Ciclos de Vida de Desarrollo

Análisis y Diseño I
Undécimo A
20 de julio del 2025

Ciclo de Vida de Desarrollo de Software

El **Ciclo de Vida de Desarrollo de Software (SDLC)**, por sus siglas en inglés (*Software Development Life Cycle*), es un proceso estructurado y sistemático que describe las fases por las que pasa un proyecto de software desde su inicio hasta su finalización y mantenimiento. No es una metodología única, sino un marco que puede ser adaptado y personalizado según las necesidades del proyecto y la organización.

El SDLC proporciona un enfoque claro y organizado para la construcción de software, asegurando que se cumplan los objetivos, se gestionen los recursos de manera eficiente y se entregue un producto de calidad.

Importancia del SDLC:

Estructura y Organización: Proporciona un plan claro y una secuencia lógica de actividades, lo que facilita la gestión y el seguimiento del proyecto.

Gestión de la Complejidad: Divide un proyecto grande y complejo en fases más pequeñas y manejables.

Reducción de Riesgos: Permite identificar y mitigar riesgos en etapas tempranas del desarrollo.

Mejora de la Calidad: A través de fases de pruebas y validación, se busca asegurar que el software cumpla con los requisitos y sea de alta calidad.

Comunicación y Colaboración: Establece puntos de control y entregables que facilitan la comunicación entre los equipos y con los stakeholders.

Optimización de Recursos: Ayuda a asignar recursos (tiempo, personal, presupuesto) de manera más efectiva.

Modelo Cascada (Waterfall Model)

El **Modelo Cascada** es uno de los SDLC más antiguos y tradicionales. Es un enfoque lineal y secuencial, donde el progreso fluye de forma descendente (como una cascada) a través de las fases del proyecto. Cada fase debe completarse y validarse antes de que la siguiente pueda comenzar.

Fases del Modelo Cascada:

Requisitos: Se recopilan y documentan todos los requisitos del sistema de manera exhaustiva.

Diseño: Se define la arquitectura del sistema, el diseño de la base de datos, la interfaz de usuario y los componentes del software.

Implementación (Codificación): Los desarrolladores escriben el código basándose en las especificaciones de diseño.

Pruebas: Se verifica que el software cumpla con los requisitos y funcione correctamente, identificando y corrigiendo errores.

Despliegue (Implementación): El software se instala y se pone en funcionamiento en el entorno del usuario final.

Mantenimiento: Se realizan correcciones de errores, mejoras y actualizaciones del software a lo largo de su vida útil.

Ventajas:

- **Simplicidad:** Es fácil de entender y de gestionar, con fases claras y entregables definidos.
- **Control:** Proporciona un control estricto sobre el proyecto, ya que cada fase debe ser aprobada antes de pasar a la siguiente.
- **Documentación:** Genera una gran cantidad de documentación en cada fase, lo que puede ser útil para proyectos con requisitos muy estables.

Desventajas:

- **Inflexibilidad:** Es muy rígido y difícil de adaptar a cambios en los requisitos una vez que una fase ha sido completada.
- **Riesgos Tardíos:** Los errores y problemas críticos a menudo se descubren en las fases de prueba o mantenimiento, lo que los hace costosos de corregir.
- **Poca Interacción con el Cliente:** El cliente tiene poca participación después de la fase inicial de requisitos, lo que puede llevar a un producto final que no satisfaga completamente sus necesidades.
- **No apto para Proyectos Complejos:** No es ideal para proyectos grandes o con requisitos inciertos o cambiantes.

Modelo Espiral (Spiral Model)

El **Modelo Espiral** es un SDLC iterativo y basado en riesgos, propuesto por Barry Boehm. Combina elementos del modelo cascada con la naturaleza iterativa del prototipado, enfocándose en la mitigación de riesgos en cada ciclo. El proyecto avanza en "espirales", donde cada vuelta de la espiral representa una fase del desarrollo.

Fases Iterativas (por cada vuelta de la espiral):

1. **Planificación:** Se definen los objetivos para la iteración actual y se identifican las alternativas para lograrlos.
2. **Análisis de Riesgos:** Se identifican, analizan y mitigan los riesgos asociados con las alternativas. Esta es la fase central del modelo.
3. **Ingeniería:** Se desarrolla y prueba el producto o una parte de él (prototipo).
4. **Evaluación (y Planificación para la Siguiente Iteración):** Se evalúan los resultados de la ingeniería, se obtiene feedback del cliente y se planifica la siguiente iteración.

