Universidad Intercontinental de la Empresa

Asignatura

Arquitectura del Software 

Profesor

Yago Fontenla Seco

{*yago.fontenla1@uie.edu*}

¿Qué son las metodologías Ágiles?

Los métodos ágiles se apoyan universalmente en el enfoque incremental para la especificación, el desarrollo y la entrega del software. Son más adecuados para el diseño de aplicaciones en que los requerimientos del sistema cambian, por lo general, rápidamente durante el proceso de desarrollo. Tienen la intención de entregar con prontitud el software operativo a los clientes, quienes entonces propondrán requerimientos nuevos y variados para incluir en posteriores iteraciones del sistema. Se dirigen a simplificar el proceso burocrático al evitar trabajo con valor dudoso a largo plazo, y a eliminar documentación que quizá nunca se emplee.

Enfoques o metodologías destacados

- **Programación Extrema**
- **Scrum**
- Crystal
- Desarrollo de software adaptativo
- Desarrollo dirigido por características

Contexto

¿Por qué surgen las metodologías ágiles?

Las empresas de hoy se encuentran en un entorno global caracterizado por el cambio continuo. Deben responder rápidamente a nuevas oportunidades de mercado y a la competencia que introduce productos y servicios innovadores.

Como resultado, el desarrollo de software ágil se convierte en una herramienta fundamental para responder a estas demandas de forma rápida, permitiendo a las organizaciones mantener su relevancia y competitividad.

Aspectos clave:

- Las empresas actuales operan en un entorno global en constante cambio.
- La rapidez es esencial para aprovechar oportunidades y responder a la competencia.
- El software se desarrolla rápidamente para adaptarse a estas necesidades.

Desafíos de los métodos tradicionales

Los enfoques tradicionales de desarrollo de software, como el modelo en cascada, son menos flexibles y están diseñados para seguir una secuencia estructurada de especificación, diseño, implementación y prueba. Este enfoque es ineficiente en un entorno donde los requisitos del negocio cambian rápidamente. Cuando estos requisitos se modifican, el software debe reelaborarse, lo que alarga los tiempos de entrega y aumenta el riesgo de que el software ya no cumpla con las necesidades del mercado al momento de ser entregado.

Limitaciones clave:

- Los métodos basados en especificaciones completas no se ajustan a la rapidez requerida.
- La naturaleza cambiante de los requisitos complica el desarrollo tradicional.
- Los requisitos suelen cambiar de forma impredecible, haciendo obsoleto el software antes de su entrega.

Orígen y evolución

Desde la década de 1980, se reconoció la necesidad de procesos de desarrollo rápidos y adaptativos. IBM fue pionera al introducir el desarrollo incremental. En la década de 1990, surgen formalmente las metodologías ágiles como DSDM, Scrum y la Programación Extrema (XP), diseñadas para responder a los cambios rápidos en los requisitos y centrarse en la entrega continua de software funcional en plazos cortos. Estos métodos priorizan la interacción directa con los clientes y la respuesta ágil ante los cambios.

Aspectos clave:

- Se reconoció la necesidad de procesos rápidos desde la década de 1980.
- En los 90s surgen enfoques ágiles como: DSDM, Scrum y Programación Extrema (XP)
- Los métodos ágiles se basan en desarrollos incrementales y entregas rápidas.

Características clave

Los procesos de desarrollo del software rápido se diseñan para producir rápidamente un software útil. El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema.

Aun cuando existen muchos enfoques para el desarrollo de software rápido, comparten algunas características fundamentales:

- **Especificación, diseño e implementación entrelazados:** Enfoque en las características clave, minimizando documentación.
- **Desarrollo incremental:** Involucra a usuarios en la evaluación de versiones.
- **Interfaces de usuario interactivas:** Rapidez en la creación de interfaces intuitivas y funcionales.

Principios

Aunque todos esos métodos ágiles se basan en la noción del desarrollo y la entrega incrementales, proponen diferentes procesos para lograrlo. Sin embargo, comparten una serie de **principios**, según el **manifiesto ágil** y, por ende, **tienen mucho en común**.

