METODOLOGÍAS ÁGILES

LEAN SOFTWARE DEVELOPMENT

Autores:
Óscar Antúnez Martinaitis
Ahmed El Moukhtari Koubaa
Alejandro Ruíz Rodríguez
Fernando Valdivia del Rosal

ÍNDICE

1. INTRODUCCIÓN

En este apartado se hará una pequeña introducción sobre qué es Lean Software Development y en que se basa dicho método.

1.1. ¿QUÉ ES LEAN SOFTWARE DEVELOPMENT?

Lean Software Development (LSD) es un marco ágil o filosofía de desarrollo de productos basada en la optimización del tiempo y los recursos, la optimización del valor del producto, la eliminación de los desperdicios, la entrega de solo lo que el producto necesita y la continua mejora del proceso de desarrollo.

1.2. ¿EN QUÉ SE BASA?

Esta técnica se basa en la aplicación de los principios del Sistema de Desarrollo de Productos de Toyota. Toyota ha sido bastante exitoso generando vehículos bastante complejos e innovadores que incluían una gran cantidad de software nuevo, desarrollado en un corto plazo de tiempo y siempre entregado a tiempo. Cuando estos principios se aplican correctamente, Lean Software Development genera un software de alta calidad, desarrollado rápidamente y al menor coste posible.

2. HISTORIA

Lean Software Development tuvo su origen gracias a la industria manufacturera, que seguía un proceso de desarrollo de productos, rápido y barato, optimizando las líneas de producción para poder reducir los desperdicios y aumentar el valor del producto.

Su nombre original fue Toyota Production System, puesto que fue dicha empresa de automóviles la que empezó a aplicar un sistema de producción ágil parecido, a mediados del siglo XX. Su sistema estaba enfocado en la optimización de la producción/costes y la reducción del tiempo y los desperdicios.

Este sistema se fue extendiendo a otras industrias y finalmente se acabó denominando LEAN. Así fue en la década de los 80 cuando se empezó a utilizar por primera vez dicho término. Sin embargo, todavía no se asociaba este nombre con la producción y desarrollo de software, sino que estaba más orientado a las prácticas JIT y de control de calidad.

En 1990, este método se popularizó gracias a la publicación del libro “La máquina que cambió el mundo”, de James Womack. Este libro se inspiró en el método LEAN, conocido así desde los años 80. Los autores utilizaron el término LEAN para referirse a cualquier práctica de administración eficiente que minimice costos y pérdidas.

Finalmente, en el año 2003, Mary Poppendieck y Tom Poppendieck publicaron el famoso libro “Lean Software Development”. Fue aquí cuando esta práctica se aplicó por primera vez en la creación y desarrollo de software.

3. LOS 7 PRINCIPIOS DE LEAN SOFTWARE DEVELOPMENT

El desarrollo software siguiendo la filosofía lean puede resumirse en siete principios.

3.1. ELIMINAR DESPERDICIOS

La filosofía lean considera todo lo que no añada valor al usuario como desperdicios o deshechos, y que estos deben ser eliminados. En esta filosofía y en japonés, a los desperdicios, se le llama *muda* (無駄). Los siete tipos de *muda* definidos en la escuela de fabricación lean de Toyota son:

- Sobreproducción: consiste en hacer más de lo que es necesario o pedido por el cliente, o hacerlo antes de que sea necesario.
- Defectos: información que falta o es incorrecta y fallos de calidad, lo que resulta en la necesidad de hacer de nuevo el trabajo o en desperdicios, que pueden ocasionar costes extra a la empresa.
- Espera: cuando no se tiene la información, las instrucciones o los recursos adecuados que se necesitan para completar una tarea.
- Sobreprocesado: el uso de herramientas caras o demasiado avanzadas para llevar a cabo tareas que se pueden hacer con herramientas más baratas y sencillas.
- Movimiento: el movimiento innecesario de personas.

- Inventario: el mantenimiento del inventario añade costes sin aportar ningún valor al cliente, el inventario en exceso aumenta los plazos de entrega, retrasa la innovación y ocupa espacio.
- Transporte: todo lo que se mueva más de lo necesario sería un desperdicio (personas, productos…).