Ventajas:

- **Excelente Gestión de Riesgos:** El enfoque en el análisis y mitigación de riesgos en cada iteración lo hace muy adecuado para proyectos de alto riesgo.
- **Flexibilidad:** Permite incorporar cambios en los requisitos en etapas posteriores del desarrollo.
- **Adecuado para Proyectos Grandes y Complejos:** Su naturaleza iterativa lo hace apto para proyectos donde los requisitos pueden evolucionar.

- **Interacción con el Cliente:** El cliente puede participar en cada iteración, proporcionando retroalimentación.

Desventajas:

- **Complejidad:** Es más complejo de gestionar que el modelo cascada.
- **Costo:** Puede ser más costoso debido a la necesidad de un análisis de riesgos constante y múltiples iteraciones.
- **Requiere Experiencia:** Necesita un equipo con experiencia en gestión de riesgos y desarrollo iterativo.
- **No apto para Proyectos Pequeños:** Su complejidad lo hace ineficiente para proyectos pequeños y de bajo riesgo.

Modelos Ágiles (Agile Models)

Los **Modelos Ágiles** representan un conjunto de metodologías de desarrollo de software que priorizan la flexibilidad, la colaboración con el cliente y la entrega rápida e incremental de software funcional. Surgieron como una respuesta a las limitaciones de los modelos tradicionales (como Cascada) en entornos donde los requisitos son cambiantes y la velocidad de entrega es crucial.

El **Manifiesto Ágil**, publicado en 2001, establece los cuatro valores fundamentales de los modelos ágiles:

- **Individuos e interacciones** sobre procesos y herramientas.
- **Software funcionando** sobre documentación exhaustiva.
- **Colaboración con el cliente** sobre negociación contractual.
- **Respuesta al cambio** sobre seguir un plan.

Principios Clave de los Modelos Ágiles:

- **Desarrollo Iterativo e Incremental:** El software se construye en pequeñas piezas (iteraciones o sprints) que se entregan de forma frecuente.
- **Adaptabilidad al Cambio:** Se valora la capacidad de responder a los cambios en los requisitos, incluso en etapas tardías del desarrollo.
- **Colaboración Continua:** Se fomenta la comunicación y colaboración constante entre el equipo de desarrollo y los stakeholders (incluyendo el cliente).
- **Entregas Frecuentes de Valor:** Se entrega software funcional en intervalos regulares y cortos.
- **Auto-organización del Equipo:** Los equipos son multifuncionales y auto-organizados.

Ejemplos de Metodologías Ágiles:

- **Scrum:** El framework ágil más popular, enfocado en la gestión de proyectos a través de iteraciones cortas llamadas "sprints".
- **Kanban:** Un método visual para gestionar el flujo de trabajo, centrado en la mejora continua y la reducción del trabajo en progreso.
- **Extreme Programming (XP):** Un conjunto de prácticas de ingeniería de software que promueven la calidad del código y la capacidad de respuesta a los requisitos cambiantes.

Modelo Scrum (Ejemplo de Ágil)

Scrum es un framework ligero para gestionar proyectos complejos, especialmente en el desarrollo de software. No es una metodología prescriptiva, sino un conjunto de valores, principios y prácticas que ayudan a los equipos a entregar valor de forma iterativa e incremental.

Elementos Clave de Scrum:

