PROCESOS DE INGENIERÍA DE SOFTWARE

AUTOR: HELLEN CUBERO

NOVIEMBRE: 2020

San Marcos

Introducción

La presente lectura tiene como objetivo conocer los procesos de ingeniería de software, también conocido como ciclo de vida, las fases que comprende y las metodologías de desarrollo de software, así como su importancia.

Con el avance de la tecnología y la actualización de metodologías de desarrollo de software se presenta una pequeña reseña sobre su evolución con el paso del tiempo.

Además se da a conocer las principales diferencias entre las metodologías tradicionales y las metodologías orientadas a objetos.

Contenido

Ciclo de vida del software

Según Campderrich (2013) el ciclo de vida del software está constituido por el conjunto de todas las etapas que preceden y suceden a la producción de software.
Los métodos y técnicas de la ingeniería del software se inscriben dentro del marco delimitado por el ciclo de vida del software y más concretamente, por las diferentes etapas que se distinguen.
La existencia de diferentes modelos de ciclos de vida del software hace comprender que no hay ninguno que sea ideal o que no tenga grandes limitaciones, sin embargo es indispensable que cada proyecto se desarrolle dentro de un marco de un ciclo de vida bien definido.

Todos los sistemas de información pasan por una serie de fases a lo largo de su vida. Según Berzal (s.f) el ciclo de vida comprende las siguientes etapas:

1. Planificación: es el proceso de organizar un proyecto, corresponde a las tareas iniciales de la fase inicial e incluyen actividades como la determinación del ámbito del proyecto, la realización de un estudio de viabilidad, el análisis de los riesgos asociados al proyecto, una estimación del coste del proyecto, su planificación temporal y la asignación de recursos a las distintas etapas del proyecto.

2. Análisis: corresponde al proceso mediante el cual se intenta descubrir qué es lo que se necesita para obtener una comprensión adecuada de los **requerimientos del sistema**.
Es una etapa de esencial porque debe obtenerse con precisión lo que se necesita, de lo contrario ningún proceso de desarrollo permitirá obtenerlo. Un aspecto fundamental que debe considerarse en esta etapa es que muchas veces el cliente no sabe con exactitud lo que necesita, por lo tanto se debe colaborar en averiguarlo.

3. Diseño: hace referencia a los modelos que se utilizan para representar las **características del sistema** que permiten la implementación de manera efectiva, es decir, se estudian las posibles alternativas de implementación para el sistema que debe construirse y se decide la estructura general que

se otorgará (el diseño arquitectónico).
Es importante tener en cuenta que la propuesta inicial de diseño no siempre es la adecuada, por lo que debe refinarse y, tampoco es necesario iniciar de cero, pues existen auténticos catálogos de patrones de diseño que pueden servir para aprender de los errores que otros han cometido y tratar de evitarlos.

4. Implementación: después de conocer las funciones que debe desempeñar el sistema de información (análisis) y decidir sobre cómo organizar sus distintos componentes (diseño), inicia la etapa de implementación. Antes de iniciar a programar el sistema o crear la base de datos, es fundamental haber comprendido el problema o la necesidad que pretende resolver y haber aplicado principios básicos de diseño que le permitan construir un sistema de calidad.
En esta fase se debe seleccionar las herramientas adecuadas, el entorno de desarrollo, el lenguaje de programación apropiado para el tipo de sistema que se desea construir.
En el momento de programar se debe procurar que el código no resulte indescifrable. Para que el código sea legible, debe tomarse en cuenta aspectos como seleccionar algoritmos y estructuras de datos adecuados, mantener la lógica de la aplicación de manera sencilla, documentar cada proceso del sistema, facilitar la interpretación visual, entre otros.

