Instituto Universitario para el Desarrollo Humano

(UNINPAHU)

Ingeniería de software

Metodología de Software

LUIS ALBERTO REYES PRIETO

Tercer Semestre

Luisa Fernanda Peña Sanchez

Bogotá D.C

Introducción:

Cuando la metodología ágil experimentó una amplia adopción a principios de la década de los 2000, transformó nuestra forma de desarrollar software. Sin embargo, surgió un fallo: los procesos y requisitos del equipo de operaciones quedaron al margen de esta revolución. Esto llevó a la metodología DevOps, un enfoque que favorecía la coordinación de los equipos de desarrollo y operaciones.

¿Qué es Ágil?

La metodología ágil es un enfoque iterativo de la gestión de proyectos y el desarrollo de software que se centra en la colaboración, el feedback de los clientes y las publicaciones rápidas. Incluye marcos como Scrum y XP, y se consolidó en el Manifiesto Ágil con 12 principios basados en 4 valores

Fundamentales.

Los cuatro valores fundamentales del Manifiesto Ágil

1. Individuos e interacciones sobre procesos y herramientas 2. Software funcionando sobre documentación extensiva 3. Colaboración con el cliente sobre negociación contractual 4. Respuesta ante el cambio sobre seguir un

Plan

**DevOps metologia agil**

DevOps es un enfoque que permite compilar, probar y publicar software de forma más rápida y fiable mediante automatización y colaboración. A diferencia del ágil tradicional, DevOps incluye las operaciones como parte integral del ciclo de vida.

Los tres principios de DevOps

1. Pensamiento sistémico → Comprender que las aplicaciones son sistemas complejos 2. Aumentar los ciclos de feedback →Mejor comunicación entre equipos 3. Cambio cultural aprendizaje continuos

¿Cuándo se combinan Ágil y DevOps?

Experimentación y

DevOps puede considerarse una evolución de las prácticas ágiles o la pieza que faltaba. Mientras Ágil se centra en la iteración del desarrollo, DevOps añade la automatización y la gestión de operaciones para lograr entrega continua y mejoras constantes.

Diferencias y similitudes

Ágil se centra en la colaboración entre desarrollo y gestión de productos, DevOps añade operaciones. Ágil idea hasta código. DevOps entrega y mantenimiento. Ágil enfatiza iteración y lotes pequeños. DevOps enfatiza automatización. Ágil estructura el trabajo planificado. DevOps incluye trabajo no planificado. Ambos comparten valores: personas, colaboración, adaptabilidad.

Conclusión

En última instancia, la metodología ágil y DevOps tienen el mismo objetivo: mejorar la velocidad y la calidad del desarrollo de software. No se excluyen, sino que se complementan. Incorporar DevOps puede ayudar a las organizaciones que no alcanzan los beneficios esperados del ágil por sí solo.

**Metodología ITIL (Information Technology Infrastructure Library)**

**ITIL** es un marco de mejores prácticas para la gestión de servicios de TI. Su objetivo principal es asegurar que los servicios de TI estén alineados con la estrategia y las necesidades del negocio para ofrecer el máximo valor. No es una metodología rígida, sino una guía flexible que se adapta a las necesidades de cada organización.

La última versión, ITIL 4, se enfoca en la co-creación de valor y se integra con enfoques modernos como Agile, DevOps y Lean. Sus principios rectores clave son:

* **Enfoque en el valor**: Asegurar que cada actividad y servicio de TI contribuya directamente al valor del negocio.
* **Progresar de forma iterativa con retroalimentación**: Fomentar la mejora continua en cada paso.
* **Colaborar y promover la visibilidad**: Romper los silos y fomentar la transparencia entre los equipos.
* **Mantenerlo simple y práctico**: Eliminar pasos innecesarios para aumentar la eficiencia.
* **Optimizar y automatizar**: Aprovechar la tecnología para mejorar los procesos y reducir la intervención manual.