Principio	Descripción
Participación del cliente	Los clientes deben intervenir estrechamente durante el proceso de desarrollo. Su función consiste en ofrecer y priorizar nuevos requerimientos del sistema y evaluar las iteraciones del mismo.
Entrega incremental	El software se desarrolla en incrementos y el cliente especifica los requerimientos que se van a incluir en cada incremento.
Personas, no procesos	Tienen que reconocerse y aprovecharse las habilidades del equipo de desarrollo. Debe permitirse a los miembros del equipo desarrollar sus propias formas de trabajar sin procesos establecidos.
Adoptar el cambio	Esperar a que cambien los requerimientos del sistema y, de este modo, diseñar el sistema para adaptar dichos cambios.
Mantener simplicidad	Enfocarse en la simplicidad tanto en el software a desarrollar como en el proceso de desarrollo. Siempre que sea posible, trabajar de manera activa para eliminar la complejidad del sistema.

Dificultades

Si bien las metodologías ágiles ofrecen beneficios importantes, como una mayor adaptabilidad a los cambios y una entrega rápida de software funcional, su implementación no está exenta de desafíos. En la práctica, las organizaciones y equipos a menudo enfrentan dificultades para adherirse a los principios ágiles.

Involucramiento del Cliente

Desafío: Los métodos ágiles requieren una participación activa del cliente, quien debe colaborar de manera continua con el equipo de desarrollo.

Problema: A menudo, los representantes del cliente tienen otras responsabilidades y no pueden dedicar el tiempo necesario. Además, puede que no representen todos los intereses y necesidades de los usuarios finales.

Personalidad y Participación del Equipo

Desafío: Los métodos ágiles se basan en la colaboración y comunicación constante.

Problema: Algunos miembros del equipo pueden no tener la personalidad adecuada para la participación intensa, afectando la interacción y efectividad del equipo.

Priorizar Cambios

Desafío: Los cambios en los requisitos son comunes y deben priorizarse.

Problema: En sistemas con múltiples stakeholders, cada uno puede tener prioridades diferentes, dificultando el consenso sobre qué cambios son más importantes.

Dificultades

Si bien las metodologías ágiles ofrecen beneficios importantes, como una mayor adaptabilidad a los cambios y una entrega rápida de software funcional, su implementación no está exenta de desafíos. En la práctica, las organizaciones y equipos a menudo enfrentan dificultades para adherirse a los principios ágiles.

Mantener la Simplicidad

Desafío: Los métodos ágiles promueven soluciones simples y adaptables.

Problema: Bajo la presión de plazos ajustados, el equipo puede no tener el tiempo necesario para simplificar adecuadamente el diseño o el código, comprometiendo la calidad.

Cultura Organizacional

Desafío: Los métodos ágiles requieren flexibilidad y procesos informales.

Problema: En organizaciones grandes y estructuradas, la transición hacia una cultura ágil puede ser difícil y lenta, ya que sus procesos tradicionales suelen estar profundamente arraigados y son resistidos al cambio.

Dificultades

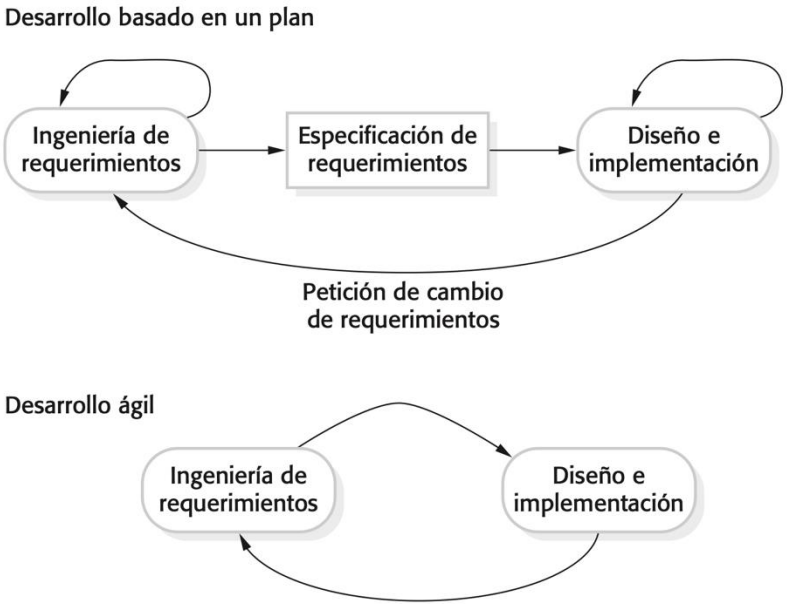
Otro problema que no es técnico, es decir, que consiste en un problema general con el desarrollo y la entrega incremental, ocurre cuando el cliente del sistema acude a una organización externa para el desarrollo del sistema. Por lo general, el documento de requerimientos del software forma parte del contrato entre el cliente y el proveedor. Como la especificación incremental es inherente en los métodos ágiles, quizá sea difícil elaborar contratos para este tipo de desarrollo.

