UF2406: El ciclo de vida del desarrollo de aplicaciones

Elaborado por: José Luis Ávila Jiménez

Edición: 5.1

EDITORIAL ELEARNING S.L.

ISBN: 978-84-16492-65-7

Impreso en España - Printed in Spain

Presentación

Identificación de la Unidad Formativa

Bienvenido a la Unidad Formativa **UF2406: El ciclo de vida del desarrollo de aplicaciones**. Esta Unidad Formativa pertenece al **Módulo Formativo MF0227_3: Programación orientada a objetos** que forma parte del Certificado de Profesionalidad **IFCD0112: Programación de lenguajes orientados a objetos y bases de datos relacionales**, de la familia de **Informática y comunicaciones**.

Presentación de los contenidos

La finalidad de esta Unidad Formativa es enseñar al alumno a implementar los componentes software encomendados, manipular bases de datos a través de interfaces para integrar el lenguaje de programación con el lenguaje de acceso a datos, probar los componentes software desarrollados, así como utilizar los componentes orientados a objeto y elaborar la documentación del código desarrollado según los estándares de la organización.

Para ello, se desarrollará el proceso de ingeniería del software, planificación y seguimiento, se realizará el diagramado, el desarrollo de la GUI, y por último, se analizará la calidad en el desarrollo del software, pruebas, excepciones y documentación.

Objetivos de la Unidad Formativa

Al finalizar esta Unidad Formativa aprenderás a:

- Manejar las herramientas de ingeniería de software.
- Verificar la corrección de las clases desarrolladas mediante la realización de pruebas.
- Elaborar la documentación completa relativa a las clases desarrolladas y pruebas realizadas.
- Realizar modificaciones de clases existentes por cambios en las especificaciones.
- Desarrollar interfaces de usuario en lenguajes de programación orientados a objeto, a partir del diseño detallado.

Índice

UD1. Proceso de ingeniería del software

Área: informática y comunicaciones

UD1

Proceso de ingeniería del software

1.1. Distinción de las fases del proceso de ingeniería software: especificación, diseño, construcción y pruebas unitarias, validación, implantación y mantenimiento

1.2. Análisis de los modelos del proceso de ingeniería: modelo en cascada, desarrollo evolutivo, desarrollos formarles, etc

1.3. Identificación de requisitos: concepto, evolución y trazabilidad

1.4. Análisis de metodologías de desarrollo orientadas a objeto

1.5. Resolución de un caso práctico de metodologías de desarrollo que utilizan UML

1.6. Definición del concepto de herramienta CASE

- 1.6.1. Herramientas de Ingeniería del Software
- 1.6.2. Entornos de desarrollo
- 1.6.3. Herramientas de prueba
- 1.6.4. Herramientas de gestión de configuración
- 1.6.5. Herramientas para métricas

1.1. Distinción de las fases del proceso de ingeniería software: especificación, diseño, construcción y pruebas unitarias, validación, implantación y mantenimiento

Desde el momento en el que se introdujeron computadores con capacidad para soportar aplicaciones de tamaño considerable en los años sesenta, se descubrió que las técnicas de desarrollo para los hasta entonces pequeños sistemas dejaron progresivamente de ser válidas.

Estas primitivas técnicas consistían básicamente en codificar y corregir, es decir, no existe necesariamente una especificación del producto final, en su lugar se tienen algunas anotaciones sobre las características generales del programa.

Inmediatamente al comienzo de un proyecto se empieza la codificación y simultáneamente se va depurando el programa resultante. Cuando el programa cumple con las especificaciones y parece que no tiene errores se entrega.

Las ventajas de esta forma de trabajar son que no se gasta tiempo en planificación, gestión de los recursos, documentación, etc.

En el caso de que el proyecto es de un tamaño muy pequeño y lo realiza una sola persona no es un mal sistema para producir un resultado pronto, aunque este enfoque no es muy adecuado cuando se trata de desarrollar un trabajo en equipo, como ocurren en el desarrollo de la mayoría de sistemas software

Hoy en día es un método de desarrollo que se usa cuando hay plazos muy breves para entregar el producto final y no existe una exigencia explícita por parte de la organización de usar alguna metodología de ingeniería del software. Puede dar resultado en algunas ocasiones pero la calidad es imprevisible.

