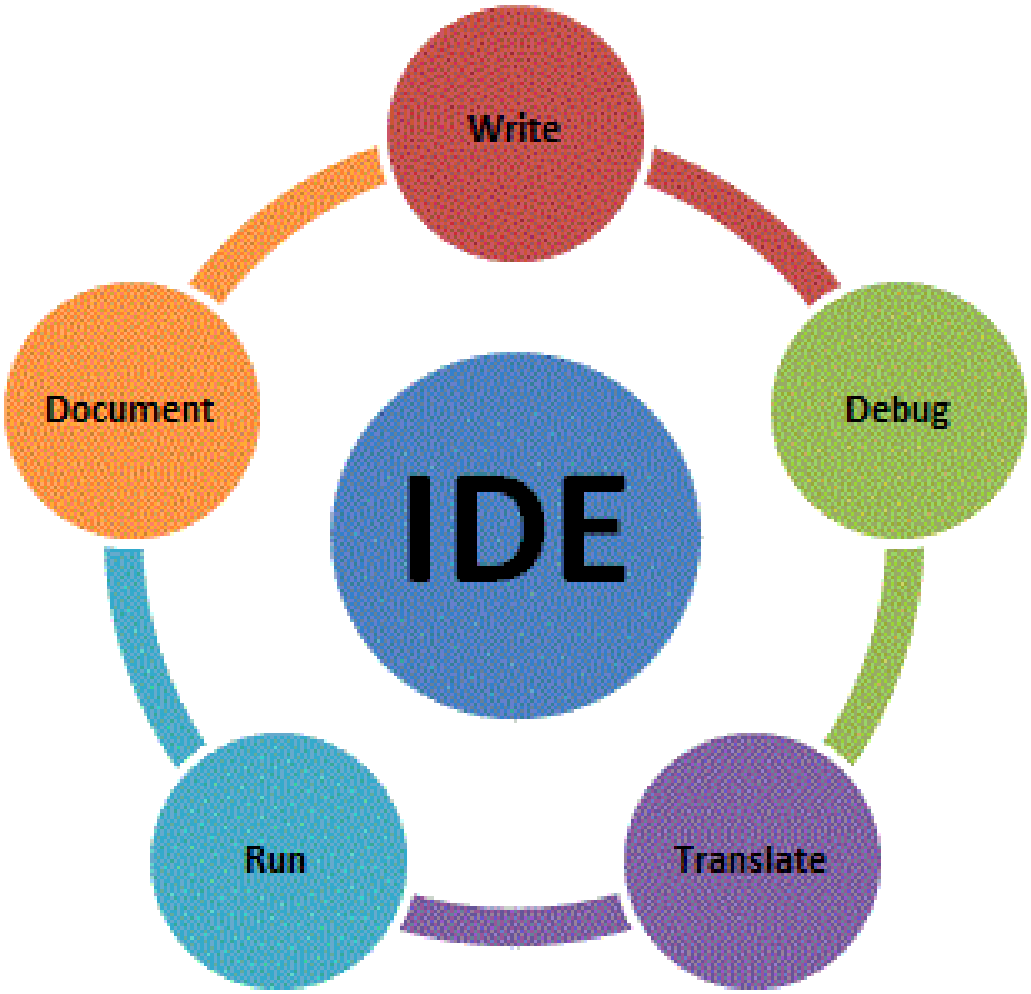

UNIDAD 1. DESARROLLO SOFTWARE

Anexo I: Fases del ciclo de vida

1. INTRODUCCIÓN

Antes de desarrollar un producto de software hay que elegir un modelo de ciclo de vida. Será necesario examinar las características del proyecto para elegir un modelo u otro. Independientemente del modelo elegido, hay unas etapas que se deben cumplir para construir un proyecto de calidad y que se nombraron en el apartado anterior.

Entramos a ver ahora más detenidamente cada una de esas etapas.