Como resultado, los métodos ágiles deben apoyarse en contratos, en los cuales el cliente pague por el tiempo requerido para el desarrollo del sistema, en vez de hacerlo por el desarrollo de un conjunto específico de requerimientos. En tanto todo marche bien, esto beneficia tanto al cliente como al desarrollador. No obstante, cuando surgen problemas, sería difícil discutir acerca de quién es culpable y quién debería pagar por el tiempo y los recursos adicionales requeridos para solucionar las dificultades.

Diferencias principales

Desarrollo Basado en un Plan: Sigue un enfoque secuencial con etapas definidas (requisitos, diseño, implementación, pruebas) y documentación formal entre cada fase. Es ideal para proyectos donde los requisitos son estables y se requiere un diseño detallado desde el principio.

Desarrollo Ágil: Se centra en la flexibilidad y en la entrega rápida mediante iteraciones continuas. Las actividades como el diseño, la implementación, la recolección de requisitos y las pruebas se integran y evolucionan conjuntamente, lo que permite responder rápidamente a los cambios.



Cuestiones clave

Conclusión: Cada proyecto tiene necesidades específicas; por lo tanto, es crucial evaluar estas preguntas para seleccionar el enfoque que mejor se adapte a los objetivos del proyecto y a las capacidades de la organización.

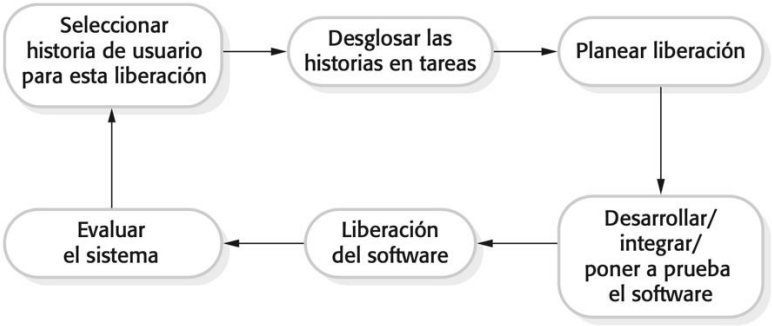
- **Especificación Detallada:** ¿Es esencial tener una especificación completa y detallada desde el inicio?
- **Entrega Incremental:** ¿Es viable y útil entregar el software en versiones incrementales para obtener retroalimentación temprana?
- **Tamaño del Proyecto:** ¿Qué tan grande es el sistema a desarrollar? El desarrollo ágil es más adecuado para equipos pequeños y proyectos acotados.
- **Tipo de Sistema:** ¿Requiere análisis detallado antes de la implementación, como en sistemas críticos de tiempo real?
- **Duración del Proyecto:** ¿Se espera un largo ciclo de vida? Los proyectos de larga duración pueden requerir más documentación para el mantenimiento futuro.
- **Tecnologías Disponibles:** ¿Existen herramientas adecuadas para soportar el diseño ágil y visualizar la evolución del sistema?
- **Organización del Equipo:** ¿El equipo está distribuido o es local? Equipos distribuidos pueden necesitar más documentación para comunicarse.
- **Cultura Organizacional:** ¿Es la organización receptiva a procesos ágiles o se rige por una cultura estructurada de desarrollo?
- **Habilidades del Equipo:** ¿Los diseñadores y programadores cuentan con la experiencia y habilidades para un enfoque ágil?
- **Regulaciones Externas:** ¿El proyecto está sujeto a regulaciones que exigen documentación formal y procesos de validación rigurosos?

Programación Extrema (XP)

La **programación extrema (XP)** es quizás el método ágil mejor conocido y más ampliamente usado. El nombre procede de que el enfoque se desarrolló llevando a niveles “extremos” las prácticas reconocidas, como el desarrollo iterativo.

En la programación extrema, los requerimientos se expresan como **escenarios** llamados **historias de usuario**, que se implementan directamente como una serie de tareas.