Las consecuencias de este enfoque, que desembocaron en lo que se denomino la crisis del software, fueron:

– El costo de los proyectos era mucho mayor de lo originalmente previsto.

– El tiempo de desarrollo también excedía lo proyectado.

– La calidad del código producido era imprevisible y en promedio baja.

– Era prácticamente imposible mantener las aplicaciones así desarrolladas.

La Ingeniería del software surgió en aquella época como disciplina con el objetivo de idear métodos y técnicas que solucionaran estos problemas y proporcionaran un marco de trabajo técnico adecuado para llevar acabo la construcción del software.

La Ingeniería del Software se puede definir como aquella rama de las ciencias de la computación que trata del establecimiento de los principios y métodos de la ingeniería, orientados a obtener software económico, que sea fiable y funcione de manera eficiente sobre máquinas reales.

El software requiere de un tiempo y esfuerzo considerable para ser desarrollado, y durante aún más tiempo debe de estar en uso antes de ser retirado o substituido.

Durante todo este período de tiempo se identifican una serie de etapas que en su conjunto se denominan “ciclo de vida del software”.

Las etapas principales de cualquier ciclo de vida son las siguientes:

– Análisis: se identifican los requisitos que debe de cumplir el software y se construye un modelo de dichos requisitos.

– Diseño: A partir del modelo de análisis se identifican los procesos y las estructuras de datos en las que se descomponen el sistema, y además se construye un modelo del sistema a desarrollar.

– Codificación: se construye el sistema en sí mismo.

– Prueba: se comprueba que el sistema construido es correcto y cumple con el modelo de requisitos.

– Mantenimiento: esta fase tiene lugar tras la entrega del producto acabado y en ella se trata de asegurar que el sistema siga funcionando y adaptándose a nuevos requisitos.

Desde cualquiera de ellas se puede volver a la anterior si el desarrollo posterior detecta algún error cometido en las fases anteriores.

Dependiendo de la manera en que se estructuren estas etapas surgen los diversos ciclos de vida del software los cuales se pueden clasificar en tres tipos genéricos, ciclos de vida en cascada, ciclos de vida en espiral o incrementales y ciclos de vida Orientados a Objetos

1.2. Análisis de los modelos del proceso de ingeniería: modelo en cascada, desarrollo evolutivo, desarrollos formarles, etc

El ciclo de vida en cascada, inicialmente propuesto por Royce en 1970, fue el primer ciclo de vida que se propuso y es, actualmente el más ampliamente seguido por una multitud de organizaciones y empresas de desarrollo.

Este modelo tiene la posibilidad de hacer iteraciones o repeticiones, es decir, que si durante las modificaciones y cambios que se hacen en la fase de mantenimiento se puede detectar la necesidad de cambiar algo en el diseño, por ejemplo, lo cual significa que se van a hacer los cambios que sean necesarios en la codificación y se tendrán que realizar de nuevo las pruebas.

Sin embargo, si se tiene que volver a una de las fase anteriores al mantenimiento hay que realizar de nuevo el resto de las etapas hasta llegar al final.

Después de cada etapa se hace una revisión para chequear si se puede pasar a la siguiente etapa. En el se trabaja en base a documentos, es decir, la entrada y la salida de cada etapa es un tipo de documento específico.

Este ciclo de vida conlleva una serie de ventajas:

– La planificación del proyectoes sencilla y fácil de hacer.

– La calidad del producto si se aplica correctamente es alta.

– Permite trabajar con empleados con menor cualificación.

Sin embargo también presenta una serie de inconvenientes bastante graves que hacen que no se suela implementar “tal cual” en la realidad:

– Su mayor inconveniente es la necesidad de detallar todos los requisitos al comienzo del proyecto. Lo normal es que el cliente no tenga perfectamente claras las especificaciones del software que desea, o puede ser que surjan otras necesidades no previstas durante el proyecto.