ITIL define un **ciclo de vida del servicio** que incluye fases de Estrategia, Diseño, Transición, Operación y Mejora Continua, lo que proporciona una estructura sólida para gestionar servicios de TI de manera integral. La implementación de ITIL ayuda a las empresas a mejorar la calidad del servicio, reducir costos operativos y promover la mejora continua en la gestión de la tecnología.

**Metodología COBIT ( control objectives for information and related techology)**

Es un marco de gobierno y gestión de TI que busca alinear la tecnología con los objetivos del negocio, proteger los activos de información y satisfacer las necesidades de las partes interesadas.

Historia:

1996: surge como un conjunto de objetivos de control de TI.

2000: tercera versión, incorpora pautas de gestión.

2011: COBIT 5, marco de gobernanza empresarial.

2018: COBIT 2019, más flexible, integra mediciones de madurez y capacidad.

Principios:

1. Satisfacer necesidades de las partes interesadas.
2. Facilitar un enfoque holístico.
3. Gobernanza dinámica y adaptable.
4. Separación entre gobernanza y gestión.
5. Definir y priorizar componentes de acuerdo a la empresa.
6. Evaluación y mejora continua.

Facilitadores:

* Principios, políticas y marcos.
* Procesos.
* Estructuras organizacionales.
* Cultura, ética y comportamiento.
* Información.
* Servicios, infraestructura y aplicaciones.
* Personas, habilidades y competencias.

Estructura:

Se basa en tres niveles:

1. Requisitos de negocio (eficacia, integridad, disponibilidad, confidencialidad, etc.).
2. Recursos de TI (infraestructura, aplicaciones, información y personas).
3. Procesos de TI divididos en dominios.

Componentes:

* Marco de procesos.
* Objetivos de control.
* Directrices de gestión.
* Modelos de madurez y capacidad.

Beneficios:

* Información confiable para la toma de decisiones.
* Uso innovador y eficiente de TI.
* Reducción de riesgos tecnológicos.
* Optimización de costos de TI.
* Cumplimiento normativo.
* Excelencia operativa.

Ejemplos de uso:

* Telecomunicaciones: preparación para auditorías, reportes de seguridad, controles normativos.
* Banca: controles de acceso, políticas de respaldo, monitoreo de riesgos en tiempo real.

Conclusión:

COBIT ofrece un enfoque estructurado para el gobierno de TI, garantizando control, seguridad, eficiencia y alineación estratégica entre la tecnología y los objetivos del negocio.

**Mitología incremental**

La metodología incremental es un enfoque moderno de desarrollo de software que divide un proyecto en etapas pequeñas llamadas incrementos. Cada incremento agrega nuevas funcionalidades, permitiendo contar con un producto usable desde fases tempranas.

Ideas principales:

* Entregas parciales: se obtiene un producto funcional en cada ciclo.
* Retroalimentación constante: se ajusta el desarrollo según la opinión del cliente.
* Adaptabilidad: permite integrar cambios fácilmente según las necesidades.

Características:

1. División del proyecto en módulos manejables.
2. Entregas frecuentes de funcionalidades
3. Flexibilidad para cambios.
4. Progreso acumulativo: cada incremento se basa en el anterior.

Fases del ciclo incremental:

1. Requerimientos.
2. Definición y diseño.
3. Desarrollo (codificación).
4. Validación (pruebas).
5. Integración.
6. Entrega final del sistema completo.

Beneficios:

* Menor riesgo y validación temprana.
* Fácil de probar y depurar.
* Entrega rápida de valor al cliente.
* Producto usable desde las primeras etapas.

Desafíos:

1. Requiere planificación extensa.
2. No es ideal para sistemas en tiempo real o con requisitos muy fijos.
3. Riesgo de falta de cohesión si no se coordinan bien los incrementos.

Ejemplo:

Un procesador de textos: primero se crea la edición básica, luego corrector y formato, después insertar imágenes/tablas, y finalmente funciones avanzadas como plantillas o colaboración.