5. Pruebas: esta fase tiene como objetivo detectar los errores cometidos en las etapas anteriores del proyecto (y eventualmente corregirlos). Existen diferentes tipos de pruebas:

- Pruebas de unidad: sirven para comprobar el correcto funcionamiento de componente específico.
- Pruebas de integración: se realizan cuando se unen los componentes que conforman el sistema y sirven para detectar errores en sus interfaces.
- Pruebas alfa: las realiza el usuario final después de finalizado el sistema, el objetivo es ayudar a pulir aspectos de la interfaz de usuario.
- Pruebas beta: se realizan cuando el sistema no es un producto a la medida, sino que se venderá como un producto en el mercado. Son realizadas por usuarios finales, ajenos al equipo de desarrollo.

- Test de aceptación: se utilizan en sistemas a la medida, si es aceptado de manera exitosa, se informa oficialmente el final del proceso de desarrollo e inicia la fase de mantenimiento.
- Por último, a lo largo del ciclo de vida, se realizan revisiones de todos los productos generados a lo largo del proyecto, desde la especificación de requerimientos hasta el código de los diferentes módulos que integran la aplicación.

6. Instalación o despliegue: antes de poner el sistema en producción, se debe planificar el ambiente o entorno donde funcionará el sistema, contemplando aspectos tanto de hardware como de software: equipos necesarios y su configuración física, redes de interconexión entre equipos internos y externos, sistemas operativos actualizados, bibliotecas, componentes suministrados por terceros, entre otros.

7. Uso y mantenimiento: en esta fase se pueden presentar tres tipos de mantenimiento:

- Mantenimiento correctivo: para eliminar los defectos que se detecten en su vida útil.
- Mantenimiento adaptativo: para adaptar el sistema a nuevas necesidades (por ejemplo hardware, software)
- Mantenimiento perfectivo: para agregar nuevas funcionalidades, que mejoren el sistema ya existente.

Para obtener más información acerca de estas fases, visite el siguiente enlace.

Procesos del ciclo de vida (ISO 12207)

Según the International Organization for Standardization (2017), la norma ISO 12207 establece un marco común para los procesos de ciclo de vida del software, con terminología bien definida, que puede ser referenciado por la industria del software.
Contiene procesos, actividades y tareas que se aplican durante la

adquisición de un producto o servicio de software y durante el suministro, desarrollo, operación, mantenimiento y eliminación de un sistema.

Los procesos del ciclo de vida establecidos mediante la norma ISO 12207, según García, Lameiro & Quattrocchio (2014), son los siguientes:

- **Procesos primarios:** que posee las fases de adquisición, suministro, desarrollo y mantenimiento.
- **Procesos de soporte:** que contiene ocho procesos de apoyo: documentación, administración de la configuración, aseguramiento de la calidad, verificación, validación, revisiones conjuntas, auditorías y resolución de problemas.
- **Procesos organizacionales:** contiene un conjunto de cuatro procesos de la organización: administración o gestión, infraestructura, mejoras o progreso y entrenamiento o capacitación.

Para obtener más información, haga clic en el enlace y observe la lectura complementaria ISO_12207.

Ciclos de vida tradicionales

Los ciclos de vida tradicionales están basados en una metodología estructurada, esto significa que cada función a realizar se descompone en pequeños módulos individuales. (Universidad del Papaloapan, 2015) Entre los más conocidos se encuentran los siguientes:

Ciclo de vida clásico

Conocido también como ciclo de vida en cascada. Según Berzal (s.f), se avanza en orden, de una etapa a la siguiente solo tras finalizar con éxito las tareas de verificación y validación propias de la etapa. Si resulta necesario, se da marcha atrás hasta la fase inmediatamente anterior.

Este modelo tradicional, exige una aproximación secuencial al proceso de desarrollo de software, sin embargo provoca graves inconvenientes debido a:

- Los proyectos reales raramente siguen el flujo secuencial de actividades que propone este modelo.
- Normalmente, es difícil conocer todos los requerimientos del proyecto desde el inicio.

- No existe una versión operativa del sistema hasta las etapas finales del proyecto, por lo que cualquier rectificación de cualquier decisión tomada de manera equivocada, supondrá un coste adicional, tanto económico como temporal.