Los programadores **trabajan en pares** y antes de escribir el código desarrollan **pruebas** para cada tarea. Todas las pruebas deben ejecutarse con éxito una vez que el nuevo código se integre en el sistema.



Proceso XP para producir un incremento de un sistema.

Principios de la programación extrema

La programación extrema incluye una serie de practicas que reflejan los **principios de los métodos ágiles**:

1. El **desarrollo incremental** se apoya en pequeñas liberaciones del sistema. Los requerimientos se fundamentan en historias del cliente usados como base para decidir qué funcionalidad debe incluirse en un incremento.
2. La **inclusión del cliente** se apoya a través de un enlace continuo. El representante del cliente participa en el desarrollo y es responsable de definir las pruebas de aceptación.
3. Las personas, no los procesos, se basan en la **programación en pares**, en la propiedad colectiva del código y en un proceso de desarrollo sostenible.
4. El **cambio** se acepta mediante liberaciones regulares del sistema a los clientes, desarrollo de primera prueba, refactorización del código e integración continua de nueva funcionalidad.
5. Mantener la **simplicidad** se logra mediante la refactorización constante, que mejora la calidad del código, y con el uso de diseños simples que no anticipan innecesariamente futuros cambios.

Principio o práctica	Descripción
Planeación incremental	Los requerimientos se registran en tarjetas de historia (*story cards*) y las historias que se van a incluir en una liberación se determinan por el tiempo disponible y la prioridad relativa. Los desarrolladores desglosan dichas historias en "tareas" de desarrollo. Vea las figuras 3.5 y 3.6.
Liberaciones pequeñas	Al principio se desarrolla el conjunto mínimo de funcionalidad útil, que ofrece valor para el negocio. Las liberaciones del sistema son frecuentes y agregan incrementalmente funcionalidad a la primera liberación.
Diseño simple	Se realiza un diseño suficiente para cubrir sólo aquellos requerimientos actuales.
Desarrollo de la primera prueba	Se usa un marco de referencia de prueba de unidad automatizada al escribir las pruebas para una nueva pieza de funcionalidad, antes de que esta última se implemente.
Refactorización	Se espera que todos los desarrolladores refactoricen de manera continua el código y, tan pronto como sea posible, se encuentren mejoras de éste. Lo anterior conserva el código simple y mantenible.
Programación en pares	Los desarrolladores trabajan en pares, y cada uno comprueba el trabajo del otro; además, ofrecen apoyo para que se realice siempre un buen trabajo.
Propiedad colectiva	Los desarrolladores en pares laboran en todas las áreas del sistema, de manera que no se desarrollan islas de experiencia, ya que todos los desarrolladores se responsabilizan por todo el código. Cualquiera puede cambiar cualquier función.
Integración continua	Tan pronto como esté completa una tarea, se integra en todo el sistema. Después de tal integración, deben aprobarse todas las pruebas de unidad en el sistema.
Ritmo sustentable	Grandes cantidades de tiempo extra no se consideran aceptables, pues el efecto neto de este tiempo libre con frecuencia es reducir la calidad del código y la productividad de término medio.
Cliente en sitio	Un representante del usuario final del sistema (el cliente) tiene que disponer de tiempo completo para formar parte del equipo XP. En un proceso de programación extrema, el cliente es miembro del equipo de desarrollo y responsable de llevar los requerimientos del sistema al grupo para su implementación.

Historias de usuario

¿Qué son las Historias de Usuario?

Las historias de usuario son **descripciones breves de funcionalidades** que el usuario necesita en el sistema. Se enfocan en los resultados o funcionalidades que se requieren, no en cómo implementarlas. Se crean en colaboración entre el equipo de desarrollo y el cliente o product owner y se enfocan en expresar el valor que la funcionalidad aportará al usuario.

Gestión: Las historias se agrupan y priorizan en función del valor para el negocio y del tiempo de desarrollo estimado. **Se dividen en tareas** manejables que se asignan al equipo. El cliente puede ajustar o añadir historias de usuario durante el desarrollo, permitiendo la adaptación continua a los cambios en los requisitos.

Criterios de Buena Historia: Las historias de usuario deben ser INDependientes, Negociables, Valorables, Estimables, Pequeñas y Testables (INVEST).

Relación con casos de uso: Mientras que los casos de uso describen interacciones detalladas y pueden incluir flujos alternativos, las historias de usuario son más breves y orientadas al usuario final. Sirven como un paso inicial para desarrollar casos de uso detallados en proyectos más formales.

