UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO.

ACADEMIA DE INFORMÁTICA

PLANTEL CUATEPEC

PROFESOR:

GERARDO HERNANDEZ

ALUMNA:

CABELLO LUNA MARÍA SARAI

LICENCIATURA:

INGENIERÍA DE SOFTWARE

METERIA:

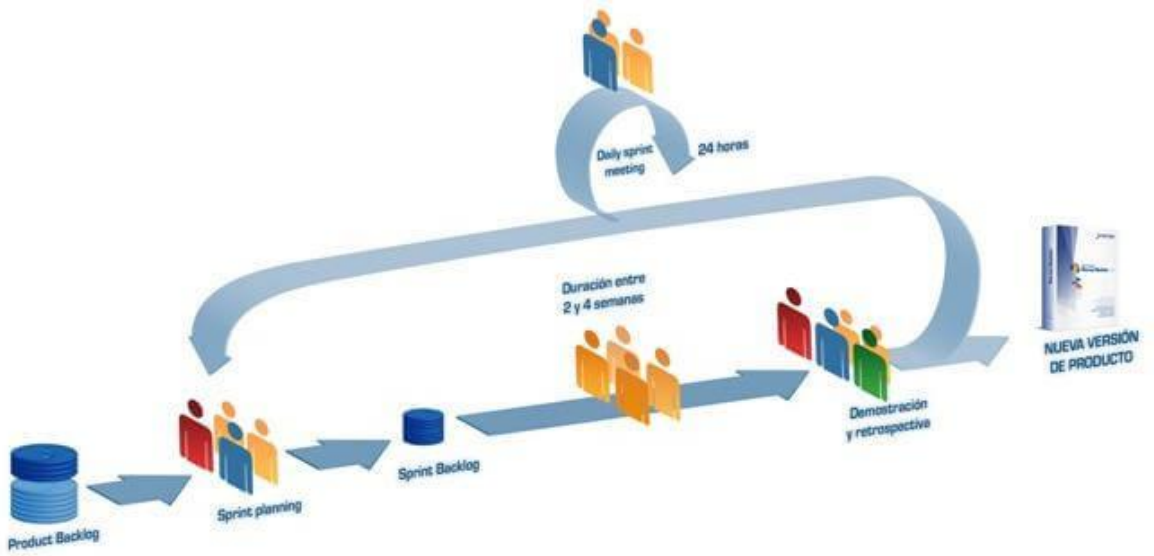
INTRODUCCION A LA INGENIERÍA DE SOFTWARE

TRABAJO:

METODOLOGÍAS ÁGILES

SCRUM

Scrum es una metodología ágil y flexible para gestionar el desarrollo de software, cuyo principal objetivo es maximizar el retorno de la inversión para su empresa (ROI). Se basa en construir primero la funcionalidad de mayor valor para el cliente y en los principios de inspección continua, adaptación, auto-gestión e innovación.



Esta metódica de trabajo promueve la innovación, motivación y compromiso del equipo que forma parte del proyecto, por lo que los profesionales encuentran un ámbito propicio para desarrollar sus capacidades.

BENEFICIOS

- **Cumplimento de expectativas:** El cliente establece sus expectativas indicando el valor que le aporta cada requisito / **historia** del proyecto, el equipo los estima y con esta información el **Product Owner** establece su prioridad. De manera regular, en las demos de Sprint el **Product Owner** comprueba que efectivamente los requisitos se han cumplido y transmite se feedback al equipo.
- **Flexibilidad a cambios:** Alta capacidad de reacción ante los cambios de requerimientos generados por necesidades del cliente o evoluciones del mercado. La metodología está diseñada para adaptarse a los cambios de requerimientos que conllevan los proyectos complejos.

- **Reducción del Time to Market:** El cliente puede empezar a utilizar las funcionalidades más importantes del proyecto antes de que esté finalizado por completo.

- **Mayor calidad del software:** La metódica de trabajo y la necesidad de obtener una versión funcional después de cada iteración, ayuda a la obtención de un software de calidad superior.
- **Mayor productividad:** Se consigue entre otras razones, gracias a la eliminación de la burocracia y a la motivación del equipo que proporciona el hecho de que sean autónomos para organizarse.
- **Maximiza el retorno de la inversión (ROI):** Producción de software únicamente con las prestaciones que aportan mayor valor de negocio gracias a la priorización por retorno de inversión.
- **Predicciones de tiempos:** Mediante esta metodología se conoce la velocidad media del equipo por sprint (los llamados puntos historia), con lo que consecuentemente, es posible estimar fácilmente para cuando se dispondrá de una determinada funcionalidad que todavía está en el Backlog.
- **Reducción de riesgos:** El hecho de llevar a cabo las funcionalidades de más valor en primer lugar y de conocer la velocidad con que el equipo avanza en el proyecto, permite despejar riesgos eficazmente de manera anticipada.