Estos *muda* aplicados al desarrollo de software son:
- Código o funcionalidades innecesarias o no deseadas.
- Empezar más de lo que se completa.
- Dedicarse a muchas cosas al mismo tiempo.
- Retraso en el proceso de desarrollo del software.
- Mala toma de requisitos.
- Requisitos que cambian constantemente o no son claros.
- Burocracia, debido a que resta velocidad al desarrollo.
- Problemas en la comunicación interna (mala o poco efectiva).
- Trabajo parcialmente hecho.
- Defectos y problemas en la calidad.
- Traspaso de tareas entre personas.
- Documentación excesiva.

3.2. AMPLIFICAR EL APRENDIZAJE

También conocido como aprender continuamente. El desarrollo de software es un proceso de aprendizaje continuo en el que el conocimiento es creado e incorporado en un producto. Es por ello por lo que el equipo debe de estar continuamente adquiriendo conocimientos nuevos y relevantes para crear un producto de innovación que satisfaga las necesidades de sus clientes.

Amplificar el aprendizaje significa incrementar la habilidad del equipo para aprender de forma rápida y efectiva sobre las necesidades del cliente, la solución y el proceso. Además, este principio trata de que documenten y retengan esos conocimientos.

El proceso de aprendizaje es acelerado con el uso de ciclos de iteración cortos, cada uno con pruebas de integración y refactorización (refactoring). También, el aumento de retroalimentación que se consigue mediante sesiones cortas de evaluación con el cliente ayudan a determinar la fase actual de desarrollo, ajustar el

esfuerzo, aprender más sobre el dominio del problema y encontrar posibles soluciones.

Para el registro y la retención de este conocimiento se pueden utilizar herramientas como la programación en parejas, las revisiones de código, la documentación y sesiones para compartir conocimiento.

3.3. TOMAR DECISIONES LO MÁS TARDE POSIBLE

Se trata de aplazar el compromiso y asegurar que se toman las decisiones clave solo cuando se está seguro de su éxito. Esto es llevado a cabo mediante el retraso de las decisiones hasta el último momento razonablemente posible, para tomar decisiones conociendo la suficiente cantidad de información y no en base a suposiciones o predicciones. Debido a que conforme avanza el progreso se obtiene una mayor cantidad de información, cuanto antes se tome una decisión hay más probabilidades de que esta sea errónea. Además, se tiene más tiempo para planificar, organizar y experimentar diferentes ideas.

Este aplazamiento del compromiso también se puede lograr mediante el desarrollo concurrente o en pequeños lotes, haciendo cambios sobre la decisión cuando sea sencillo, evitando repetir código (para que los cambios solo tengan que hacerse una vez), o usando una arquitectura acoplada libremente (uso de diseño modular, encapsulación, parámetros, etc).

3.4. ENTREGAR LO ANTES POSIBLE

Se basa en que cuanto más rápido el producto sea entregado sin defectos graves, antes se podrá recibir retroalimentación y antes se incorporará a la siguiente iteración. Es por ello por lo que cuanto más cortas son las iteraciones, mejor es el aprendizaje y la comunicación en el equipo, minimizando el riesgo y el desperdicio. La velocidad hace que se aseguren las necesidades actuales del cliente y no las que requirió ayer, lo que permite darles la oportunidad de retrasar la decisión sobre lo que realmente necesitan hasta que tengan un mejor conocimiento, por lo que complementa el principio anteriormente mencionado. La rapidez de las entregas de un producto de calidad es valorado por los clientes.

La ideología de producción just-in-time (justo a tiempo) puede aplicarse al desarrollo software. Esto se consigue mediante la presentación del resultado necesitado y dejando que el equipo se organice solo y se divida las tareas para llevar a cabo el resultado requerido para la iteración específica. Al comienzo el cliente les da el resultado que necesita (puede ser mediante pequeñas cartas o historias que estima el equipo). A partir de ahí la forma de trabajo cambia a un sistema de auto-pull (self-pulling), en el que diariamente se realiza un stand-up meeting en el que se revisa que se ha hizo el día anterior y que se hará el día actual y el siguiente (cada miembro realiza el trabajo por sí mismo), y solicita los aportes necesarios de sus compañeros o del cliente.

La implementación lean ha demostrado que es una buena práctica entregar rápido para poder ver y analizar el resultado lo antes posible. Conseguir un desarrollo software más rápido implica identificar los pasos que están retrasando al equipo y eliminar los que no contribuyan a la calidad del software que se está entregando.

3.5. EMPODERAR AL EQUIPO

La filosofía lean sigue el principio ágil de que los proyectos se construyan con individuos motivados y se confíe en ellos para la realización del trabajo, alentando al progreso, eliminando impedimentos y encontrando errores pero no dirigiendo. Los directores deben ser enseñados a escuchar a su equipo de desarrollo y dejarles tomar decisiones críticas.

Además, no se considera a la gente como recursos, de forma que se les motiva y se les da propósitos para trabajar, entre otras cosas, se les permite tener acceso a la retroalimentación del cliente, comunicarse con otros departamentos y sugerir mejoras a lo largo del ciclo de vida del producto. Además, otra forma de empoderarlos es respetarlos y reconociendo su trabajo.

3.6. CREAR LA INTEGRIDAD

Se basa en asegurar que el producto hace exactamente lo que el cliente espera que haga.

Distinguimos distintos tipos de integridad. La integridad percibida incluye cómo se anuncia, entrega, desarrolla, accede, lo intuitivo que es su uso, su precio y cómo de bien resuelve problemas. Por otro lado, la integridad conceptual significa que las componentes separadas del sistema trabajan bien juntas como un todo balanceándose entre mantenibilidad, flexibilidad, eficiencia y capacidad de respuesta. Esto puede conseguirse entendiendo el dominio del problema y resolviéndolo al mismo tiempo. La integridad de la arquitectura se puede lograr mediante la refactorización (refactoring).

Al final, la integridad debe ser verificada con el testeo, debido a que asegura que el sistema haga lo que se espera, y sea fácil de mantener, mejorar y reutilizar. Las pruebas automatizadas no deben ser un objetivo, sino un medio para conseguir la reducción de defectos. Con las pruebas evitamos añadir *muda* (desechos) al software.

3.7. OPTIMIZAR EL CONJUNTO

Los sistemas modernos no son simplemente la suma de sus partes, sino sus interacciones. Es por ello por lo que este principio se basa en considerar el sistema completo cuando se optimiza en vez de solo considerar componentes individuales, lo que permite aumentar la calidad de experiencia de usuario y un mayor valor para el cliente y el equipo.

También, la presión de los desarrolladores por realizar entregas a cualquier coste, que puede ser motivado por la industria en la que los clientes buscan constantemente nuevas funciones, soluciones y tecnología; puede llevar a cabo a que se apresuren en el proceso y no se cumplan los requisitos de calidad. Esto puede evitarse optimizando todo el proceso, entendiendo la capacidad del equipo y siendo consciente del impacto del trabajo.

Como conclusión, estos siete principios pueden ser resumidos por el eslogan “piensa grande, actúa pequeño, equivócate rápido y aprende con rapidez”.

4. PRÁCTICAS

Las prácticas usadas en el lean software development son las originales del desarrollo de software ágil con alguna ligera reformulación. Entre ellas encontramos el mapeo de flujo de valor, el desarrollo basado en conjuntos, los sistemas pull, la teoría de colas, la motivación, las mediciones o el desarrollo impulsado por pruebas. A continuación, vamos a describir las prácticas más utilizadas.

4.1. MAPEO DEL FLUJO DE VALOR – VALUE STREAM MAPPING (VSM)

El mapeo del flujo de valor es un método que permite al equipo visualizar, analizar y mejorar todos los pasos del proceso de entrega del producto. La organización efectiva es necesaria para sostener los principios lean y el mapeo del flujo de valor es una de las mejores maneras para la creación de un diagrama de flujo y la documentación de cada paso del proceso.

Su uso aporta como ventaja el dar a los equipos una indicación clara del progreso. Por el contrario, si no es llevada a cabo con el cuidado y detalle suficiente, puede inducir a error y dejar de reflejar las tareas realizadas por los desarrolladores.

4.2. DISEÑO BASADO EN CONJUNTOS – SET BASED DESIGN (SBD)

El diseño o desarrollo basado en conjuntos se basa en el mantenimiento de las opciones de diseño flexibles tanto como sea posible durante el proceso de desarrollo. Está en consonancia con el principio de retrasar las decisiones tanto como sea posible para asegurarse de que se han considerado las mejores opciones.

Su ventaja es que permite tener mucho tiempo para identificar y revisar posibles opciones, lo que permite encontrar defectos que de otro modo habrían pasado inadvertido. Se recomienda usar en entornos con innovación y de naturaleza cambiante, ya que su uso para decisiones simples puede llevar a retrasos innecesarios y a ser contraintuitivo.

4.3. SISTEMAS PULL – PULL SYSTEMS

Contribuye al principio de eliminar desperdicios. En la filosofía lean se encarga de que cuando los componentes sean consumidos se reemplacen para asegurar que el negocio solo hace los productos suficientes para satisfacer las necesidades del cliente.

Esta práctica permite a los desarrolladores tener un Work In Progress (WIP) que permita escoger tareas cuando hayan completado las suyas, limitando el número de tareas que puedan escoger. Esto previene el intercambio de tareas, caracterizado por ser ineficiente. Es por ello por lo que la ventaja de esta práctica es que permite a los desarrolladores centrarse en una tarea y a los directores el control de costes y recursos.

5. ROLES

Al contrario que otras metodologías ágiles que definen roles como el cliente o el product owner, que poseen la responsabilidad de decidir qué se necesita realizar, en el desarrollo lean no existen debido a que en este el software es un paso en el flujo de valor del producto y pretende que los desarrolladores pertenezcan a un equipo de producto mayor y que se involucren en el éxito del producto.

Los equipos de desarrollo lean están dirigidos por alguien con un rol similar a un empresario que lidera una startup. Dependiendo de la empresa puede cambiar el rol, por ejemplo, en Toyota son los ingenieros jefe, y en Microsoft los directores de programa. Estos roles se diferencian de los roles de cliente y product owner en que un ingeniero jefe posee la responsabilidad de un producto completo (hasta su éxito en el mercado) e involucra a todos en un equipo de producto multidisciplinar para llevar a cabo el proceso con éxito, sin haber ningún rol intermedio que se encargue de priorizar el trabajo para un equipo de desarrollo separado.

6. BIBLIOGRAFÍA

- M. Poppendieck, "Lean Software Development," *29th International Conference on Software Engineering (ICSE'07 Companion)*, 2007, pp. 165-166, doi: 10.1109/ICSECOMPANION.2007.46. https://ieeexplore.ieee.org/document/4222727

- Ebert, Christof, Pekka Abrahamsson, and Nilay Oza. "Lean software development." *IEEE Computer Architecture Letters* 29.05 (2012): 22-25. https://assets.vector.com/cms/content/consulting/publications/Ebert_LeanDevelopment_Intro_IEEESoftware2012.pdf

- M. Poppendieck and M. A. Cusumano, "Lean Software Development: A Tutorial," in *IEEE Software*, vol. 29, no. 5, pp. 26-32, Sept.-Oct. 2012, doi: 10.1109/MS.2012.107. https://ieeexplore.ieee.org/document/6226341

- Origen del Lean y el Software Development - Javiergarzas: https://www.javiergarzas.com/2012/10/lean-software-development-2.html

- What is Lean Software Development (LSD)? | Definition and Overview - Productplan: https://www.productplan.com/glossary/lean-software-development/

- Lean software development - Wikipedia: https://en.wikipedia.org/wiki/Lean_software_development

- Lean Software Development (LSD): Los siete principios - Netmind: https://netmind.net/es/lean-software-development-lsd-los-siete-principios/

- Guiding Principles of Lean Development - Planview: https://www.planview.com/resources/articles/lkdc-principles-lean-development/

- What Is Lean Software Development? Definition and Overview - Netguru: https://www.netguru.com/blog/lean-software-development

- Lean: Principles, Types, Lean Startup, Lean Software Development, and Examples - Medium: https://medium.com/@sofiia./lean-principles-types-lean-startup-lean-software-development-and-examples-32e15e2e0fe1