Historias de usuario

Prescripción de medicamentos

Kate es una médica que quiere prescribir fármacos a un paciente que se atiende en una clínica. El archivo del paciente ya se desplegó en su computadora, de manera que da clic en el campo del medicamento y luego puede seleccionar "medicamento actual", "medicamento nuevo" o "formulario".

Si selecciona "medicamento actual", el sistema le pide comprobar la dosis. Si quiere cambiar la dosis, ingresa la dosis y luego confirma la prescripción.

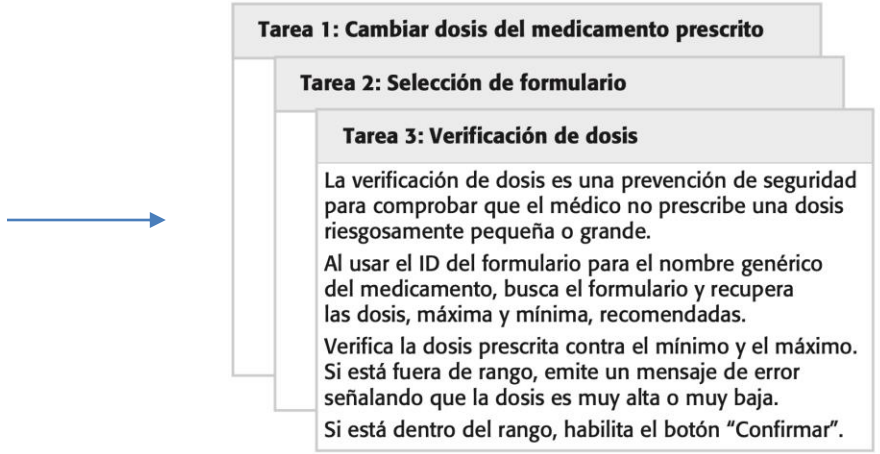
Si elige "medicamento nuevo", el sistema supone que Kate sabe cuál medicamento prescribir. Ella teclea las primeras letras del nombre del medicamento. El sistema muestra una lista de medicamentos posibles cuyo nombre inicia con dichas letras. Posteriormente elige el fármaco requerido y el sistema responde solicitándole que verifique que el medicamento seleccionado sea el correcto. Ella ingresa la dosis y luego confirma la prescripción.

Si Kate elige "formulario", el sistema muestra un recuadro de búsqueda para el formulario aprobado. Entonces busca el medicamento requerido. Ella selecciona un medicamento y el sistema le pide comprobar que éste sea el correcto. Luego ingresa la dosis y confirma la prescripción.

El sistema siempre verifica que la dosis esté dentro del rango aprobado. Si no es así, le pide a Kate que la modifique.

Después de que ella confirma la prescripción, se desplegará para su verificación. Kate hace clic o en "OK" o en "Cambiar". Si hace clic en "OK", la prescripción se registra en la base de datos de auditoría. Si hace clic en "Cambiar", reingresa al proceso de "prescripción de medicamento".

Historia de usuario.



Historia descompuesta en tareas.

Pruebas en XP

En la programación extrema (XP), las pruebas son un **componente fundamental** del proceso de desarrollo. XP se basa en la prueba continua y temprana del software, lo que permite detectar errores desde las primeras fases y asegura que el sistema cumpla con los requisitos funcionales. Las pruebas no son solo una etapa final, sino que están integradas en todo el ciclo de desarrollo, enfocándose en las siguientes prácticas clave:

- Desarrollo de la primera prueba
- Pruebas incrementales basadas en escenarios
- Participación del usuario en la validación
- Uso de herramientas de pruebas automatizadas

Desarrollo de primera prueba

La práctica de desarrollo de la primera prueba consiste en escribir pruebas antes de implementar cualquier código. Esta técnica, también conocida como **TDD (Test-Driven Development)**, asegura que cada funcionalidad se pruebe desde el inicio:

Proceso: Se escribe la prueba, se implementa el código para que pase la prueba y luego se refactoriza el código si es necesario.

Beneficios: Define la interfaz y el comportamiento esperado de cada componente, reduce errores de interpretación de requisitos y evita la acumulación de errores durante el desarrollo

Prueba 4: Comprobación de dosis

Entrada:
1. Un número en mg que represente una sola dosis del medicamento.
2. Un número que signifique el número de dosis individuales por día.