1. **Roles:**
    - **Product Owner:** Representa los intereses del cliente y los stakeholders. Es responsable de definir y priorizar el "Product Backlog".
    - **Scrum Master:** Facilita el proceso Scrum, elimina impedimentos y asegura que el equipo siga las prácticas de Scrum. No es un gerente de proyecto tradicional.
    - **Equipo de Desarrollo:** Un equipo auto-organizado y multifuncional que realiza el trabajo de desarrollo.
2. **Artefactos:**
    - **Product Backlog:** Una lista dinámica y priorizada de todas las funcionalidades, características, mejoras y correcciones que se desean para el producto.
    - **Sprint Backlog:** Un subconjunto de elementos del Product Backlog seleccionados por el Equipo de Desarrollo para ser completados en un Sprint.
    - **Incremento:** La suma de todos los elementos del Sprint Backlog completados durante un Sprint, más el valor de todos los incrementos anteriores. Debe ser un software potencialmente entregable.
3. **Eventos (Reuniones):**
    - **Sprint Planning:** Al inicio de cada Sprint, el equipo planifica qué se entregará y cómo se hará.
    - **Daily Scrum (Stand-up):** Una reunión diaria de 15 minutos para que el Equipo de Desarrollo sincronice actividades y planifique el trabajo para las próximas 24 horas.
    - **Sprint Review:** Al final del Sprint, el equipo presenta el Incremento a los stakeholders para obtener retroalimentación.
    - **Sprint Retrospective:** Después del Sprint Review, el equipo reflexiona sobre el Sprint y planifica mejoras para el próximo.

Ventajas de Scrum:

- **Adaptabilidad:** Muy flexible a los cambios de requisitos.
- **Entrega Rápida:** Proporciona software funcional de forma regular y frecuente.
- **Colaboración:** Fomenta la comunicación y el trabajo en equipo.
- **Transparencia:** El progreso es visible y se obtiene feedback continuo.
- **Mejora Continua:** Las retrospectivas permiten al equipo aprender y mejorar constantemente.

Desventajas de Scrum:

- **Requiere Compromiso:** El éxito depende del compromiso de todo el equipo y los stakeholders.
- **Falta de Documentación Detallada:** Puede no ser adecuado para entornos que requieren documentación exhaustiva desde el inicio.
- **Dependencia del Scrum Master:** Un Scrum Master inexperto puede afectar la eficiencia.

Comparación de Modelos SDLC

Característica	Modelo Cascada	Modelo Espiral	Modelos Ágiles (Scrum)
Enfoque Principal	Lineal, secuencial, fases rígidas	Iterativo, basado en riesgos	Iterativo, incremental, centrado en el cliente
Flexibilidad al Cambio	Baja (difícil y costoso)	Alta (permite cambios en cada iteración)	Muy Alta (bienvenida al cambio continuo)
Gestión de Riesgos	Baja (riesgos descubiertos tarde)	Muy Alta (análisis y mitigación constante)	Alta (adaptativa, feedback temprano)
Interacción Cliente	Baja (principalmente al inicio y final)	Media a Alta (en cada iteración)	Muy Alta (colaboración continua)
Tamaño de Proyecto	Pequeños, requisitos estables y bien definidos	Grandes, complejos, requisitos cambiantes o inciertos	Cualquier tamaño, requisitos evolutivos, entregas rápidas
Documentación	Muy alta y formal	Media a Alta	Baja (énfasis en software funcionando)
Tiempo de Entrega	Largo (entrega única al final)	Iteraciones, entregas de prototipos	Corto (entregas frecuentes de incrementos funcionales)

¿Cuándo Elegir Cada Modelo?

La elección del modelo SDLC adecuado depende de varios factores del proyecto, incluyendo la claridad de los requisitos, el nivel de riesgo, el tamaño del proyecto, la disponibilidad del cliente y la cultura organizacional.

- **Modelo Cascada:**
  - **Ideal para:** Proyectos pequeños con requisitos muy claros, estables y bien definidos desde el principio.
  - **Ejemplos:** Sistemas regulatorios con especificaciones fijas, proyectos donde la documentación es más crítica que la flexibilidad.
- **Modelo Espiral:**
  - **Ideal para:** Proyectos grandes, complejos y de alto riesgo, donde los requisitos no están completamente definidos al inicio y se espera que evolucionen.
  - **Ejemplos:** Desarrollo de nuevos productos con tecnologías emergentes, sistemas de misión crítica donde la mitigación de riesgos es primordial.
- **Modelos Ágiles (Scrum/Kanban):**
  - **Ideal para:** Proyectos con requisitos que cambian con frecuencia, donde se necesita una entrega rápida y continua de valor. Cuando la colaboración activa con el cliente es posible y deseada.
  - **Ejemplos:** Desarrollo de aplicaciones móviles, plataformas web, startups, proyectos con alta incertidumbre en el mercado.