Casos de uso:

* Proyectos con requisitos cambiantes.
* Aplicaciones web y móviles.
* Organizaciones ágiles.

Comparación con Cascada:

Cascada: rígido, secuencial, entrega solo al final.

Incremental: flexible, iterativo, entrega parcial y temprana.

**Metodología KANBAN**

Es una metodología ágil de gestión visual del trabajo que organiza tareas mediante tableros divididos en columnas (por hacer, en proceso, hecho). Se basa en tarjetas que se mueven ssegun el avance de la tarea.

Origen:

* Década de 1940, Japón, creada por taiichi ohno en toyota.
* Filosofía: justo a tiempo, jidoka(automatización con toque humano) y kaizen(mejora continua)
* En 2007, David J. Anderson la adapto al desarrollo de software.

Características principales:

1. Visualización del flujo de trabajo (tableros físicos o digítales como Tello, jira, Azure DevOps).
2. Limites de trabajo en progreso(wip)para evitar sobrecarga.
3. Entrega continua y enfoque en terminar tareas.
4. Flexibilidad sin interaciones fijas (a diferencias de scrum).
5. Metricas: leadtime, cycle time, throughput.

Metodología en software:

1. Definir el flujo (Backlog -> Desarrollo -> Pruebas -> Deploy).
2. Diseñar el tablero KANBAN.
3. Crear tarjetas de tareas, bugs o historias de usuario.
4. Aplicar limites wip.
5. Monitorear bloqueos con reuniones rápidas.
6. Mejorar procesos con métricas.

Evolución y versiones:

* KANBAN original (toyota, manufactura).
* KANBAN en ti (2004-2007, Anderson).
* KANBAN Method (210, formalizado por kanban University).
* Scrum-ban (2011, mezcla entre Scrum y Kanban).
* Enterprise Services Planning (ESP) (2014, gestion de portafolio).
* Flight Levels (2019, aplicación en niveles: equipos, coordinación y estrategias).

Usos en software:

* Gestión de bugs y tickets.
* Priorización de funcionalidades.
* Soporte técnico..
* Coordinación de DevOps
* Planificación de raleases.

Comparación con otras metodologías:

* Scrum: ambos usan tableros, preo Scrum tiene roles y sprints fijos, mientras Kanban es flexible y continuo.
* XP: ambos buscan calidad y entregas rápidas, pero XP se centra en prácticas de codificación y Kanban en gestión del flujo.
* Cascada: rígido y secuencial, en contraste con la flexibilidad visual e interactiva en Kanban.

**METODOLOGIA XP**

Extreme Programming (XP) es una metodología ágil que optimiza el desarrollo

de software. Su objetivo principal es entregar productos de alta calidad

mediante la implementación de ciclos cortos, una colaboración constante y

una adaptabilidad excepcional a los cambios del proyecto.

**5 VALORES FUNDAMENTALES**

1. COMUNICACION
2. SIMPLICIDAD
3. RETROALIMENTACIÓN
4. CORAJE
5. RESPETO.

**FUNCIONALIDAD**

Funciona con un ciclo de 4 actividades

* Codificar:Transformar los requisitos en funcionalidades tangibles mediante la escritura de código limpio y eficiente.
* Diseñar:Mantener un diseño de software simple y flexible que facilite futuras modificaciones y expansiones.
* Probar:Realizar pruebas unitarias y de integración de manera constante para detectar y corregir errores tempranamente.
* Escuchar:Comprender a fondo las necesidades cambiantes del cliente y ajustar el desarrollo en función de su feedback

**BENEFICIOS**

* Mayor Calidad:Las pruebas continuas y las revisiones de código aseguran un software robusto y fiable.
* Alta Flexibilidad:El modelo se adapta con facilidad a los cambios en los requisitos sin afectar el ritmo del proyecto.
* Mejor Colaboración:Fomenta una comunicación fluida, reduciendo errores y malentendidos dentro del equipo.
* Ahorro:Minimiza retrabajos costosos y tardíos, optimizando el tiempo y los recursos invertidos.
* Satisfacción:Resultados que satisfacen tanto al equipo de desarrollo como al cliente final.

