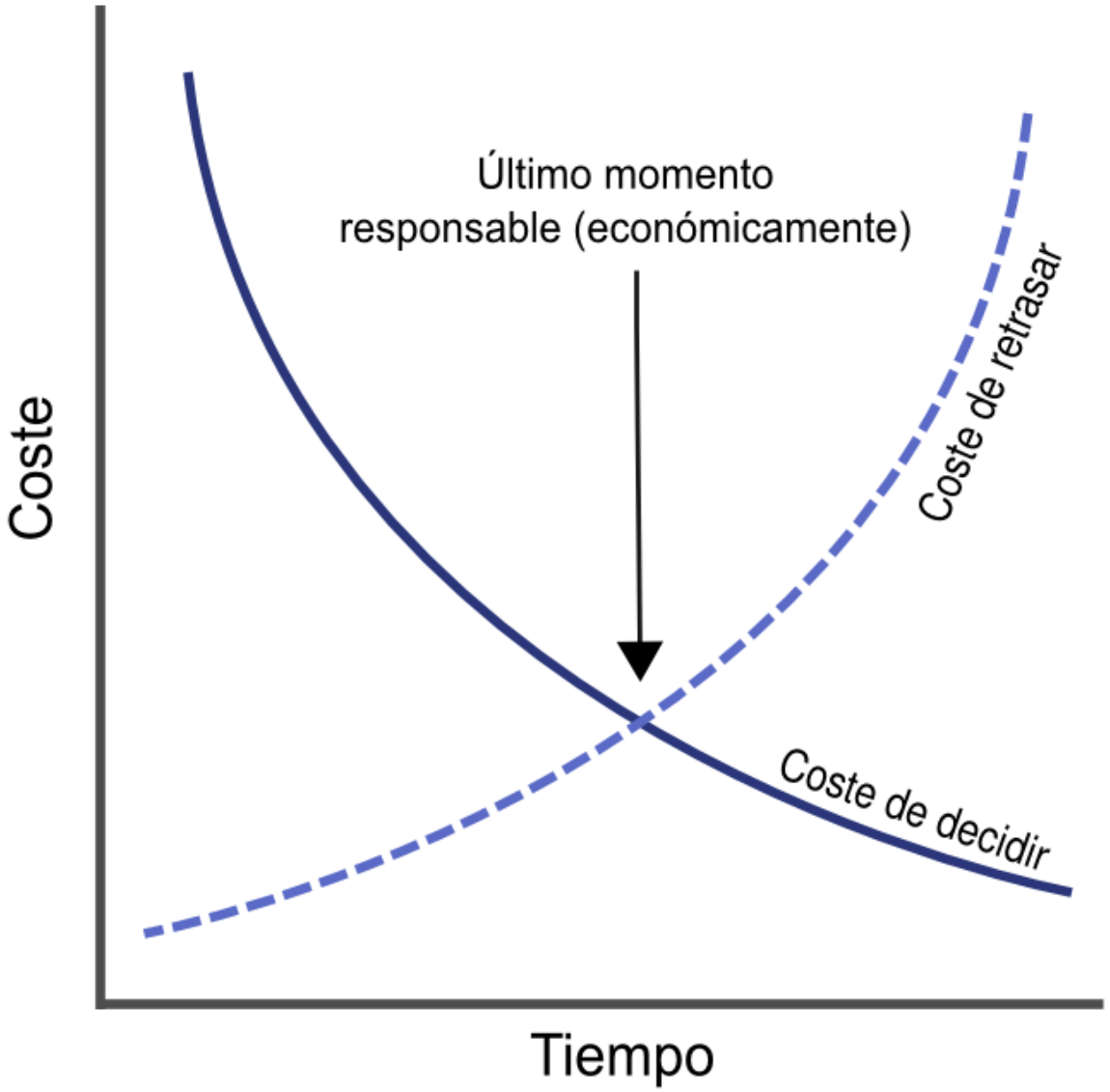
2. Predicción y adaptación

Predicción y adaptación

- Con Scrum se intenta equilibrar el deseo de predecir con la necesidad de adaptarse. Para ello:
  - Mantén opciones abiertas
  - Acepta que no lo puedes entender bien por adelantado
  - Favorece una aproximación adaptativa y explorativa
  - Acepta el cambio de forma sensata
  - Equilibra el trabajo por adelantado con el trabajo adaptable y *just-in-time*
  - Usa aprendizaje validado

Mantén opciones abiertas

- En desarrollo tradicional, hay decisiones que se tienen que tomar, revisar y aprobar antes de poder avanzar de fase
  - Aún cuando solo se tenga un conocimiento limitado de la situación
- Scrum no exige tomar decisiones prematuras porque lo dicte un proceso
  - Se toman en el último momento responsable (*last responsible moment*)
- ¿Cual es el último momento responsable?
- Justo antes de que el coste de no tomar la decisión sea mayor que el de tomarla
  - Determinar cuándo es este momento no es obvio


Último momento
responsable (económicamente)
Coste de retrasar
Coste de decidir
Coste
Tiempo

Acepta que no lo puedes entender bien por adelantado

- Si ni siquiera vas a tener todos los requisitos, ¿cómo vas a crear planes detallados correctos por adelantado?
- Scrum acepta esto con entusiasmo
  - Mucho de lo que necesitas saber solo surgirá más adelante
  - Crear una gran cantidad de requisitos de baja calidad no es útil
- Con Scrum, también se documentan requisitos y planes por adelantado
  - Pero solo lo suficiente para empezar, y con la asunción de que completaremos los detalles cuando sepamos más sobre el producto que estamos creando

Favorece una aproximación adaptativa y explorativa

- En desarrollo tradicional usas lo que sabes y predices lo que no
- En Scrum te adaptas, y aplicas prueba-y-error mediante una **exploración apropiada**
  - Explorar es obtener conocimiento haciendo algo
    - Crear un prototipo, una prueba de concepto, realizar un estudio o llevar a cabo un experimento

DISCO STU DICE

Favorece una aproximación adaptativa y exploRativa

- La exploración tiene un coste
  - Si este coste es elevado, tiene sentido, económicamente hablando, dedicar el esfuerzo a explotar lo que sabes y a tratar de predecir lo que no para alcanzar una buena solución
- Por fortuna, lo que en los 80 (*) podía resultar costoso, hoy cada día lo es menos. Hay menos razones que desincentiven la **exploración consciente y sistemática**
  - No hablamos de prueba-y-error a ciegas. Hablamos de exploración sistemática
- A menudo cuesta menos crear algo para explorar el *feedback* de los usuarios, que tratar de adivinarlo para acertar por adelantado