Pruebas:
1. Probar las entradas donde la dosis individual sea correcta, pero la frecuencia muy elevada.
2. Probar las entradas donde la dosis individual sea muy alta y muy baja.
3. Probar las entradas donde la dosis individual × frecuencia sea muy alta y muy baja.
4. Probar las entradas donde la dosis individual × frecuencia esté en el rango permitido.

Salida:
OK o mensaje de error que indique que la dosis está fuera del rango de seguridad.

Descripción de un caso de prueba

Pruebas incrementales y basadas en escenarios

En XP, las pruebas se desarrollan de manera incremental y a partir de escenarios específicos. Los escenarios se derivan de las historias de usuario y reflejan casos de uso realista del sistema:

Escenarios: Cada escenario se convierte en un conjunto de casos de prueba que simulan el uso real del software.

Incrementalidad: A medida que el desarrollo progresa, se añaden nuevas pruebas para cada historia de usuario, validando continuamente la funcionalidad del sistema

Prueba 4: Comprobación de dosis

Entrada:
1. Un número en mg que represente una sola dosis del medicamento.
2. Un número que signifique el número de dosis individuales por día.

Pruebas:
1. Probar las entradas donde la dosis individual sea correcta, pero la frecuencia muy elevada.
2. Probar las entradas donde la dosis individual sea muy alta y muy baja.
3. Probar las entradas donde la dosis individual × frecuencia sea muy alta y muy baja.
4. Probar las entradas donde la dosis individual × frecuencia esté en el rango permitido.

Salida:
OK o mensaje de error que indique que la dosis está fuera del rango de seguridad.

Descripción de un caso de prueba

Participación del usuario y automatización

La colaboración con el usuario es esencial en XP. Los usuarios participan en el proceso de validación, verificando que el sistema se comporte conforme a sus expectativas:

- **Pruebas de aceptación:** Los usuarios o clientes definen las pruebas de aceptación, que validan que el software sea apto para su propósito.
- **Retroalimentación continua:** La participación del usuario permite ajustes rápidos y asegura que el producto final esté alineado con las necesidades del negocio.

XP hace un uso extensivo de herramientas de pruebas automatizadas para asegurar que el sistema sea probado de manera consistente y eficiente:

- **Automatización:** Las pruebas de unidad y de integración se automatizan para ser ejecutadas con cada cambio en el código.
- **Beneficios:** La automatización reduce el esfuerzo manual, mejora la cobertura de pruebas y facilita la integración continua al proporcionar un entorno donde los errores se detectan rápidamente

Programación por pares

La programación por pares es una práctica fundamental de XP donde dos desarrolladores trabajan juntos en una misma estación para escribir código:

- **Dinámica de trabajo:** Uno de los programadores escribe el código mientras el otro revisa, aportando ideas y detectando posibles errores.
- **Objetivo:** Fomentar la colaboración y la revisión continua del código para mejorar la calidad y la comprensión colectiva del Sistema.

La programación por pares ofrece varias **ventajas** importantes:

- **Propiedad Colectiva:** Al trabajar juntos, los programadores comparten la responsabilidad del código y aumentan el conocimiento colectivo.
- **Revisión Continua:** Cada línea de código se revisa en tiempo real, lo que facilita la detección temprana de errores.
- **Apoyo a la Refactorización:** Los pares colaboran para mantener el código simple y limpio, fomentando la mejora continua mediante la refactorización

Desafíos y Productividad en la Programación por Pares

Si bien la programación por pares puede incrementar la calidad, existen desafíos en términos de productividad:

- **Productividad:** Algunos estudios indican que dos programadores trabajando juntos no producen el doble de código que si trabajaran por separado, aunque se reduce el número de errores.
- **Intercambio de Conocimiento:** Los beneficios de calidad y el intercambio de conocimientos pueden compensar la reducción en la productividad, especialmente en entornos donde el conocimiento compartido reduce los riesgos de dependencia.

Administración de un Proyecto ágil

La responsabilidad principal de los administradores es dirigir el proyecto, de modo que el software se entregue a tiempo y con el presupuesto planeado. Supervisan el trabajo de los ingenieros de software y monitorizan el avance en el desarrollo del software.

El enfoque estándar de la administración de proyectos es el basado en un plan. Sin embargo, no funciona bien con los métodos ágiles, donde los requerimientos se desarrollan incrementalmente, donde el software se entrega en rápidos incrementos cortos, y donde los cambios a los requerimientos y el software son la norma. Esto requiere un enfoque diferente a la administración del proyecto, que se adapte al desarrollo incremental y a las fortalezas particulares de los métodos ágiles.