**METODOLOGIA DE PROTOTIPOS**

Un modelo de prototipo es el proceso de crear, probar y desarrollar una réplica funcional de un producto o servicio. En este caso, nos referimos a software web o de aplicaciones móviles . Es popular y uno de los modelos de Ciclo de Vida de Desarrollo de Software (SDLC) más utilizados . En este proceso, se crea una versión aceptable del software, se obtienen comentarios de los clientes y se perfecciona hasta que sea satisfactoria

1. Descarte rápido

2. Evolutivo

3. Incremental

4. Extremo

**VENTAJAS**

* Comentarios y participación: Los usuarios pueden opinar desde el inicio, lo que mejora el software y su satisfacción.
* Reutilizable: Los prototipos pueden aprovecharse en otros proyectos.
* Corrección temprana: Se detectan y solucionan errores antes del producto final.
* Mejor comprensión: Los usuarios aprenden sobre el desarrollo del software al participar activamente.
* Modelo sencillo: Es fácil de entender y no requiere gran experiencia técnica.

**DESVENTAJAS**:

* Requiere tiempo: El proceso es lento por las pruebas y los ajustes continuos.
* Costoso: Se crean varias versiones que a veces no se usan, generando gastos.
* Difícil implementar todo: No siempre se pueden incluir todas las sugerencias de los usuarios.
* Dependencia excesiva: Confiar demasiado en los comentarios puede causar requisitos poco claros al inicio.

**METODOLOGÍA RAD**

La metodología RAD (desarrollo rápido de aplicaciones) tiene enfoque sgul para el desarrollo de software, prioriza la creación fugaz de prototipos funcionales a través de iteraciones y la retroalimentación constante de los usuarios.

**OBJETIVOS**

* Asegurar la uniformidad y calidad del desarrollo y sistema
* Cumplir con las necesidades del usuario
* Tener un buen rendimiento y eficiencia en su trabajo
* Unificar criterios en la organización para el desarrollo del l sostén la de información
* Proporcionar puntos de control y revisión
* Ayudar a identificar lo antes posible cualquier cambio que se tenga que realizar dentro del desarrollo

**SUS 5 FASES**

1. Modelado de negocios
2. Modelado de datos
3. Modelado de procesos
4. Generación de aplicaciones
5. Pruebas y revisión

**BENEFICIOS**

* **los ciclos de desarrollo son más cortos de lo normal**
* El producto se crea en base a los comentarios del usuario
* Utiliza un enfoque flexible para adaptarse a los requisito cambiantes
* Los bloques de código reutilizables permiten realiza r pruebas más rápidas y reducir los defectos

**DESVENTAJASr**

* Requiere equipos muy capacitados y con experiencia.
* Proyectos grandes con muchos interesados pueden tener problemas de colaboración.
* Solo funciona bien en sistemas que se pueden dividir en módulos.
* Los requisitos del usuario deben estar claramente definidos durante todo el proyecto.
* Puede ser costoso por el modelado y generación automática de código.
* Es mejor para proyectos de corto plazo; en proyectos largos no es tan efectivo.

**METODOLOGÍA SCRUM**

**ROLES**

* product owner define y prioriza el trabajo
* Scrum Master quita obstáculos del objetivo
* Equipo de desarrollo trabaja en el proyecto

**SPRINTS**

* Sprint interacción 1 a 4 semanas de trabajo
* Sprint plannig planear el trabajo a realizar
* Sprint daily reunión diaria para acomodar roles
* Sprint rewiew ver el trabajo finalizado hacer retroalimentación

Sprint retrospective buscar mejorías