Leire Requena García, Clara María Romero Lara
Metodologías de Desarrollo Ágil
January 3, 2023

Relación de problemas 4

UNIVERSIDAD DE GRANADA

1. Tema 5: eXtreme Programming

Ejercicio 1. El problema básico del desarrollo del software es el riesgo, algunos ejemplos de situaciones de riesgo son:

- **Retrasos de planificación: llega el día de la entrega, y le decimos al cliente que el software no estará disponible.**
- **Proyecto cancelado: después de numerosos retrasos, el proyecto se cancela sin haber entrado nunca en producción.**
- **El sistema se deteriora: el software se pone satisfactoriamente en producción, pero después de un par de años, los costes de hacer cambios o la tasa de defectos crecen tanto que el sistema debe ser reemplazado.**
- **Tasa de defectos: el software se pone en producción, pero la tasa de defectos es tal alta que no se usa.**
- **Requisitos mal comprendidos: el software se pone en producción, pero no resuelve el requisito planteado inicialmente.**
- **Cambios en el negocio: el software se pone en producción, pero el problema de negocio para el que se diseñó la solución inicial fue reemplazado hace seis meses por otro problema más acuciante.**
- **Falsa riqueza de características: el software tiene un montón de características potencialmente interesantes, todas las cuales fueron divertidas de programar, pero ninguna hace que el cliente gane dinero.**
- **Cambios de personal: después de dos años, todos los buenos programadores del proyecto comienzan a odiar el programa y se marchan.**

¿Cómo trata XP el riesgo que hemos comentado anteriormente?
Responsable: Clara

- Retrasos de planificación - XP lidia con los retrasos mediante una planificación flexible y adaptativa. Las estimaciones no son planes de futuro, sino el mejor escenario posible. Además, los sprints de pequeño tamaño también ayudan a evitar esta situacion
- Proyecto cancelado - No sucederá a no ser que se cancele en etapas extremadamente tempranas. XP genera en cada sprint de corta duración contenido apto para producción.
- El sistema se deteriora - Una de las máximas de XP es el énfasis en las pruebas. El contenido nuevo se añade al programa en un proceso de producción e integración continua, lo cual genera un sistema estable.
- Tasa de defectos - De nuevo, las pruebas son muy importantes en XP, así como las revisiones de código en parejas y un diseño sólido y revisado diariamente.
- Requisitos mal comprendidos - En XP se acepta el cambio como una realidad. Durante la vida de un proyecto, los requisitos cambiarán, se malinterpretarán, se volverán innecesarios... La planificación es un diálogo continuo y, si algo cambia, nos adaptamos a ello y continuamos.
- Cambios en el negocio - Mismo planteamiento que la anterior: el cambio es parte del proceso de desarrollo. Pero gracias a la metodología de XP, sabemos que no pasarán 6 meses hasta que nos demos cuenta de este problema, ya que el diálogo continuo y los sprints cortos nos permitirán implementar los cambios necesarios mucho antes.
- Falsa riqueza de características - En XP se premia la simplicidad: hay que hace lo más sencillo que funcione. Esto requiere invertir tiempo previo en diseño, pero como máxima nos garantiza que este riesgo no va a darse.
- Cambios de personal - El código pertenece a todo el equipo, y nos apoyamos los unos a los otros. Si tras un tiempo los programadores más veteranos del programa deciden marcharse, el resto del equipo debería conocer el código tan bien como ellos. Todos los miembros del equipo deberían sentirse cómodos con todo el código del programa.

Ejercicio 2. Comenta brevemente las diferencias principales entre XP y Scrum.
Responsable: Clara

La diferencia más notable es que Scrum no determina metodologías de desarrollo, sino un marco de trabajo que adaptar a nuestro entorno de trabajo. En XP, se recomiendan técnicas específicas (como la programación en parejas). Esto nos permite aplicar XP directamente a un equipo, cuando para aplicar Scrum tendríamos que completar el marco de trabajo con otras metodologías de desarrollo.