EXTREME PROGRAMMING (XP)

La metodología XP o Programación Extrema es una metodología ágil y flexible utilizada para la gestión de proyectos.

Extreme Programming se centra en potenciar las relaciones interpersonales del equipo de desarrollo como clave del éxito mediante el trabajo en equipo, el aprendizaje continuo y el buen clima de trabajo.

Esta metodología pone el énfasis en la retroalimentación continua entre cliente y el equipo de desarrollo y es idónea para proyectos con requisitos imprecisos y muy cambiantes.

CARACTERÍSTICAS

- *Se considera al equipo de proyecto como el principal factor de éxito del proyecto*
- *Software que funciona por encima de una buena documentación.*
- Interacción constante entre el cliente y el equipo de desarrollo.
- Planificación flexible y abierta.
- *Rápida respuesta a cambios.*

ROLES

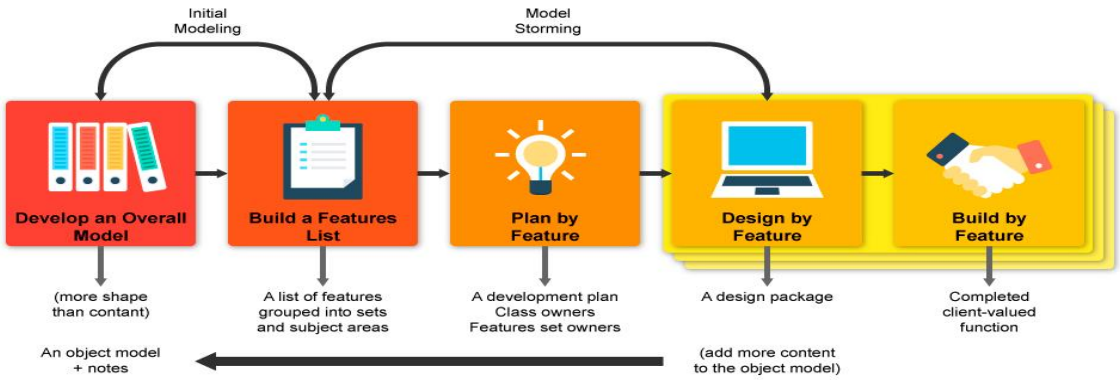
- **Cliente**: responsable de definir y conducir el proyecto, así como sus objetivos.
- **Programadores**: estiman tiempos de desarrollo de cada actividad y programan el proyecto.
- **Tester**: Encargado de Pruebas.
- **Tracker**: Encargado de Seguimiento.
- **Coach**: Entrenador. Su papel es guiar y orientar al equipo.
- **Big Boss**: Gestor del proyecto, gerente del proyecto, debe tener una idea general del proyecto y estar familiarizado con su estado.

FEATURE-DRIVEN DEVELOPMENT (FDD)

Es una metodología ágil diseñada para el desarrollo de software, basada en la calidad y el monitoreo constante del proyecto. Fue desarrollada por Jeff De Luca y Peter Coad a mediados de los años 90. Esta metodología se enfoca en iteraciones cortas, que permiten

entregas tangibles del producto en un periodo corto de tiempo, de como máximo dos semanas.

El equipo de trabajo está estructurado en jerarquías, siempre debe haber un jefe de proyecto, y aunque es un proceso considerado ligero también incluye documentación (la mínima necesaria para que algún nuevo integrante pueda entender el desarrollo de inmediato).



CARACTERÍSTICAS

- Se preocupa por la calidad, por lo que incluye un monitoreo constante del proyecto.
- Ayuda a contrarrestar situaciones como el exceso en el presupuesto, fallas en el programa o el hecho de entregar menos de lo deseado.
- Propone tener etapas de cierre cada dos semanas. Se obtienen resultados periódicos y tangibles.
- Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que el cliente y la dirección de la empresa pueden ver y monitoriar.
- Define claramente entregas tangibles y formas de evaluación del progreso del proyecto.
- No hace énfasis en la obtención de los requerimientos sino en como se realizan las fases de diseño y construcción.

VENTAJAS

- El equipo de desarrollo no malgasta el tiempo y dinero del cliente desarrollando soluciones innecesariamente generales y complejas que en realidad no son un requisito del cliente.
- Cada componente del producto final ha sido probado y satisface los requerimientos.
- Rápida respuesta a cambios de requisitos a lo largo del desarrollo.
- Entrega continua y en plazos cortos de software funcional.
- Trabajo conjunto entre el cliente y el equipo de desarrollo.
- Minimiza los costos frente a cambios.
- Importancia de la simplicidad, al eliminar el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño.
- Mejora continua de los procesos y el equipo de desarrollo.
- Evita malentendidos de requerimientos entre el cliente y el equipo.