El ciclo de vida en cascada se representa con la siguiente figura:

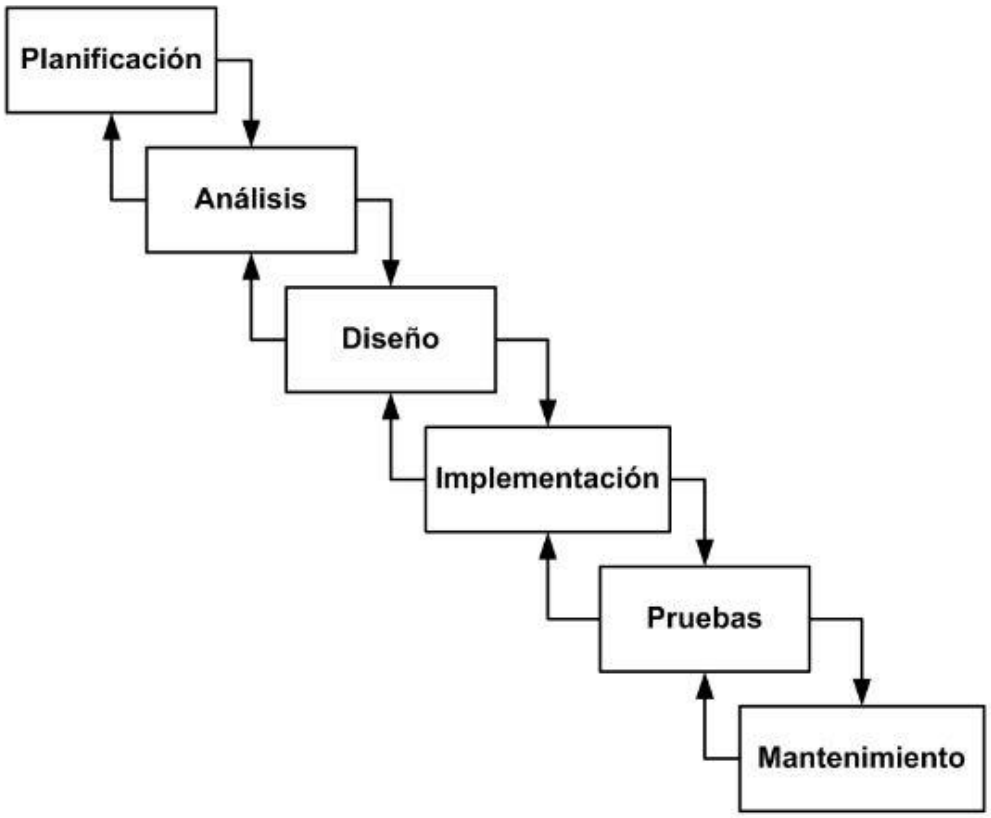

Ilustración 1: Ciclo de vida clásico
Fuente: Berzal (s.f)

Ciclo de vida iterativo e incremental

También derivado del ciclo de vida en cascada, este modelo busca reducir el riesgo que surge entre las necesidades del usuario y el producto final por malos entendidos durante la etapa de solicitud de requerimientos.

Al final de cada iteración se le entrega al cliente una versión mejorada o con mayores funcionalidades del producto. El cliente es quien luego de cada iteración, evalúa el producto y lo corrige o propone mejoras. (Universidad del Papaloapan, 2015)

Se representa con el siguiente diagrama:

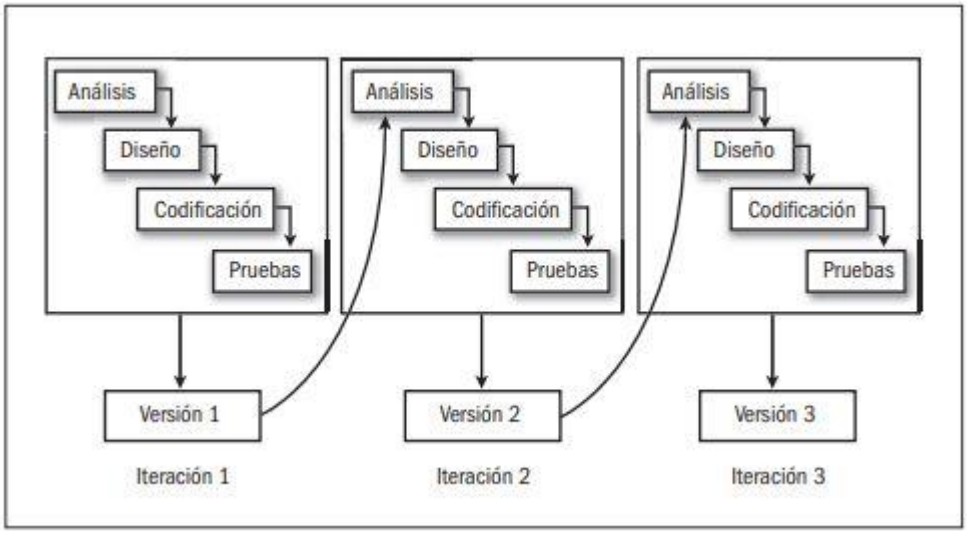

Ilustración 2: Ciclo de vida iterativo
Fuente: Universidad del Papaloapan (2015)

Ciclo de vida en espiral

Este modelo hace hincapié en la gestión de riesgos y define cuatro actividades principales: planificación (determinar objetivos, alternativas y restricciones del proyecto), análisis de riesgos (análisis de alternativas, identificación y resolución de riesgos), ingeniería (desarrollo del producto) y evaluación (valoración por parte del cliente y valoración de los resultados obtenidos frente a la siguiente iteración).

Se representa con el siguiente diagrama:

Ilustración 3: Ciclo de vida en espiral
Fuente: Berzal (s.f)

Ciclo de vida en V

Este ciclo fue diseñado por Alan Davis, y contiene las mismas etapas del ciclo de vida en cascada. A diferencia de aquél, a este se le agregaron dos subetapas de retroalimentación entre las etapas de análisis y mantenimiento, y entre las de diseño y debugging.

Se representa con el siguiente diagrama:

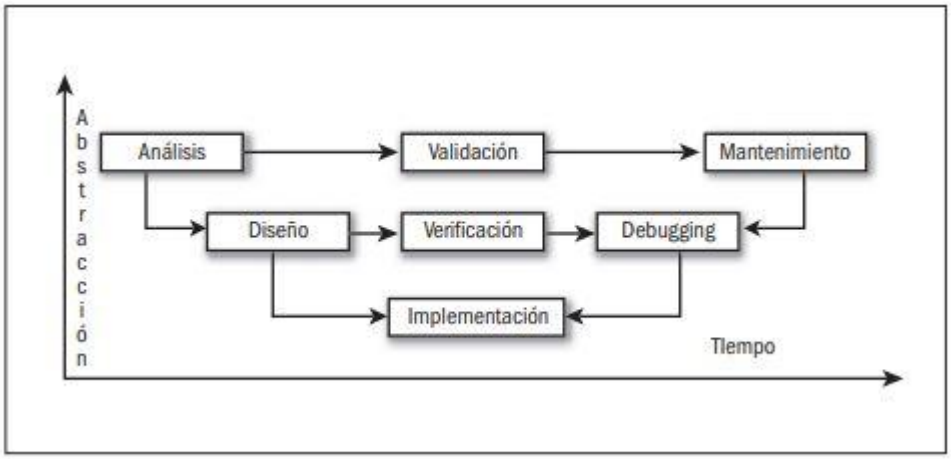

Ilustración 4: ciclo de vida en V
Fuente: Universidad del Papaloapan (2015)