– Si se cometen errores en una fase y no se detectan a tiempo es difícil volver atrás, ya que una vez que se ha finalizado una fase y se ha generado la documentación correspondiente, un paso atrás representa repetir la fase completamente

– No se desarrolla el producto hasta el final, esto quiere decir que si se tiene un fallo la fase de análisis este probablemente no será descubierto hasta la entrega, con el lo que conlleva un gasto inútil de recursos. Debido a esto el cliente no ve resultados hasta el final, con lo que puede impacientarse.

– No se tienen indicadores fiables del progreso del trabajo, lo cual puede llevar al síndrome del 90%, es decir, las tareas se indican como realizadas en un 90% pero el restante 10% va a necesitar de un esfuerzo considerablemente mayor que el resto.

Sin embargo fue el primer modelo de desarrollo de software que se planteó por lo que ha influenciado numerosos ciclos ce vida que se han propuesto posteriormente.

El ciclo de vida en cascada ha inspirado numeroso modelos de ciclos de vida, como el ciclo de vida en V, el modelo sashimi, o el ciclo de vida en espiral.

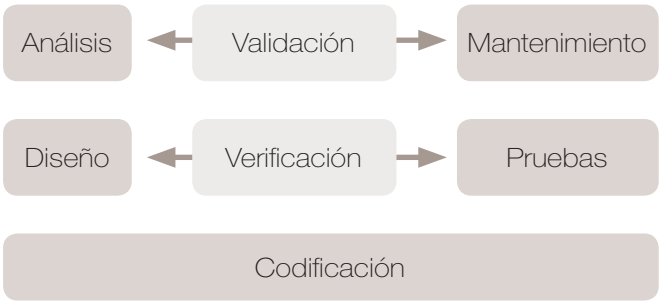
El ciclo de vida en V fue propuesto por Alan Davis, y tiene las mismas fases que el ciclo de vida en cascada pero teniendo en consideración en consideración el nivel de abstracción de cada una.

Se considera que la fase con mayor nivel de abstracción es la fase de análisis, para posteriormente pasar a trabajar a menos nivel en el diseño.

En la codificación se trabaja al mínimo nivel de abstracción. Posteriormente durante las distintas fases de prueba se va subiendo de nivel de abstracción

fase además de utilizarse como entrada para la siguiente, sirve para validar o verificar otras fases posteriores . La estructura de las tareas es la que se muestra en el esquema.

Una fase además de utilizarse como entrada para la siguiente, sirve para validar o verificar otras fases posteriores.



De esta forma la tarea de validación consiste en comprobar los resultados del análisis, es decir, si el software cumple con los requisitos que se le exigieron al principio del desarrollo.

Esto se hace durante la fase de mantenimiento en la que el usuario final durante su trabajo del día a día informa al desarrollador de aquellos aspectos que no cumplen con lo especificado y determinado durante el análisis.

De la misma forma surge el concepto de Verificación, en el que se comprueba que el software funciona correctamente acorde al diseño que se ha realizado.

Estos dos conceptos, verificación y validación son los mayores aportes de este modelo de ciclo de vida, y se han extendido a toda la ingeniería del software.

Definición

Podemos definir **verificación** como el proceso para determinar que un sistema software está libre de errores, y la validación como el proceso para determinar que un determinado sistema cumple con los requisitos esperados.

El modelo Sashimi es otro modelo de ciclos de vida. Si seguimos el modelo en cascada como fue definido, una fase sólo puede empezar cuando ha terminado la anterior.

En el caso de este ciclo de vida, sin embargo, se permite un solapamiento entre fases. Por ejemplo, sin tener terminado del todo el diseño se puede comenzar a implementar.

El nombre “sashimi” deriva del estilo de presentación en rodajas de pescado crudo en Japón.

Una ventaja de este modelo es que no necesita generar tanta documentación como el ciclo de vida en cascada puro debido a que se continúa con el mismo grupo de trabajo durante las distintas fases y por lo tanto conocen el proyecto en profundidad.

Los problemas que plantea este modelo de ciclo de vida son básicamente los mismos que el modelo de ciclo de vida en cascada, pero agravados, y son los siguientes:

– Es más difícil que en ciclo de vida clásico el controlar el progreso del proyecto, debido a la falta de puntos de referencia. Debido a que las fases se solapan constantemente, los finales de fase ya no son un punto de referencia específico.

– Al realizar las fases en paralelo, pueden ocurrir a menudo problemas de comunicación entre los miembros del equipo, de los que pueden surgir inconsistencias que necesiten de cambios y modificaciones alterando la planificación.

La fase de “concepto” se añade en este modelo de ciclo de vida y en ella se trata de definir los objetivos del proyecto, beneficios, tipo de tecnología y tipo de ciclo de vida.

La fase de diseño se divide a su vez en dos fases diferentes, el diseño arquitectónico y el diseño detallado o de componentes.

En la fase de diseño arquitectónico es diseño de alto nivel de abstracción, el detallado es de bajo nivel de abstracción, cuando se especifican en detalle cada uno de los componentes del software.

Al terminar una iteración se comprueba que lo que se ha realizado cumple con los requisitos que se establecieron al principio. También se verifica que funciona correctamente y es el propio cliente quien evalúa el producto para ver si es satisfactorio para resolver su necesidad.

En el ciclo de vida Sahimi no existe una diferencia muy clara entre cuándo termina el proyecto y cuándo empieza la fase de mantenimiento ya que cuando hay que hacer un cambio, éste puede consistir en un nuevo ciclo.

Presenta numerosas ventajas en cuanto a su utilización:

– No necesita una definición detallada de los requisitos para empezar a funcionar.

– Al entregar productos desde el final de la primera iteración es más fácil validar los requisitos frente a la solicitud del usuario.

– El riesgo en general es menor porque, si todo se hace mal, sólo se pierde el tiempo y recursos invertidos en una iteración (las anteriores iteraciones están bien por definición).

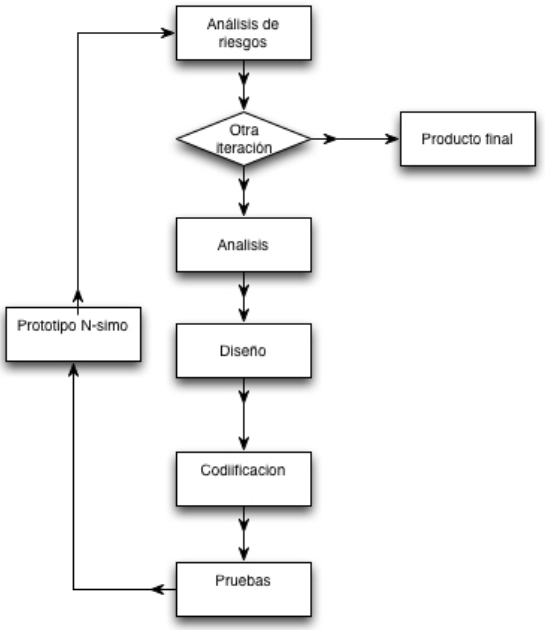
– El riesgo de sufrir retrasos es menor, ya que se identifican los problemas en etapas tempranas cuando aun hay tiempo de subsanarlos.

Sin embargo también presenta algunos inconvenientes que conviene tener en cuenta si se decide optar por el a la hora de realizar un proyecto:

– Es difícil llevar a cabo una evaluación correcta de los riesgos. Por el propio concepto de riesgo, este lleva implicado una dosis de incertidumbre que limita el hecho de poder realizar una estimación adecuada.

– Necesita de la participación continua de la parte cliente, esfuerzo al que algunos clientes pueden no estar de acuerdo en realzar, por lo que convienen aclarar cual va a ser su participación antes de comenzar.

Los tipos de ciclos de vida que se han visto hasta ahora se refieren al análisis y diseño estructurados, pero hay que tener en cuenta que el desarrollo de sistemas orientados a objetos tiene la particularidad de estar basados en un diseñado de componentes que se relacionan entre sí a través de una serie de interfaces, o lo que es lo mismo, son más modulares y por lo tanto el trabajo se puede particionar en un conjunto de pequeños proyectos o miniproyectos.



Esquema de ciclo de vida incremental

Además, hoy en día se trata de tender a reducir los riesgos y, en este sentido, el ciclo de vida en cascada no proporciona muchas ventajas. Debido a todo esto, el ciclo de vida típico en una metodología de diseño orientado a objetos alguna variación del ciclo de vida en espiral.

Un ejemplo de ciclo de vida Orientado a Objetos es elllamado “modelo fuente”, que fue desarrollado por Henderson-Sellers y Edwards en 1990.Es un tipo de ciclo de vida pensado para ser aplicado siguiendo el paradigma de la orientación a objetos y posiblemente el más seguido con la ventaja de que permite un desarrollo solapado e iterativo.

Un proyecto en modelo fuente se divide en las siguientes fases:

– Planificación del negocio.

– Construcción: Es la más importante y se subvidide a su vez en otras tantas actividades: Planificación, Investigación, Especificación, Implementación y Revisión.

– Entrega o ”liberación”.

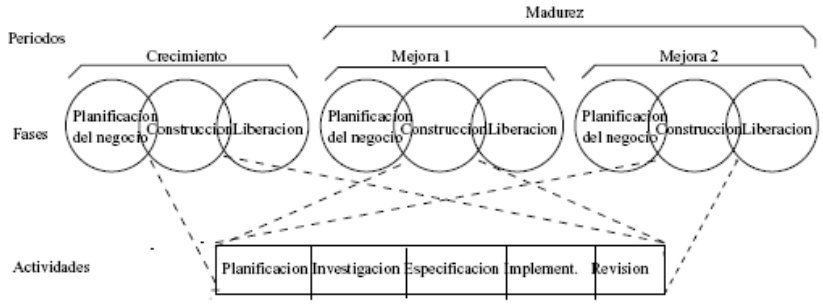
La primera y la tercera fase son independientes de la metodología de desarrollo orientado a objetos. Que se utilice Además de las tres fases, existen dos periodos:

– Crecimiento: Es el tiempo durante el cual se esta construyendo el sistema.

– Madurez: Es el periodo de mantenimiento del producto. En el cada mejora se planifica igual que el periodo anterior, o sea, llevando a cabo las fases de Planificación del negocio, Construcción y Entrega.

Cada una de las clases de la aplicación desarrollada puede tener un ciclo de vida propio debido a que cada clase puede estar en una fase diferente en un momento cualquiera.

Un esquema de su estructura es el siguiente:



Esquema de ciclo de vida orientado a objetos

La fase de análisis de requisitos comienza tras el análisis del sistema donde se va a encuadrar el producto que se va desarrollar y tiene como objetivo crear un modelo de los requisitos que debe de cumplir el software. Este modelo se plasmará en un documento, la "especificación de requisitos del software" que será el producto final de esta fase y el punto de comienzo de la siguiente fase, el diseño.

El análisis de requisitos podemos subdividirlo en tres partes, la búsqueda de requisitos y el modelado de requisitos.

La búsqueda de requisitos tiene como objetivo descubrir las verdaderas necesidades del cliente que ha encargado el desarrollo del sistema software.

En la mayoría de las ocasiones el cliente que desea un desarrollo no tiene al comienzo muy claro que producto necesita. En otras ocasiones puede tener claro que es lo que quiere pero la labor del ingeniero del software es determinar que es lo que necesita.

Para ello se pueden utilizar varias técnicas, como las que se definen a continuación.

1.3. Identificación de requisitos: concepto, evolución y trazabilidad

Entrevistas: reuniones entre el cliente y el equipo desarrollador, en la que se determinan los requisitos del sistema.

Estas siempre la tendremos, al menos al inicio del proyecto, aunque conviene que se hagan con cierta frecuencia para que se expliciten los requisitos y estos se refinen. Conviene que las entrevistas no solamente sean con la alta dirección o la gerencia, sino que en ella se involucren otros actores, preferiblemente los usuarios que van a trabajar con la aplicación final y que conocerán mucho menos los rusitos de esta.

Desarrollo conjunto de aplicaciones (JAD): Es un tipo de entrevista con muchos participantes desarrollada por IBM que se apoya en la dinámica de grupos. LaPlanificación conjunta de requisitos (JRP) es un subconjunto de las sesiones JAD, dirigidas a la alta dirección y los productos que resultan de ellas son los requisitos de alto nivel o estratégicos.