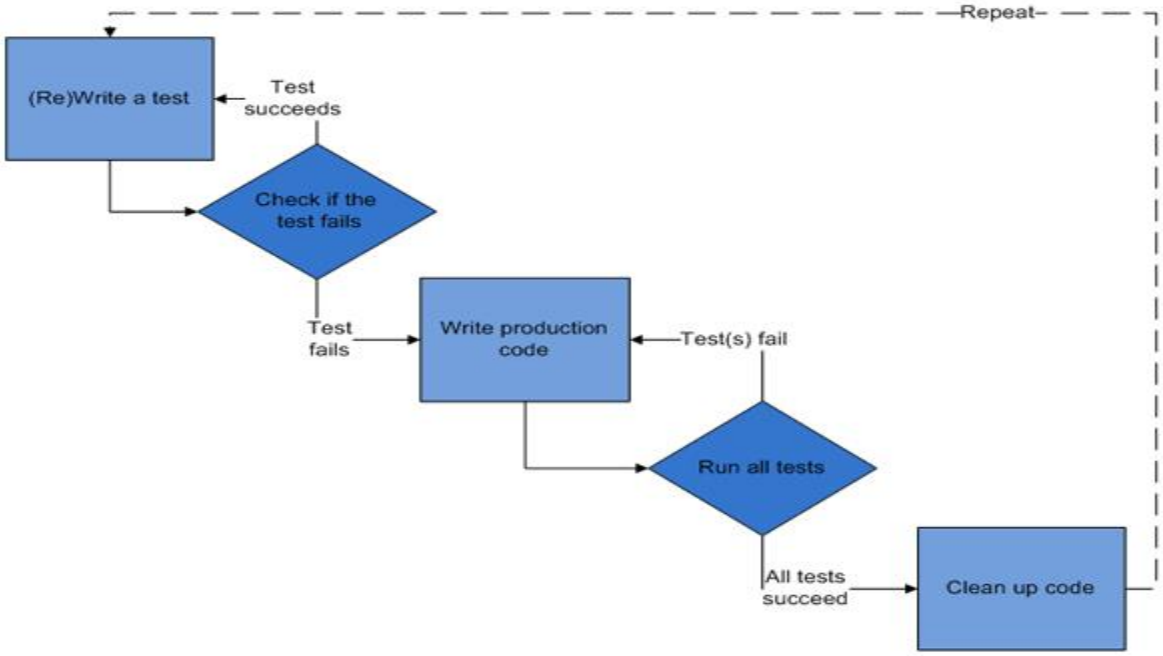
DESVENTAJAS

- Falta de documentación del diseño. El código no puede tomarse como una documentación. En sistemas de tamaño grande se necesitar leer los cientos o miles de páginas del listado de código fuente.
- Problemas derivados de la comunicación oral. Este tipo de comunicación resulta difícil de preservar cuando pasa el tiempo y está sujeta a muchas ambigüedades.

TEST-DRIVEN DEVELOPMENT (TDD)

Es una práctica de programación que consiste en escribir primero las pruebas (generalmente unitarias), después escribir el código fuente que pase la prueba satisfactoriamente, y por último refactorizar el código escrito. Con esta práctica se consigue un código más robusto, más seguro, más mantenible y una mayor rapidez en el desarrollo.

Para que funcione el desarrollo guiado por pruebas, el sistema que se programa tiene que ser lo suficientemente flexible como para permitir que sea probado automáticamente. Cada prueba será suficientemente pequeña como para que permita determinar unívocamente si el código probado pasa o no la verificación que ésta le impone. El diseño se ve favorecido ya que se evita el indeseado "sobre diseño" de las aplicaciones y se logran interfaces más claras y un código más cohesivo.



CICLO DE DESARROLLO CONDUCIDO POR PRUEBAS

1. **Elegir un requisito:** Se elige de una lista el requisito que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácilmente implementable.
2. **Escribir una prueba:** Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces.
3. **Verificar que la prueba falla:** Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.
4. **Escribir la implementación:** Escribir el código más sencillo que haga que la prueba funcione. Se usa la expresión "Déjelo simple" ("Keep It Simple, Stupid!"), conocida como principio de Kiss.
5. **Ejecutar las pruebas automatizadas:** Verificar si todo el conjunto de pruebas funciona correctamente.

6. **Eliminación de duplicación:** El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado. Se hace un pequeño cambio cada vez y luego se corren las pruebas hasta que funcionen.
7. **Actualización de la lista de requisitos:** Se actualiza la lista de requisitos tachando el requisito implementado. Asimismo, se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requisitos de diseño (P. ej que una funcionalidad esté desacoplada de otra).
8. Tener un único repositorio universal de pruebas facilita complementar TDD con otra práctica recomendada por los procesos agiles de desarrollo, la "Integración Continua". Integrar continuamente nuestro trabajo con el del resto del equipo de desarrollo permite ejecutar toda batería de pruebas y así descubrir si nuestra última versión es compatible con el resto del sistema. Es recomendable y menos costoso corregir pequeños problemas cada pocas horas que enfrentarse a problemas enormes cerca de la fecha de entrega fijada.