Scrum es un marco de trabajo ágil para la administración de proyectos que enfatiza el desarrollo iterativo y adaptativo. Fue diseñado para ofrecer una estructura que permita a los equipos entregar software funcional en ciclos cortos y repetitivos:

- **Estructura:** Basado en sprints (ciclos de trabajo de tiempo fijo) y reuniones diarias.
- **Propósito:** Facilitar la entrega rápida de incrementos de software, mejorando la adaptación a cambios y retroalimentación constante con el cliente.

Nota: Aunque el enfoque de Scrum es un método ágil general, su enfoque está en la administración iterativa del desarrollo, y no en enfoques técnicos específicos para la ingeniería de software ágil.

Roles y eventos

Scrum se organiza en torno a **roles y eventos clave** para asegurar la colaboración continua y el progreso del proyecto:

Roles:

- **Product Owner:** Representa los intereses del cliente y prioriza el backlog del producto.
- **Scrum Master:** Facilita el proceso, eliminando obstáculos y asegurando que el equipo pueda enfocarse en el sprint.
- **Equipo de Desarrollo:** Trabaja en la implementación de las tareas del sprint.

Eventos:

- **Sprint:** Ciclo de trabajo de tiempo fijo, generalmente de 2 a 4 semanas.
- **Daily Scrum:** Reunión diaria breve donde el equipo discute avances y desafíos.
- **Sprint Review y Sprint Retrospective:** Revisan el incremento del producto y reflexionan sobre mejoras

Proceso Scrum

Existen tres fases con Scrum. La primera es la planeación del bosquejo, donde se establecen los objetivos generales del proyecto y el diseño de la arquitectura de software. A esto le sigue una serie de ciclos sprint, donde cada ciclo desarrolla un incremento del sistema. Finalmente, la fase de cierre del proyecto concluye el Proyecto.

Sprint: El sprint es la unidad central del trabajo en Scrum, en la cual el equipo desarrolla un incremento de software completamente funcional:

- **Backlog de Producto:** Lista priorizada de todas las tareas y características necesarias.
- **Sprint Planning:** Se seleccionan las tareas de mayor prioridad del backlog para el sprint.
- **Desarrollo del Incremento:** Durante el sprint, el equipo se enfoca exclusivamente en las tareas seleccionadas.
- **Revisión y Retroalimentación:** Al final del sprint, el incremento se presenta al cliente para obtener retroalimentación.

Proceso Scrum

Administración con Scrum

La administración en Scrum se basa en la auto-organización y la adaptabilidad:

Enfoque Adaptativo: Los requerimientos y prioridades pueden ajustarse al inicio de cada sprint, facilitando la respuesta a cambios de forma rápida.

Transparencia y Comunicación: Las reuniones diarias y la retroalimentación frecuente aseguran que todo el equipo esté alineado y se minimicen malentendidos.

Responsabilidad Compartida: Scrum evita la figura de un “administrador de proyecto” tradicional y empodera al equipo para tomar decisiones clave

Escalamiento

Los métodos ágiles se desarrollaron para usarse en **pequeños equipos** de programación, que podían trabajar juntos en la misma habitación y comunicarse de manera informal. Por lo tanto, los métodos ágiles se emplean principalmente para el diseño de sistemas pequeños y medianos.

El escalamiento de Scrum para proyectos más grandes y complejos requiere adaptaciones y una mayor coordinación:

- **Equipos Múltiples:** La colaboración entre equipos distribuidos implica establecer mecanismos de comunicación y coordinación.
- **Gestión del Backlog:** Los proyectos grandes suelen necesitar backlogs adicionales, como el backlog de programa o el backlog de portafolio, para coordinar los esfuerzos de diferentes equipos.
- **Integración Continua:** La integración regular de componentes es crucial, especialmente cuando múltiples equipos contribuyen al mismo sistema.
- **Adaptación Cultural y Estructural:** Las organizaciones grandes a menudo enfrentan desafíos al adoptar métodos ágiles debido a su estructura y cultura establecida

Universidad Intercontinental de la Empresa

Asignatura

Arquitectura del Software 

Profesor

Yago Fontenla Seco

{*yago.fontenla1@uie.edu*}