Ciclo de vida para sistemas orientados a objetos

A diferencia de la metodología estructurada, la metodología orientada a objetos no comprende los procesos como funciones sino que arma módulos basados en componentes, es decir, cada componente es independiente del otro. Esto nos permite que el código sea reutilizable.
La metodología estructurada realiza división de procesos según complejidad, mientras que la orientada a objetos realiza la división según funcionalidad. (Universidad del Papaloapan, 2015)

Ciclo de vida orientado a objetos
Esta técnica fue presentada en la década del 90. Al igual que la filosofía del paradigma de la programación orientada a objetos, en esta metodología cada funcionalidad, o requerimiento solicitado por el usuario, es considerado un objeto. Los objetos están representados por un conjunto de propiedades, a los cuales denominamos **atributos**, por otra parte, al comportamiento que tendrán estos objetos los denominamos **métodos**. (Universidad del Papaloapan, 2015)

Se representa con el siguiente diagrama:

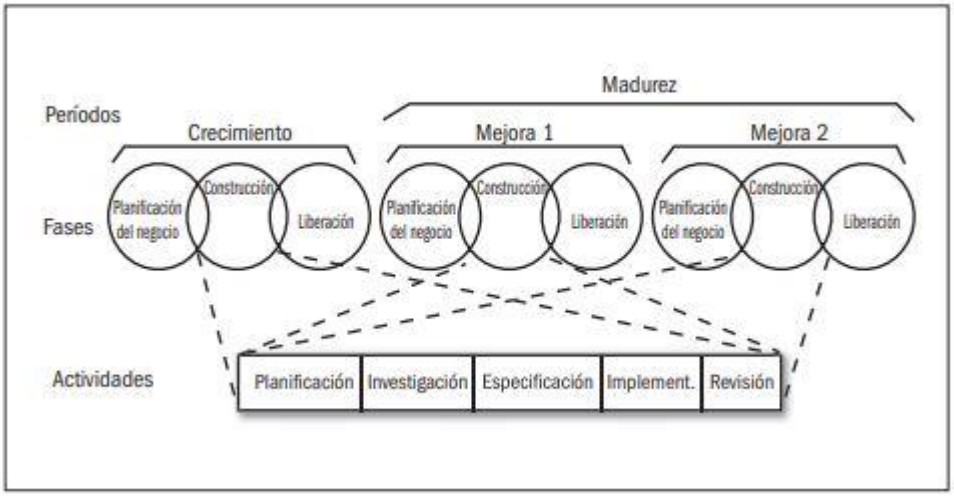

Ilustración 5: Modelo orientado a objetos
Fuente: Universidad del Papaloapan (2015)

Metodologías de desarrollo de software: concepto, evolución histórica y tipos

Concepto

Una metodología de desarrollo de software es un marco de trabajo que se usa para estructurar, planificar y controlar el proceso de desarrollo de sistemas de información. (Maida & Pacienzia, 2015).

Evolución histórica

Las metodologías de Desarrollo de Software (DS.) han experimentado un proceso histórico y evolutivo que inicia en los años 40 con la aparición de las primeras computadoras, entonces no se contaban con parámetros ni estándares, el DS. Era prácticamente empírico y artesanal lo que llevó a que una buena parte de los proyectos fallaran en cubrir las exceptivas de los usuarios, así como en entregas extemporáneas y presupuestos excedidos, sobreviniendo la "crisis del Software" la respuesta para superarla fue la adopción de modelos y metodologías clásicas que progresivamente fueron incorporando estándares, controles y formalidades al DS. En un afán que llegó a ser definido como "triángulo de hierro." La evolución no se detuvo, con la llegada del Internet surgen proyectos caracterizados por requerimientos cambiantes y tiempos de entregas breves para los que las metodologías existentes no se adaptaban idóneamente, surgen las metodologías ágiles, enfocadas en interacción equipo-usuarios, entregas tempranas y adaptación a los cambios; conviven con los esquemas tradicionales y agrupan a comunidades activas. Este esfuerzo documental busca reseñar de manera integral todo ese cambio evolutivo, por cuanto la mayoría de los trabajos en el área se enfocan en divulgar los métodos ágiles dejando subestimada su procedencia. (Zumba Gamboa, 2018)

Tipos

Existen dos tipos de metodologías, la metodología tradicional y la metodología ágil.