2. FASE DE ANÁLISIS

Lo más importante para el éxito de un proyecto es entender y comprender el problema que se necesita resolver para una vez comprendido darle solución. En este punto se especifican los requisitos o capacidades que el sistema debe tener porque el cliente así lo ha pedido. Esto no suele ser tarea fácil ya que el cliente puedo no tenerlo claro, pueden surgir nuevos requisitos, pueden cambiar, pueden existir malos entendidos, Etc. Para realizar un trabajo satisfactorio es necesario obtener unos buenos requisitos, para lo que es necesario una buena comunicación entre el cliente y los desarrolladores. Para ello, se pueden utilizar varias técnicas cómo las que se describen a continuación:

- Entrevista: Técnica habitual que consiste en hablar con el cliente
- Desarrollo de conjunto de aplicaciones (JAD): Dinámica de grupo muy estructurada donde cada persona juega un rol muy concreto.
- Planificación conjunta de requisitos (JRP): Es como el JAD, pero dirigidas a alta dirección y en consecuencia a productos de alto nivel o estratégicos.
- Brainstorming: Es un tipo de reuniones en grupo cuyo objetivo es generar ideas desde diferentes puntos de vista para la resolución de un problema. Es un uso adecuado al inicio del proyecto para exportar un problema desde muchos puntos de vista.
- Prototipos: es una versión inicial del sistema, se usa para clarificar algunos puntos.

- Caso de usos: es la técnica definida en UML. Se basa en escenarios que describen cómo se usa el software en determinada situación.

Se especifican dos tipos de requisitos:

- Requisitos funcionales: Describen con detalle la función que realiza el sistema, como reacciona ante determinadas entradas, cómo se comporta en situaciones particulares, etc. Serían ejemplos de éstos: que el usuario pueda agregar un nuevo contacto, eliminarlo, ver el contacto, pueda imprimir una lista de contacto, …
- Requisitos no funcionales: Tratan sobre las características del sistema, como puede ser la fiabilidad, mantenibilidad, sistema operativo, plataforma hardware, restricciones, limitaciones, por ejemplo, que la aplicación pueda funcionar en distintos sistemas operativos, el tiempo de respuesta a consultas, que la interfaz de usuario sea a través de ventanas, que sea intuitiva, etc.

Para representar los requisitos se utilizan distintas técnicas:

- DIAGRAMAS DE FLUJO DE DATOS (DFD). Es un diagrama que representa el flujo de datos entre los distintos procesos, entidades externas y almacenes que forman el sistema. Los procesos identifican funciones dentro del sistema, se representa mediante burbujas ovaladas o circulares. Las entidades externas representan componentes que no forman parte del sistema (una persona, un departamento) pero proporcionan datos al sistema o los reciben de él, se representan mediante rectángulos. Los almacenes representan los datos desde el punto de vista estático, es decir, representan el lugar donde se almacenan los datos procesados o desde donde se recuperan para apoyar el proceso; se representa gráficamente mediante dos líneas horizontales y paralelas. Por último, el flujo de datos representa el movimiento de datos dentro del sistema, se representa mediante flechas.
- DIAGRAMAS DE FLUJO DE CONTROL (DFC): similar a los anteriores con la diferencia de que muestra el flujo de control en lugar de datos.
- DIAGRAMAS DE TRANSICIÓN DE ESTADOS (DTE). Representa cómo se comporta el sistema como consecuencia de sucesos externos.
- DIAGRAMA DE ENTIDAD/RELACIÓN. Se usa para representar datos y la forma en la que se relacionan entre ellos.

- DICCIONARIO DE DATOS (DD) es una descripción detallada de los datos utilizados por el sistema que gráficamente se encuentra representado por los flujos de datos y almacenes presentes sobre el conjunto de DFD.

Todo lo realizado en esta fase de análisis debe quedar reflejado en un documento de Especificación de Requisitos del Software (ERS), este documento no puedes ser ambiguo, debe ser completo, consistente, fácil de verificar y modificar, fácil de utilizar en la fase de explotación y mantenimiento, y fácil de identificar el origen y las consecuencias de los requisitos. La estructura de este documento ERS propuesta por el IEEE es la siguiente:

1. Introducción
    1. Propósito.
    2. Ámbito del Sistema.
    3. Definiciones, Acrónimos y Abreviaturas.
    4. Referencias
    5. Visión general
2. Descripción General
    1. Perspectiva del producto.
    2. Funciones del producto
    3. Características de los usuarios
    4. Restricciones
    5. Suposiciones y Dependencias
    6. Requisitos futuros
3. Requisitos específicos
    1. Interfaces externas
    2. Funciones
    3. Requisitos de rendimiento
    4. Restricciones del diseño
    5. Atributos del sistema
    6. Otros requisitos
4. Apéndices

Se puede consultar el documento https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf para ver qué es lo que se desarrolla en cada aparatado.

3. FASE DE DISEÑO

Una vez identificado los requisitos, es necesario componer la forma en que se solucionará el problema. En esta etapa se traducen los requisitos funcionales y no funcionales en una representación de software.

Principalmente hay dos tipos de diseño, el diseño estructurado que está basado en el flujo de los datos a través del sistema, y el diseño orientado a objetos donde el sistema se entiende como un conjunto de objetos que tienen propiedades y comportamientos, además de eventos que activan operaciones que modifican el estado de los objetos; los objetos interactúan de manera formal con otros objetos.

El diseño estructurado (diseño clásico) produce un modelo de diseño con 4 elementos:

- Diseño de datos: Está basado en los datos y las relaciones definidos en el diagrama entidad/relación y en la descripción detallada de los datos.
- Diseño arquitectónico: Se centra en la representación de la estructura de los componentes del software. Partiendo de los DFD se establece la estructura modular del software a desarrollar
- Diseño de interfaz: El resultado de esta tarea es la creación de formatos de pantalla.
- Diseño a nivel de componentes (diseño procedimental): Transforma los elementos estructurales en una descripción procedimental de los componentes del software.

Las bases para el diseño a nivel de componentes quedaron establecidas cuando se propuso un conjunto de construcciones lógicas con las que podía formarse cualquier programa. Dichas construcciones son: secuencial, condicional y repetitiva, y que son fundamentales en la programación estructurada.

- La construcción secuencial: implementa los pasos del proceso esenciales para la especificación de cualquier algoritmo
- La construcción condicional: permite seleccionar un proceso u otro tras la evaluación de una condición lógica. Dentro de esta construcción encontramos la selección múltiple.
- La repetitiva: proporciona los bucles.

Notaciones gráficas para el diseño

Para representar el diseño se usan herramientas gráficas como el diagrama de flujo, diagramas de caja, tablas de decisión o pseudocódigos.

1. Diagrama de flujo es una herramienta muy usada en el diseño procedimental. Usa los símbolos:
    - caja para indicar paso del proceso
    - rombo para representar condición lógica
    - flechas para flujo de control.

1. Pseudocódigo: Usa texto descriptivo para realizar el diseño de un algoritmo. A primera vista parece un lenguaje de programación, ya que mezcla el lenguaje natural con las estructuras sintácticas que incluyen palabras clave que permiten construir las estructuras básicas de la programación estructurada. Depende mucho de quién lo escribe ya que no hay estándar definido. Al no ser lenguaje de programación no puede ser compilado.

3. Diagramas de cajas: el elemento fundamental es la caja:
    - Para representar una secuencia se conectan varias cajas seguidas.
    - Para una condicional se representa una caja para la parte Si y otra para la parte No, encima se indica la condición.
    - En las repetitivas, el proceso a repetir se encierra en una caja que está dentro de otra caja donde en la parte superior (do- while) o inferior (repeat-until) se indica la condición del bucle.
    - Por último, en la selección múltiple en la parte superior se indica el caso de condición, se define tantas columnas como valores se vayan a comprobar en la condición debajo de cada valor se indica la parte a realizar.

        Os dejo el link donde podréis ver algunos ejemplos:http://kimberlingvaldez.blogspot.com/.

4. Tablas de decisión: es una herramienta que sirve para representar de una manera más fácil la lógica de un problema cuando ésta es más o menos complicada. Para ello, hay que identificar en el problema las acciones que hay que ejecutar y las condiciones que se deben cumplir para ejecutar esas acciones.

Partes de una tabla:

- Conjunto de condiciones: que intervienen en el problema. Entrada de condiciones posibles: si, no, da igual.
- Conjunto de acciones: Abarca las acciones que se tienen que ejecutar cuando se cumple un conjunto dado de condiciones.
- Salida de ejecución: determina cuándo se ejecuta la acción

Regla de decisión: Es una combinación de un estado en la entrada de condiciones y de una o más acciones asociadas a la parte de salida.

CONDICIÓN	REGLA DECISIÓN
CONDICIONES	ENTRADA DE ACCIONES
ACCIONES	ENTRADA DE CONDICIONES

Cada columna de los cuadrantes de la derecha forma una regla de procesamiento, las reglas establecen las acciones a realizar con cada combinación de condiciones. Para construir una tabla se realizan los siguientes pasos:

1. Hacer una lista de todas las acciones y todas las condiciones.
2. Asociar conjuntos específicos de condiciones con acciones específicas, eliminando combinaciones imposibles de condiciones.
3. Determinar las reglas indicando que acción o acciones ocurren para un conjunto de condiciones.

Tipos de tablas de decisión:

1. Según el número de valores que puedan tomar sus condiciones:
    a. Tablas de decisión binaria: Todas las condiciones son binarias, la evaluación de todas las condiciones está limitada a dos valores posibles. (s/n o 0/1).
    b. Tablas de decisión múltiples: Todas sus condiciones pueden tomar más de dos valores. También se denominan ampliadas o extendidas.
    c. Tablas de decisión mixtas: Intervienen condiciones binarias y múltiples.

2. Según se encadene o no con otras tablas:
    a. Tablas abiertas: cuando sus acciones tienen referencia en otra tabla de decisión.
    b. Tablas cerradas: una vez ejecutada la tabla llamada, devuelve el control a la tabla que lo llamó.

4. FASE DE CODIFICACIÓN

Durante esta etapa se realizan las tareas de programación que consiste en llevar a código fuente, en el lenguaje de programación elegido, todo lo diseñado anteriormente.

Esta tarea la realiza el programador siguiendo los requisitos del diseño y teniendo siempre en consideración los requisitos funcionales y no funcionales especificados en la fase de análisis.

Si las fases anteriores se han llevado a cabo de manera eficiente y metódica, será relativamente fácil y rápido llevar a cabo la codificación. Teniendo siempre en cuenta que cuanto menor sea el nivel del lenguaje en el que codifiquemos mayor será el tiempo de programación requerido, se tardará más en codificar un algoritmo en lenguaje ensamblador que en lenguaje C.

Mientras se programa la aplicación, sistema o software se van realizando tareas de depuración, ya que se va liberando al código de los fallos factibles de ser hallados en esta fase (semántica, sintáctica, lógica). Por lo que existe un solapamiento con la fase siguiente.

5. FASE DE PRUEBAS

En esta etapa ya tenemos el software y se trata de encontrar posibles errores, no solo de codificación sino también relativos a la especificación y diseño. Durante esta fase se realizan tareas de

- VERIFICACIÓN para comprobar si se está construyendo el producto correctamente
- VALIDACIÓN para comprobar que el producto se adapta a los requisitos del cliente.

Para construir las pruebas los pasos son los siguientes:

1. Hay que generar un plan de pruebas partiendo de la documentación del proyecto y de la del producto a probar.
2. A partir del plan se diseñan pruebas.
3. Generación de los casos de prueba, se confeccionan distintos casos según la técnica indicada previamente
4. Definición de los procedimientos de prueba hay que especificar cómo se va a llevar a cabo el proceso, quién lo va a realizar, cuándo, etc
5. Ejecución de las pruebas
6. Evaluación, Se identifican los posibles errores. Es necesario realizar un informe con el resultado de ejecución.
7. Depuración
8. Análisis de errores

DISEÑO DE CASOS DE PRUEBAS

- DISEÑO PRUEBA DE CAJA BLANCA: se centra en validar la estructura interna del programa (necesita conocer los detalles procedimentales del código)
- DISEÑO PRUEBA DE CAJA NEGRA: se centran en validar los requisitos funcionales sin fijarse en el procedimiento interno del programa.

Estas pruebas no son excluyentes y se pueden realizar las dos para detectar diferentes tipos de errores.

6. FASE DE DOCUMENTACIÓN

Todas las etapas del desarrollo deben quedar documentadas. En esta etapa es necesario reunir todos los documentos generados y clasificarlos según el nivel técnico de sus descripciones. En general, se divide en dos clases:

1. Documentación del proceso: desarrollo y mantenimiento
2. Documentación del producto: describe el producto desarrollado. Dentro de este se definen dos partes
    1. Documentación del sistema: que describe el producto desde el punto de vista técnico y que está orientado al desarrollo y mantenimiento del software

2. Documentación del usuario: ofrece una descripción del producto orientada a los usuarios que utilizarán el sistema

7. FASE DE EXPLOTACIÓN Y MANTENIMIENTO

En la fase de explotación se lleva a cabo la instalación y puesta en marcha del producto software en el entorno de trabajo del cliente

En la fase de mantenimiento se puede llevar a cabo desde cuatro tipos, dependiendo de la demanda del cliente:

- mantenimiento adaptativo, modifica el producto por los cambios de hardware y software del entorno en el que se ejecuta.
- mantenimiento correctivo es probable que después de la entrega al cliente este encuentre errores este tipo de mantenimiento se encarga de corregirlos.
- mantenimiento perfectivo modifica el producto con nuevas funcionalidades
- mantenimiento preventivo hace cambios en programas con el fin de poder corregir, adaptar y mejorar más fácilmente.