Otra diferencia sería la duración del sprint y su flexibilidad: en Scrum, los sprints duran un mes; y el tiempo no es negociable. En XP, los sprints son mucho más cortos (de una a tres semanas), y si hay problemas, se puede repetir la reunión de planificación y reestimar.

Finalmente, Scrum se centra mucho más en prácticas enfocadas a la auto-organización; mientras que XP se dedica más a reforzar las buenas prácticas de desarrollo

Ejercicio 3. Compara los roles de XP con los roles de Scrum
Responsable: Clara

- Programador VS Equipo de desarrollo: El rol del equipo de programadores en XP y equipo de desarrollo en Scrum es prácticamente el mismo. Se encargan del apartado técnico, estiman las historias y hacen las pruebas de unidad. Además, pese a su nombre, no es un rol limitado a los programadores: se incluyen todos los roles necesarios para el desarrollo del producto (diseñadores, programadores...)
- Cliente VS Product Owner: Es el mismo rol. Se encarga de ser el nexo entre el producto y los desarrolladores, es parte del equipo y como tal tiene ciertas responsabilidades con ellos: se encarga de las historias de usuario y su priorización, así como de las pruebas funcionales, ya que es la persona encargada de dar el visto bueno a lo producido en la iteración.
- Entrenador, Gestor, Rastreador VS Scrum Master:
  - Similitudes:
    - Entrenador: responsable del proceso y su llegada a buen puerto. Aunque está formado en su campo, a la hora de la programación queda en un segundo plano, ya que su trabajo se centra más en garantizar que los procesos se cumplen correctamente y el equipo funciona.
    - Gestor: nexo entre el cliente y los programadores.
    - Rastreador: ambos son encargados de llevar seguimiento del proyecto, reflejarlo en gráficos, interpretar los datos y aplicar cambios acordes a éstos.
  - Diferencias:
    - Los roles de Entrenador y Gestor tienen una carga jerárquica, se ubican encima de los desarrolladores. En Scrum, el Scrum Master no puede considerarse un superior del equipo de desarrollo.
- Probador VS Stakeholder:
  - Similitudes:
    - El rol de probador se corresponde en parte con el de stakeholder, puesto que son los que proveen feedback sobre el producto y lo testean.
  - Diferencias:
    - Los stakeholders son normalmente un grupo de directivos y otros implicados. Un probador podría ser cualquiera.

Ejercicio 4. Relacionar los 14 principios de XP2 con las prácticas propuestas en XP
Responsable: Clara

Principio	Práctica	Descripción
Humanidad	Historias de usuario	**Humanidad**: El software lo desarrollan yusan personas, y es importante tener presentes los factores humanos para crear un software de calidad. **Historias de usuario:** Las historias de usuario se favorecen frente a los requisitos, porque nos muestran la experiencia real del usuario. Los requisitos dan ideas de desarrollo estático, sin lugar a cambios.
Economía	Negociar contratos y su alcance	**Economía:** El producto que se cree debe ser rentable tanto a corto como a largo plazo, debe producir beneficios. **Negociar contratos y su alcance:** Planteamos términos y condiciones realistas para ambos bandos, en los que el cambio y la negociación se dan por supuestos.
Beneficio Mutuo	Cliente in-situ	**Beneficio mutuo:** Pensar siempre en el beneficio de todas las partes implicadas en un desarrollo. **Cliente in-situ:** El cliente es parte del equipo de desarrollo y toma decisiones que le ayudan a alcanzar los objetivos de su negocio.
Auto-similitud	Estándares de programación	**Auto-similitud:** Buscar soluciones similares en diferentes contextos. Los patrones de la forma de trabajo deben repetirse adaptándolos a la situación en la que estemos. **Estándares de programación:** Desarrollamos siguiendo estándares que facilitan la lectura y modificación del código. El código debe ser cómodo para todo el equipo.
Mejora	Integración continua	**Mejora:** Excelencia en el desarrollo a través de la mejora. Se busca perfeccionar, no lo perfecto. **Integración continua:** Lo mejor es más frecuente, integramos y enviamos el código cada día para no postergarlo. Trabajamos continuamente sobre nuestro código para mejorarlo.
Diversidad	Pair programming	**Diversidad:** Dos ideas sobre un diseño presentan una oportunidad, no un conflicto. **Pair programming:** Todo el código que va a producción debe ser escrito por dos personas sentadas frente a la máquina. Sus conversaciones y discusiones harán posible obtener una mejor solución que si fuera código de un solo autor.

Principio	Práctica	Descripción
Reflexión	Espacios de trabajo informativos	**Reflexión:** Los buenos equipos piensan cómo y por qué hacen el trabajo, no esconden errores. **Espacios de trabajo informativos:** Nuestro espacio de trabajo refleja el estado del proyecto, así como de los imprevistos e incidencias que han dado pie a cambios en la planificación.
Flujo	Ciclo semanal	**Flujo:** Entrega continua de valor. **Ciclo semanal:** Al final de cada semana entregamos un producto utilizable tras un ciclo corto.
Oportunidad	Slack	**Oportunidad:** Los problemas son oportunidades de cambio. **Slack:** El cambio es parte del ciclo de vida de un proyecto. Es importante poder discutir y renegociar los compromisos que hicimos cuando surgen problemas.
Redundancia	Pruebas antes de implementar	**Redundancia:** Los problemas difíciles deben poder resolverse de diferentes maneras. **Pruebas antes de implementar:** Escribimos las pruebas antes que el código. Si éste supera las pruebas, será una solución válida, sin importar el modo en el que se haya resuelto.
Fallo	Refactoring	**Fallo:** Un fallo no es un desperdicio si me sirve para aprender algo. **Refactoring:** Realizar modificaciones al código de forma que afecte a su estructura interna, no a la externa. Nos permite mejorar el programa existente para añadir funcionalidad.
Calidad	Trabajo energizado	**Calidad:** No vas a ir más rápido aceptando rebajar la calidad. **Trabajo energizado:** Trabajamos a pleno rendimiento, pero solo tantas horas como se pueda ser productivo de forma sostenida. Cuando la calidad desciende, paramos.
Pasos de bebé	Diseño incremental	**Pasos de bebé:** ¿Qué es lo menos que puedes hacer que sea reconocible y que vaya en la dirección correcta? **Diseño incremental:** Diseño gradual, con pasos pequeños y seguros. No todo se hace al principio.
Responsabilidad	Sentarse juntos	**Responsabilidad:** El que acepta algo se hace responsable de ello. La responsabilidad no se asigna, se acepta. **Sentarse juntos:** Disponer de un espacio abierto común a todo el equipo, favorece la comunicación, las reuniones y las conversaciones cara a cara, creando un entorno humano en el que comunicarnos con nuestros compañeros

Ejercicio 5. Clasifica las prácticas de XP que hemos comentado en el tema 5 en las siguientes categorías

Responsable: Leire

- **Planificación y análisis de requisitos:**
  - Espacios de trabajo informativos
  - 40 horas por semana
  - Historias de usuario
  - Ciclo semanal
  - Ciclo trimestral
  - Cliente in-situ
  - Eliminar errores y su causa
  - Negociar contratos por alcance
- **Factores humanos y equipo:**
  - Sentarse juntos
  - Sensación de equipo
  - Trabajo a pleno rendimiento
  - Aflojar
  - Propiedad colectiva del código
  - Continuidad del equipo
  - Reducir el tamaño del equipo
- **Diseño:**
  - Pruebas antes de implementar
  - Diseño incremental
  - Metáforas
- **Codificación del software y entrega:**
  - Programación en parejas
  - Ten minute build
  - Integración continua
  - Refactoring
  - Estándares de programación

5.1. **Seminario 3: Tableros Kanban**

Ejercicio 6. ¿Para qué se utiliza WIP (Work in progress) en Kanban
Responsable: Clara

Limitación y visualización de la carga de trabajo del equipo. Es decir: si una tarea bloquea el desarrollo porque han surgido problemas, el avance se para hasta que se solucione y la tarea problemática salga de WIP. Teniendo un número limitado de tareas en WIP podemos permitir que el equipo completo se centre en las tareas más complejas cuando nos dan problemas.

Ejercicio 7. Lee "La Guía oficial del Método Kanban" (disponible en Prado) y comenta los principios y métricas principales de Kanban.
Responsable: Leire

Los principios de kanban se dividen en dos tipos: principios de gestión del cambio y principios de entrega del servicio.

Los principios de gestión del cambio son comunes en todas las implementaciones de Kanban y son tres:

- Comenzar con lo que haces ahora
- Acordar la búsqueda de mejora a través del cambio evolutivo
- Fomentar actos de liderazgo en todos los niveles

Gracias a estos principios, con Kanban se usa un enfoque de cambio evolutivo; se parte de una base de trabajo que ya existe y se intenta mejorar con la retroalimentación y colaboración.

Con el segundo tipo de principios, los centrados en la entrega del servicio, Kanban anima a tener un enfoque orientado al servicio, lo que lleva a ver la organización como un ser vivo que tiene una red de servicios. Cuando se quiere mejorar un servicio, hay que guiarse por una serie de principios:

- Comprender y enfocarse en cumplir las necesidades y expectativas del cliente
- Gestionar el trabajo, los trabajadores se auto organizan
- Revisar de forma periódica la red de servicios y sus políticas para buscar los puntos de mejora y poder aumentar la productividad de estos

Kanban también ofrece una serie de métricas para valorar el proceso y el producto e identificar puntos a mejorar, son tres:

- El Lead Time o tiempo de entrega: tiempo que tarda un elemento en pasar desde el principio del proceso hasta el final.
- Tasa de entrega: número de elementos de trabajo terminados por unidad de tiempo. Por ejemplo, funcionalidades acabadas por semana. Si el equipo trabaja rápido o las funcionalidades son muy pequeñas esta medida tenderá a ser más alta que en otros casos.
- El trabajo en curso o WIP: cantidad total de elementos del sistema en un momento dado. Lo ideal es que según avanza el proyecto esta cantidad vaya disminuyendo.

Todas estas medidas se usan para crear gráficos que ayuden al análisis del proceso y la implementación de Kanban.

Ejercicio 8. Comenta brevemente las diferencias entre Kanban y Scrum.
Responsable: Leire

Primero mencionar que Scrum es un marco de trabajo, una metodología; y Kanban es una técnica específica. Esto quiere decir que dentro de Scrum existe espacio para la implementación de Kanban. Realmente sus similitudes a nivel de principios e ideales son más que sus diferencias.

La mayor diferencia reside en el flujo de trabajo. Kanban cuenta con un esquema de desarrollo continuo; mientras que Scrum se organiza en Sprints y distintas etapas. En general, el flujo de trabajo de Kanban es mucho más flexible que el del Sprint de Scrum, que se rige por temporizaciones estrictas y diversas prácticas (reuniones, artefactos...).

En Scrum determinamos la carga del Sprint según ciertas métricas, normalmente la velocidad del equipo y los puntos de historia, que asignamos al sprint. En kanban, optamos por las columnas WIP en cada punto del proceso, que tienen capacidad limitada. De esta forma, el trabajo concurrente se limita de forma natural, ya que cuando surge un problema que se alarga en el tiempo, eventualmente el equipo completo estará trabajando en él hasta que se solucione.

En Kanban se destaca mucho el uso de elementos físicos visibles (tableros, pizarra, post-it...). En Scrum no se menciona su uso como parte de la metodología, pero son compatibles e incluso recomendables.