> Las **metodologías tradicionales** son denominadas, a veces, de forma despectiva, como metodologías pesadas. Centran su atención en llevar una documentación exhaustiva de todo el proyecto, la planificación y control del mismo, en especificaciones precisas de requisitos y modelado y en cumplir con un plan de trabajo, … imponen una disciplina rigurosa

de trabajo sobre el proceso de desarrollo de software, con el fin de conseguir un software más eficiente.
Un **modelo de desarrollo ágil**, generalmente es un proceso *Incremental* (entregas frecuentes con ciclos rápidos), también *Cooperativo* (clientes y desarrolladores trabajan constantemente con una comunicación muy fina y constante), *Sencillo* (el método es fácil de aprender y modificar para el equipo) y finalmente Adaptativo (capaz de permitir cambios de último momento).
Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan. (Maida & Pacienzia, 2015).

Algunas de las metodologías ágiles más conocidas son: Scrum, Agile, Extreme Programming, Open Unified Process, entre otras.

Para obtener más información sobre estas metodologías visite el siguiente enlace.

Metodologías ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos
Poca documentación	Documentación exhaustiva
Muchos ciclos de entrega	Pocos ciclos de entrega

Ilustración 6: Comparación entre metodología ágil y tradicional
Fuente: (Maida & Pacienzia, 2015)

Conclusiones y recomendaciones

A modo de conclusión, después de analizar los diferentes ciclos de vida, surge la pregunta ¿qué modelo de ciclo de vida se debe elegir?, no hay un ciclo de vida mejor que otro, el modelo debe elegirse determinando el que mejor se adapte al proyecto desarrollado, considerando aspectos como la complejidad, el tiempo de entrega, si los requerimientos son correctos o existen ambigüedades, entre otros factores.

Como recomendación, es importante conocer muy bien el proyecto a desarrollar (especialmente los requerimientos y el alcance) y ayudar al cliente a expresar que es lo que requiere obtener con el sistema de información, esto porque algunas veces el cliente no tiene claro que es lo que requiere.

Referencias bibliográficas

Patponto. (2010). Ingeniería de software. [Entrada de blog]. Recuperado de https://histinf.blogs.upv.es/2010/12/28/ingenieria-del-software/

Campderrich Falgueras, B. (2013). Ingeniería del software. Editorial UOC. https://elibro.net/es/ereader/usanmarcos/56294?page=16

Berzal, F. (s.f). El ciclo de vida de un sistema de información. Recuperado de http://flanagan.ugr.es/docencia/2005-2006/2/apuntes/ciclovida.pdf

Universidad del Papaloapan. (2015). Ciclo de vida del software. Recuperado de https://www.unpa.edu.mx/~blopez/Computacion/complementario/Anexo3_Ipcu097_01_CicloDeVidaDelSoftware.pdf

Maida, EG, Pacienzia, J. Metodologías de desarrollo de software [en línea]. Tesis de Licenciatura en Sistemas y Computación. Facultad de Química e Ingeniería “Fray Rogelio

Bacon". Universidad Católica Argentina, 2015. Disponible en: http://bibliotecadigital.uca.edu.ar/repositorio/tesis/metodologias-desarrollo-software.pdf

Zumba Gamboa, J. (2018). Evolución de las metodologías y modelos utilizados en el desarrollo de software. Revista Dialnet, 3(10), 20-33.

International Organization Standardization. (2017). ISO/IEC/IEEE 12207:2017 (en). Recuperado de https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:12207:ed-1:v1:en

García. A., Lameiro, M., Quattrocchio, F. (2014). Estándar Internacional ISO/IEC 12207 ciclo de vida del software. Recuperado de https://www.academia.edu/29836475/Est%C3%A1ndar_Internacional_ISO_IEC_12207_Ciclo_de_vida_del_software

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica