Tarea 3
Ciclo de vida del software (Relación 2)

1.- ¿Qué cuatro principios rigen el desarrollo ágil expresados en el Manifiesto Ágil?

• Individuos e interacciones sobre procesos y herramientas

Este es posiblemente el principio más importante del manifiesto. Por supuesto que los procesos ayudan al trabajo. Son una guía de operación. Las herramientas mejoran la eficiencia, pero sin personas con conocimiento técnico y actitud adecuada, no producen resultados.

• Software funcionando sobre documentación extensiva

Los documentos son soporte del software, permiten la transferencia del conocimiento, registran información histórica, y en muchas cuestiones legales o normativas son obligatorios, pero se resalta que son menos importantes que los productos que funcionan. Menos trascendentales para aportar valor al producto. Los documentos no pueden sustituir, ni pueden ofrecer la riqueza y generación de valor que se logra con la comunicación directa entre las personas y a través de la interacción con los prototipos.

• Colaboración con el cliente sobre negociación contractual

Las prácticas ágiles están especialmente indicadas para productos difíciles de definir con detalle en el principio, o que si se definiera así tendrían al final menos valor que si se van enriqueciendo con retroinformación continua durante el desarrollo. También para los casos en los que los requisitos van a ser muy inestables por la velocidad del entorno de negocio.

• Respuesta ante el cambio sobre seguir un plan

Para un modelo de desarrollo que surge de entornos inestables, que tienen como factor inherente el cambio y la evolución rápida y continua, resulta mucho más valiosa la capacidad de respuesta que la de seguimiento y aseguramiento de planes preestablecidos. Los principales valores de la gestión ágil son la anticipación y la adaptación; diferentes a los de la gestión de proyectos ortodoxa: planificación y control para evitar desviaciones sobre el plan.

2.- ¿Qué es una historia de usuario?

Es una representación de un requisito escrito en una o dos frases utilizando el lenguaje común del usuario. Las historias de usuario son utilizadas en las metodologías de desarrollo ágiles para la especificación de requisitos (acompañadas de las discusiones con los usuarios y las pruebas de validación).

3.- Haz un resumen sobre qué se entiende por Lean software y qué principios lo rigen.

Es una traducción de los principios y las prácticas de la forma de producir lean, hacia el área del desarrollo de software. Inicialmente, originado en el Sistema de Producción de Toyota y ahora, apoyado por una corriente que está surgiendo desde la comunidad Ágil. Este método ofrece todo un marco teórico sólido y basado en la experiencia, para las prácticas ágiles de gestión.

El término de desarrollo de software lean se utilizó por primera vez como título de una conferencia organizada por la iniciativa ESPRIT de la Unión Europea.

Sin mantener relación alguna, Robert “Bob” Charette, planteó un año después el concepto de desarrollo de software lean como parte de su trabajo de investigación sobre mejores formas para administrar los riesgos en proyectos de software.

4. KANBAN. Estudia las ventajas e inconvenientes de tener una pizarra web digital para la metodología Kanban.

VENTAJAS	DESVENTAJAS
Versatilidad	La falta de planificación del tiempo puede causar problemas con las fechas de entrega
Flujo de trabajo homogéneo	El trabajo debe poder dividirse en distintas fases
Más transparencia	Competencias interprofesionales
Puede aplicarse en muchas situaciones	
Mejora continuada	

5. KANBAN. Haz un resumen de la metodología Kanban e indica sus diferencias frente a SCRUM.

La metodología de trabajo kanban es sumamente importante entre los equipos de software ágiles y de DevOps de hoy en día, pero se remonta a hace más de 50 años. A finales de los años 40, Toyota empezó a optimizar sus procesos de ingeniería a partir del modelo que utilizaban los supermercados para llenar las estanterías. Los supermercados almacenan únicamente los productos suficientes para suplir la demanda del cliente, una práctica que optimiza el flujo entre el supermercado y el cliente.

6. SCRUM. Explica cómo funciona Scrum.

En Scrum un proyecto se ejecuta en ciclos temporales cortos y de duración fija (iteraciones que normalmente son de 2 semanas, aunque en algunos equipos son de 3 y hasta 4 semanas, límite máximo de feedback de producto real y reflexión). Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite.

diagrama-proceso-scrum

El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto. En esta lista el cliente (Product Owner) prioriza los objetivos balanceando el valor que le aportan respecto a su coste (que el equipo estima considerando la Definición de Hecho) y quedan repartidos en iteraciones y entregas.

7. SCRUM. Define los siguientes términos:

Product backlog. Es una herramienta esencial para la gestión de proyectos que consiste en la elaboración de un listado de todas aquellas tareas que queremos realizar durante el desarrollo de un proyecto con el objetivo de que estas sean visibles para todo el equipo

Sprint backlog. Es la suma del Objetivo del Sprint, los elementos del Product Backlog elegidos para el Sprint, más un plan de acción de cómo crear el Incremento de Producto. Es uno de los 3 artefactos de Scrum y se construye durante el evento del Sprint Planning. Es un plan realizado por y para los Developers.
El equipo generalmente divide el trabajo en elementos llamados Sprint Backlog Ítems (SBI). Estos elementos pueden representar tareas que el equipo debe completar, bloques de construcción intermedios que se combinan en una entrega, o cualquier otra unidad de trabajo que ayude al equipo a comprender cómo lograr el Sprint Goal dentro del Sprint.

8. SCRUM. En la terminología Scrum qué términos se utilizan como sinónimo de:

Jefe de proyecto = Scrum Master
Cliente = Product Owner
Equipo de desarrollo = Development Team

9. SCRUM. Haz un resumen de los requisitos para poder utilizar Scrum.

Los siguientes puntos son de especial importancia para la implantación de una gestión ágil de proyectos como Scrum:

- Cultura de empresa basada en trabajo en equipo, delegación, creatividad y mejora continua.
- Compromiso del cliente en la dirección de los resultados del proyecto, gestión del ROI y disponibilidad para poder colaborar.
- Compromiso de la Dirección de la organización para resolver problemas endémicos y realizar cambios organizativos, formando equipos autogestionados y multidisciplinares y fomentando una cultura de gestión basada en la colaboración y en la facilitación llevada a cabo por líderes al servicio del equipo.
- Compromiso conjunto y colaboración de los miembros del equipo.
- Relación entre proveedor y cliente basada en ganar-ganar, colaboración y transparencia.
- Facilidad para realizar cambios en el proyecto.
- Tamaño de cada equipo entre 5 y 9 personas
- Equipo trabajando en un mismo espacio común para maximizar la comunicación.
- Dedicación del equipo a tiempo completo.
- Estabilidad de los miembros del equipo

10. Explica los 5 valores de la Programación Extrema.

Los valores originales de la programación extrema son: simplicidad, comunicación, retroalimentación (feedback) y coraje. Un quinto valor, respeto, fue añadido en la segunda edición de Extreme Programming Explained. Los cinco valores se detallan a continuación:

1. Simplicidad. Es la base de la programación extrema. Se simplifica el diseño para agilizar el desarrollo y facilitar el mantenimiento. Un diseño complejo del código junto a sucesivas modificaciones por parte de diferentes desarrolladores hacen que la complejidad aumente exponencialmente. También se aplica la simplicidad en la documentación, de esta manera el código debe comentarse en su justa medida, intentando eso sí que el código esté autodocumentado. Para ello se deben elegir adecuadamente los nombres de las variables, métodos y clases.

2. Comunicación. Se realiza de diferentes formas. Para los programadores el código comunica mejor cuanto más simple sea. Si el código es complejo hay que esforzarse para hacerlo legible. El código autodocumentado es más fiable que los comentarios ya que estos últimos pronto quedan desfasados con el código a medida que es modificado. Debe comentarse sólo aquello que no va a variar, por ejemplo el objetivo de una clase o la funcionalidad de un método.

3. Retroalimentación (feedback). Al estar el cliente integrado en el proyecto, su opinión sobre el estado del proyecto se conoce en tiempo real.Al realizarse ciclos muy cortos tras los cuales se muestran resultados, se minimiza el tener que rehacer partes que no cumplen con los requisitos y ayuda a los programadores a centrarse en lo que es más importante.

4. Coraje o valentía. Muchas de las prácticas implican valentía. Una de ellas es siempre diseñar y programar para hoy y no para mañana. Esto es un esfuerzo para evitar empantanarse en el diseño y requerir demasiado tiempo y trabajo para implementar el resto del proyecto. La valentía le permite a los desarrolladores que se sientan cómodos con reconstruir su código cuando sea necesario. Esto significa revisar el sistema existente y modificarlo si con ello los cambios futuros se implementarán más fácilmente.

5. Respeto. Se manifiesta de varias formas. Los miembros del equipo se respetan los unos a los otros, porque los programadores no pueden realizar cambios que hacen que las pruebas existentes fallen o que demore el trabajo de sus compañeros.

11. ¿Cuáles son las características distintivas de XP frente a otras metodologías ágiles? Explícalas.

- El juego de la planificación (the planning game). Es un permanente diálogo entre las partes empresarial (deseable) y técnica (posible).
- Pequeñas entregas (small releases). Cada versión debe de ser tan pequeña como fuera posible, conteniendo los requisitos de negocios más importantes, las versiones tiene que tener sentido como un todo, me explico no puedes implementar media característica y lanzar la versión.
- Diseño sencillo (simple design). Cuando implementamos nuevas características en nuestros programas nos planteamos la manera de hacerlo lo mas simple posible, después de implementar esta característica, nos preguntamos como hacer el programa más simple sin perder funcionalidad, este proceso se le denomina recodificar o refactorizar (refactoring)..

- Pruebas (testing). No debe existir ninguna característica en el programa que no haya sido probada, los programadores escriben pruebas para chequear el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. El resultado es un programa más seguro que conforme pasa el tiempo es capaz de aceptar nuevos cambios.
- Refactorización (refactoring). Cuando implementamos nuevas características en nuestros programas nos planteamos la manera de hacerlo lo más simple posible, después de implementar esta característica, nos preguntamos cómo hacer el programa más simple sin perder funcionalidad, este proceso se le denomina recodificar o refactorizar (refactoring).
- Programación por parejas (pair programming). Todo el código de producción lo escriben dos personas frente al ordenador, con un sólo ratón y un sólo teclado. Cada miembro de la pareja juega su papel: uno codifica en el ordenador y piensa la mejor manera de hacerlo, el otro piensa mas estratégicamente, ¿Va a funcionar?, ¿Puede haber pruebas donde no funcione?, ¿Hay forma de simplificar el sistema global para que el problema desaparezca?
- El emparejamiento es dinámico, puede estar emparejado por la mañana con una persona y por la tarde con otra, si tienes un trabajo sobre un área que no conoces muy bien puedes emparejarte con otra persona que sí conozca ese área. Cualquier miembro del equipo se puede emparejar con cualquiera.
- Propiedad colectiva (collective ownership). Cualquiera que crea que puede aportar valor al código en cualquier parcela puede hacerlo, ningún miembro del equipo es propietario del código. Si alguien quiere hacer cambios en el código puede hacerlo.
- Integración continua (continuous integration). El código se debe integrar como mínimo una vez al día, y realizar las pruebas sobre la totalidad del sistema. Una pareja de programadores se encargará de integrar todo el código en una máquina y realizar todas las pruebas hasta que estas funcionen al 100%.
- 40 horas semanales (40-hour week). Si queremos estar frescos y motivados cada mañana y cansado y satisfecho cada noche. El viernes quiero estar cansado y satisfecho para sentir que tengo dos días para pensar en algo distinto y volver el lunes lleno de pasión e ideas. Esto requiere que trabajemos 40 horas a la semana, mucha gente no puede estar más de 35 horas concentrada a la semana, otros pueden llegar hasta 45 pero ninguno puede llegar a 60 horas durante varias semanas y aun seguir fresco, creativo y confiado. Las horas extras son síntoma de serios problemas en el proyecto, la regla de XP dice nunca 2 semanas seguidas realizando horas extras.
- Cliente en casa (on-site customer). Un cliente real debe sentarse con el equipo de programadores, estar disponible para responder a sus preguntas, resolver discusiones y fijar las prioridades.
- Estándares de codificación (coding standards). Si los programadores van a estar tocando partes distintas del sistema, intercambiando compañeros, haciendo refactoring, debemos de establecer un estándar de codificación aceptado e implantado por todo el equipo.