(*) Lo siento por Disco Stu, pero hace mucho que no estamos en los 80

Exploración sistemática: definición gráfica

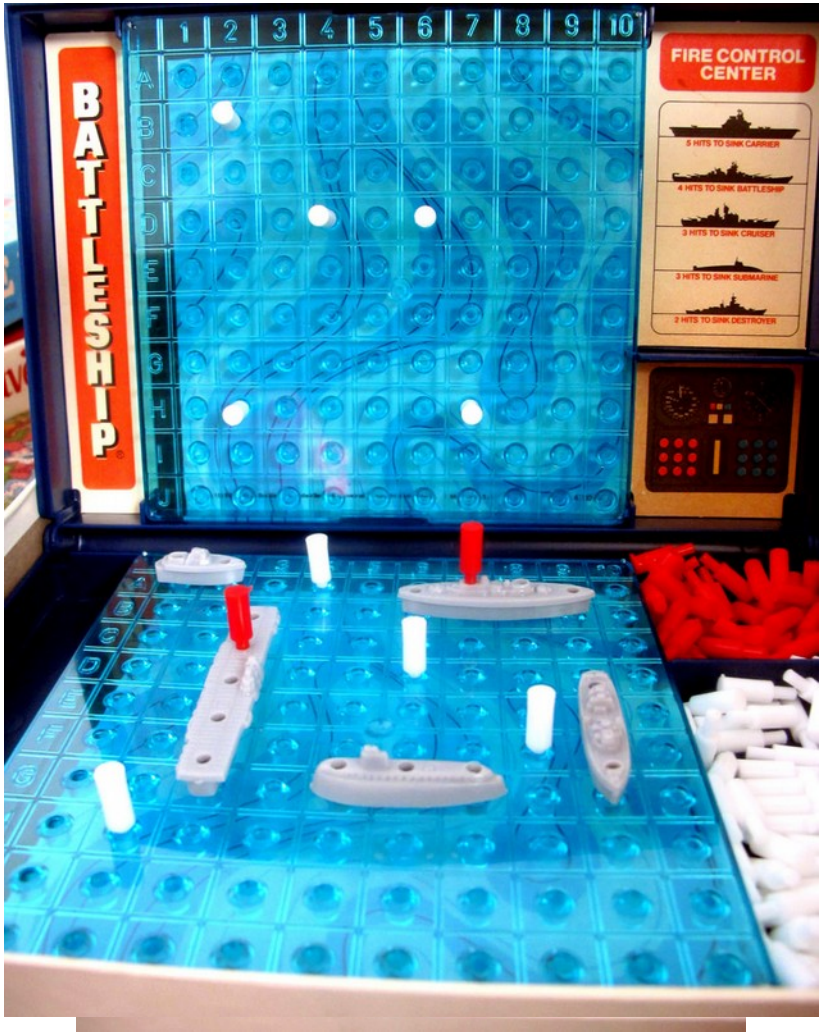


Photo CC-BY 2.0 by frankieleon https://www.flickr.com/photos/armydre2008/

Acepta el cambio de forma sensata

- En desarrollo secuencial, el cambio es más caro cuanto más tarde se produce
  - En captura de requisitos es barato, en producción es carísimo
- Para minimizar cambios se trata de predecir con precisión lo que necesitamos. Por desgracia esto no suele funcionar
  - 1. Se invierte mucho esfuerzo al principio, haciendo suposiciones poco informadas
  - 2. Se trabaja basándose en estas suposiciones
  - 3. Tarde o temprano las suposiciones acaban siendo comprobadas
    - Y si hay que descartarlas, hay que cambiar cosas y tirar parte de lo que habíamos hecho, con su consiguiente coste

Acepta el cambio de forma sensata

- En Scrum, el cambio es la norma
  - Pues la incertidumbre es inherente al desarrollo de un producto
- El objetivo por tanto es intentar que el coste del cambio sea constante (o cercano a) con respecto al tiempo
- Con Scrum se producen muchos productos de trabajo
  - Requisitos detallados, diseños, casos de prueba...
- Con filosofía *just-in-time*
  - Evitando la creación de artefactos posiblemente innecesarios
- Así, cuando se produce un cambio hay menos artefactos y decisiones que se basaron en las suposiciones que ha habido que descartar o rehacer
  - El coste del cambio así es más proporcional al tamaño del mismo, y menos al momento en que se produce
  - En algún momento el coste acabará dejando de ser proporcional al tamaño del cambio, pero este momento llegará más tarde que en desarrollo tradicional

Equilibra trabajo por adelantado y *just-in-time*

- En Scrum el trabajo por adelantado tiene que ser útil sin ser excesivo
  - Incluye requisitos, planes...
  - Maximizar la adaptabilidad basada en *feedback* frecuente
  - Minimizar el trabajo por adelantado
- El equilibrio depende del producto y de las incertidumbres sobre el mismo (de fines y de medios)
  - También de otras restricciones externas o requisitos legales
- El equilibrio está en algún punto entre "adivinar el futuro" (puramente predictivo) y "el caos" (puramente adaptativo)

Usa aprendizaje validado (*validated learning*)

- Confirma o refuta una suposición
- Hay que validar las suposiciones importantes pronto
  - Sin validar, son un riesgo grave
- Hay que aprender cíclicamente
  - Suposición-> experimento-> *feedback*-> aprendizaje
  - Scrums diarios, revisiones de sprints...
- El flujo de trabajo debe proporcionar *feedback* rápidamente
  - Por ejemplo, si dos componentes tienen que interactuar no esperamos a terminarlos para empezar la integración

3. Trabajo en marcha (*work in progress / process*)

Trabajo en marcha (*Work in Progress/Process - WIP*)

- El WIP es todo el trabajo empezado que aún no se ha terminado
- En general queremos tener un WIP bajo
  - La cantidad óptima de WIP depende de muchos factores y cambia con el tiempo, así que prepárate a ser flexible con esto también
- Para optimizar tus resultados gestionando correctamente el WIP, hay varios elementos que puedes considerar
  - Usa lotes de tamaños sensatos
  - Reconoce el inventario y gestiónalo para que circule bien
  - Focalízate en el trabajo parado, no en los trabajadores parados
  - Considera el coste del retraso

¿Por qué un WIP bajo? La ley de Little

- El número medio de clientes en un sistema (L) es igual a la tasa de llegada (λ) multiplicada por el tiempo medio que el cliente pasa en el sistema (W)
  - L = λ * W
- También se puede expresar como que el número medio de clientes en un sistema (L) es igual al tiempo medio que el cliente pasa en el sistema (W) multiplicado por la productividad del mismo (*throughput*) (X)
  - L = W * X
  - La productividad es la tasa de salida del sistema
  - Ten en cuenta que según sea nuestro sistema, puede que no estemos midiendo la misma cosa si consideramos λ o si consideramos X
- Un ejemplo: Si llegan 12 clientes por hora a nuestra tienda (λ), y están de media 15 minutos (W), el número medio de clientes en la tienda en cualquier momento será de L = 12 * 0,25 = 3
  - Aunque es razonablemente intuitivo, es sorprendente que esta relación no se vea afectada por la distribución del proceso de llegadas, la distribución o el orden de servicio a los clientes etc.
  - Se aplica a todo tipo de sistemas

¿Por qué un WIP bajo? La ley de Little

- Si queremos calcular el tiempo medio que pasan los clientes en el sistema (W) podemos escribir la ecuación como W = L / λ o bien como W = L / X
- Para aplicarlo en nuestro problema (saber el tiempo que nos cuesta completar las tareas que tenemos), los clientes son las tareas y el tiempo que pasan en nuestro sistema es el tiempo que tardamos en llevarlas a cabo
- Podemos escribir la última ecuación usando otra terminología: Tiempo de ciclo = WIP / X
  - Tiempo de ciclo (*): Tiempo medio que cuesta completar una tarea desde que nos ponemos a trabajar en ella
  - WIP: número de tareas en las que estamos trabajando en un momento dado
  - X: el número de tareas que se completan por unidad de tiempo
- Por ejemplo, si trabajas en 12 tareas a la vez (WIP), y completas 12 tareas al mes (X), el Tiempo de Ciclo será de 1 mes
  - Es decir, cada tarea tarda 1 mes en ser completada (en media)
- Si decides trabajar solo en 6 tareas a la vez (**reduces tu WIP a la mitad**), y no cambias nada más, el Tiempo de Ciclo pasará a ser de medio mes. **Harás las mismas tareas en un mes, 12, pero cada una tardará en completarse la mitad del tiempo** (en media)

(*) El término "tiempo de ciclo" se usa para distintos conceptos (algunos de ellos parecidos entre sí), así que cuando lo encontréis en algún sitio, aseguraos de que sabéis a qué se refieren en ese contexto

Usa lotes de tamaños sensatos

- Desarrollo secuencial: en un trabajo por fases, es mejor terminar una fase antes de empezar otra
  - P.ej.: capturar todos los requisitos primero
  - El tamaño del lote en este caso es de un 100%
- Esto se basa en las economías de escala de la fabricación de productos
  - El coste de producir una unidad es menor conforme aumenta el número de unidades (el tamaño del lote)
- Pero los lotes pequeños tienen muchos beneficios

Beneficios de los lotes pequeños

- Tiempo de ciclo reducido
  - Los lotes más pequeños producen menores cantidades de trabajo esperando a ser procesado, lo que significa menos tiempo esperando a que se haga el trabajo
    - Conseguimos resultados antes
- Menor variabilidad en la circulación
  - Pensad en un restaurante donde entran y salen pequeños grupos (se les sirve pronto, hay buena circulación). Ahora imaginad que llega un autobús con turistas (lote grande), y el efecto que produce en la circulación de gente en el restaurante
- *Feedback* acelerado
  - Los lotes pequeños nos permiten tener *feedback* antes, haciendo menores las consecuencias de cometer algún error

Beneficios de los lotes pequeños

- Sobrecarga reducida
  - Manejar lotes grandes tiene una sobrecarga
    - P.ej. mantener una lista de 300 *items* de trabajo requiere más esfuerzo que una lista de 30
- Motivación y urgencia incrementadas
  - Los lotes pequeños proporcionan foco y sentido de la responsabilidad
    - Es más fácil entender el efecto de los retrasos y los fallos cuando se trabaja con lotes pequeños que con los grandes
- Menor incremento de costes y plazos
  - Cuando nos equivocamos en lotes grandes, nos equivocamos por mucho respecto al coste y al plazo
    - En una escala menor, no nos equivocamos tanto

Reconoce el inventario y gestiónalo para que circule bien

- El inventario tiene un alto coste asociado
  - En este aspecto, la fabricación y el desarrollo de productos sí que coinciden
  - El inventario pierde valor mientras lo tenemos almacenado (y almacenarlo tiene un coste)
  - Si hay cambios en el diseño del producto, el inventario almacenado ya no sirve
- En el desarrollo de productos no es obvio ver que tenemos inventario
  - Se trabaja con muchos bienes inmateriales (digitales) que parece que "no estorban"
- Un ejemplo: necesitamos requisitos para trabajar, pero no los necesitamos todos
  - Si tenemos demasiados, cuando el cliente pida cambios tendremos un problema más grande
  - Si no tenemos suficientes, el trabajo de alguien puede quedarse estancado (mala circulación)

Focalízate en el trabajo parado, no en los trabajadores parados

- Trabajo parado
  - Lo que queremos hacer pero algo nos lo impide
    - Quizás tenemos que esperar a que otros terminen su parte
    - Quizás es que tenemos demasiado trabajo al mismo tiempo
- Trabajadores parados
  - Los que tienen disponible tiempo para hacer más cosas de las que hacen

Focalízate en el trabajo parado, no en los trabajadores parados

- En la empresa tradicional, si contratas a alguien para hacer tests, lo quieres el 100% del tiempo trabajando en eso
  - Si no lo está, le buscas más trabajo, aunque sea en distintos proyectos
- Eso reduce el desperdicio que supone un trabajador parado, pero incrementa el desperdicio que supone tener trabajo parado
  - Y en general no compensa
- Una analogía para ilustrar el punto anterior: cuando el PC está ocupado al 100% (procesador y memoria), ¿qué ocurre?
  - Se enlentece enormemente

Pregunta

- Suponed que un trabajador tarda 45 minutos de media en hacer cada tarea que se le encarga (tiempo de servicio) y le encargamos de media 1 tarea cada hora
- ¿Cuánto tiempo, en media, tarda una tarea en estar terminada desde que la hemos solicitado?
  - Las solicitudes de tareas no están uniformemente distribuidas en el tiempo, sino que llegan aleatoriamente y sin saber el tiempo que ha pasado desde que llegó la anterior
- ¿Y si encargamos 1,1 tareas por hora?

Respuesta: un poco de teoría de colas

- Si asumimos que las tareas llegan con una distribución de Poisson con tasa λ, los tiempos de servicio tienen una distribución exponencial con parámetro 1/μ (siendo μ el ritmo medio de servicio) y sabemos que tenemos un solo trabajador, tenemos lo que en teoría de colas se denomina una cola M/M/1:
  - El ratio λ/μ se llama utilización (ρ). Es la proporción de tiempo en que el trabajador está ocupado
  - Si ρ es > 1, las tareas llegan más rápido de lo que las podemos realizar, y por tanto la cola crecerá sin límites
  - Si ρ es < 1, la cola alcanzará una estabilidad con cierto número de tareas en espera en promedio
  - El tiempo total medio que una tarea está en el sistema (esperando+siendo realizada) es W = 1 / (μ - λ)
- En el ejemplo, μ = 1,33 tareas por hora (1 tarea en 45 minutos)
- Si encargamos λ = 1 tarea por hora: W = 1 / (1,33 - 1) = 3,03 horas por tarea
  - Y el trabajador está ocupado el 75% del tiempo
- Si encargamos λ = 1,1 tareas por hora: W = 1 / (1,33 - 1,1) = 4,35 horas por tarea
  - Y el trabajador está ocupado el 83% del tiempo
- Es decir, que en este caso si pasamos de un trabajador que está ocupado el 75% de su tiempo a uno ocupado el 83% del tiempo (**solo un 8% más**), cada tarea **tardará de media un 43% más de tiempo** en estar completada desde que la encargamos

Respuesta: un poco de teoría de colas

- Si ponemos a un segundo trabajador, y mantenemos el resto de asunciones, tenemos lo que en teoría de colas se denomina una cola M/M/c con c = 2:
  - El ratio λ/(c*μ) se llama utilización (ρ). Es la proporción de tiempo en que cada trabajador está ocupado
  - Si ρ es > 1, las tareas llegan más rápido de lo que las podemos realizar, y por tanto la cola crecerá sin límites
  - Si ρ es < 1, la cola alcanzará una estabilidad con cierto número de tareas en espera en promedio
  - El tiempo total medio que una tarea está en el sistema (esperando+siendo realizada) (W) no tiene una ecuación tan corta como en el caso M/M/1 (buscadla), pero os pongo un par de resultados
- Si λ = 1 tarea por hora: W = 0,88 horas por tarea
  - Y cada trabajador está ocupado el 38% del tiempo
- Si λ = 1,1 tareas por hora: W = 0,91 horas por tarea
  - Y cada trabajador está ocupado el 41% del tiempo

Considera el coste del retraso

- El coste del retraso es el coste financiero de retrasar trabajo o de retrasarse en alcanzar algún hito
- Si se cuantifica el trabajo acumulado (inventario) y el coste del retraso, se puede apreciar por qué es más importante eliminar el desperdicio del trabajo parado que el de los trabajadores parados

Considera el coste del retraso: un ejemplo

- ¿Contratamos a alguien para escribir documentos (manuales etc.) los 12 meses de duración de un proyecto, o lo contratamos solo al final del mismo?
  - Sabiendo que durante los 12 meses no va a estar trabajando ni de casualidad el 100% de su tiempo
- 12 meses de salario: 75000 €
  - El trabajo se termina en 12 meses, con documentos incluidos
- Lo contratamos al terminar los 12 meses de desarrollo durante 2 meses: 12500 €
  - El trabajo se termina en 14 meses, con documentos incluidos

Considera el coste del retraso: un ejemplo

- Si el coste de retrasar la salida del producto durante 2 meses es mayor que 75000-12500 = 62500€ nos saldría a cuenta contratarle desde el principio y pagarle el año entero
  - Aunque se pase el año entero sin hacer nada más que trabajar en nuestro producto, y por tanto solo esté ocupado un porcentaje bajo de su tiempo
- Es importante no focalizarse en optimizar el uso del documentalista si esto empeora el resultado económico global

4. Progreso

Progreso

- Scrum mide el progreso según lo que se ha entregado y validado, no por el ajuste a un plan predefinido
- Lo primero es adaptarte en tiempo real y replanificar
  - La fe en un plan no te deja ver cuando falla. Adáptate al cambio

Progreso

- Mide el progreso validando resultados del trabajo
  - No por terminar una fase y empezar otra has progresado
  - Terminar a tiempo y dentro del presupuesto sin alcanzar las expectativas del cliente no es realmente éxito
  - Scrum mide el progreso creando y validando **resultados**
- Focalízate en entregar valor
  - Con un plan tradicional, la entrega de valor se hace al final. Si te quedas sin recursos antes, no entregas nada
  - Los artefactos intermedios (p.ej. documentos de diseño) solo son valiosos internamente. Y aún así, solo si finalmente sirven para entregar resultados al cliente
  - En Scrum, las características más valiosas para el cliente son las que se abordan antes. Los clientes reciben valor desde el principio

5. Rendimiento

Rápido, pero sin apresurarse

- Scrum busca entregar rápido, obtener *feedback* rápido y dar valor al cliente pronto
- Rápido no significa apresurado. El objetivo es ir a un ritmo sostenible (*sustainable pace*)
  - El que puede mantenerse por un largo periodo de tiempo
  - Apresurarse disminuye la calidad

Calidad

- En desarrollo dirigido por planes, se supone que la calidad llega por seguir el plan, pero no se verifica realmente hasta que se integra (al final)
- En Scrum, la calidad es continua. Se prueba y se verifica en cada sprint
  - Cada incremento de valor se crea como si fuera a ponerse inmediatamente en manos del cliente

La mínima ceremonia que sea suficiente

- El desarrollo dirigido por planes suele ser ceremonioso (firma, sello, aprobación por un comité...) y estar centrado en documentos
- Scrum está centrado en el valor, así que no se enfatiza la ceremonia
  - Se evita especialmente la formalidad innecesaria (que tiene coste y añade poco o nada de valor)
  - Hacer cosas solo porque las dicta un proceso, tener que aprobar cosas por comité antes de empezar otras fases, tener que registrar cualquier petición de cambio (por mínima que sea) en cierta aplicación...
- La ceremonia mínima suficiente no es la misma para una nueva red social que para un marcapasos

La mínima ceremonia que sea suficiente

- Scrum no está en contra de la documentación
  - Solo está en contra de la documentación que no añade valor
- Documentación que añade valor
  - La entregable (manuales, instrucciones de instalación...)
  - La que captura discusiones, decisiones o acuerdos importantes
  - La que facilita a nuevos miembros del equipo ponerse en marcha rápidamente
  - La que es exigida por ley en nuestro mercado/producto
  - ...

Bibliografía

- Kenneth S. Rubin. *Essential Scrum. A practical guide to the most popular agile process*
  - Chapter 3 (Agile Principles)