5.- Fases en el desarrollo y ejecución del software.

Caso práctico
En la reunión de BK acerca del nuevo proyecto Ada, la supervisora, dejó bien claro que lo primero y más importante es tener claro qué queremos que haga el software y con qué herramientas contamos: lo demás vendría después, ya que si esto no está bien planteado, ese error se propagará a todas las fases del proyecto.
—¿Por dónde empezamos? —pregunta Juan.
—ANÁLISIS de REQUISITOS —contesta Ada.

Ya hemos visto en puntos anteriores que debemos elegir un modelo de ciclo de vida para el desarrollo de nuestro software.

Independientemente del modelo elegido, siempre hay una serie de etapas que debemos seguir para construir software fiable y de calidad.

Estas etapas son:

1. **ANÁLISIS DE REQUISITOS.**

   Se especifican los requisitos funcionales y no funcionales del sistema.

2. **DISEÑO.**

   Se divide el sistema en partes y se determina la función de cada una.

3. **CODIFICACIÓN.**

   Se elige un Lenguajes de Programación y se codifican los programas.

4. **PRUEBAS.**

   Se prueban los programas para detectar errores y se depuran.

5. **DOCUMENTACIÓN.**

   De todas las etapas, se documenta y guarda toda la información.

6. **EXPLOTACIÓN.**

   Instalamos, configuramos y probamos la aplicación en los equipos del cliente.

7. **MANTENIMIENTO.**

   Se mantiene el contacto con el cliente para actualizar y modificar la aplicación el futuro.

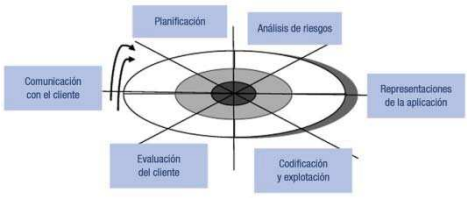
Autoevaluación

¿Crees que debemos esperar a tener completamente cerrada una etapa para pasar a la siguiente?

Sí.

No.

5.1.- Análisis.



Esta es la primera fase del proyecto. Una vez finalizada, pasamos a la siguiente (diseño).

Es la fase de mayor importancia en el desarrollo del proyecto y todo lo demás dependerá de lo bien detallada que esté. También es la más complicada, ya que no está automatizada y depende en gran medida del analista que la realice.

Es la primera etapa del proyecto, la más complicada y la que más depende de la capacidad del analista.

¿Qué se hace en esta fase?

Se especifican y analizan los requisitos funcionales y no funcionales del sistema.

Requisitos:

- **Funcionales**: Qué funciones tendrá que realizar la aplicación. Qué respuesta dará la aplicación ante todas las entradas. Cómo se comportará la aplicación en situaciones inesperadas.
- **No funcionales**: Tiempos de respuesta del programa, legislación aplicable, tratamiento ante la simultaneidad de peticiones, etc.

Lo fundamental es la buena comunicación entre el analista y el cliente para que la aplicación que se va a desarrollar cumpla con sus expectativas.

La culminación de esta fase es el documento ERS (Especificación de Requisitos Software).

En este documento quedan especificados:

- La planificación de las reuniones que van a tener lugar.
- Relación de los objetivos del usuario cliente y del sistema.
- Relación de los requisitos funcionales y no funcionales del sistema.
- Relación de objetivos prioritarios y temporización.
- Reconocimiento de requisitos mal planteados o que conllevan contradicciones, etc.

Citas para pensar

Todo aquello que no se detecte, o resulte mal entendido en la etapa inicial provocará un fuerte impacto negativo en los requisitos, propagando esta corriente degradante a lo largo de todo el proceso de desarrollo e **incrementando su perjuicio cuanto más tardía sea su detección***(Bell y Thayer 1976)(Davis 1993).*

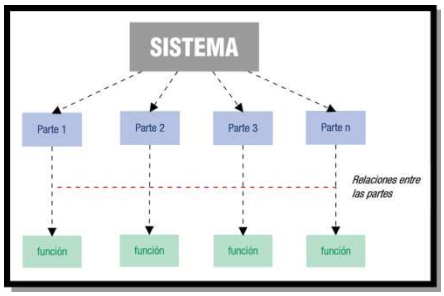
Como ejemplo de requisitos funcionales, en la aplicación para nuestros clientes de las tiendas de cosmética, habría que considerar:

- Si desean que la lectura de los productos se realice mediante códigos de barras.
- Si van a detallar las facturas de compra y de qué manera la desean.
- Si los trabajadores de las tiendas trabajan a comisión, tener información de las ventas de cada uno.
- Si van a operar con tarjetas de crédito.
- Si desean un control del stock en almacén.
- Etc.

5.2.- Diseño.

Caso práctico
Juan está agobiado por el proyecto. Ya han mantenido comunicaciones con el cliente y saben perfectamente qué debe hacer la aplicación. También tiene una lista de las características hardware de los equipos de su cliente y todos los requisitos. Tiene tanta información que no sabe por dónde empezar.
Decide hablar con Ada. Su supervisora, amable como siempre, le sugiere que empiece a dividir el problema en las partes implicadas.
—Vale, Ada, pero, ¿cómo lo divido?

Durante esta fase, donde ya sabemos lo que hay que hacer, el siguiente paso es ¿Cómo hacerlo?



Se debe dividir el sistema en partes y establecer qué relaciones habrá entre ellas.

Decidir qué hará exactamente cada parte.

En definitiva, debemos crear un modelo funcional-estructural de los requerimientos del sistema global, para poder dividirlo y afrontar las partes por separado.

En este punto, se deben tomar decisiones importantes, tales como:

- Entidades y relaciones de las bases de datos.
- Selección del lenguaje de programación que se va a utilizar.
- Selección del Sistema Gestor de Base de Datos.
- Etc.

Citas para pensar
Design is not just what it looks like and feels like. Designishowitworks.Steve Jobs ("*El diseño no es sólo lo que parece y cómo parece. Diseño es cómo se trabaja*").

Reflexiona
Según estimaciones, las organizaciones y empresas que crecen más son las que más dinero invierten en sus diseños.

5.3.- Codificación. Tipos de código.

Caso práctico
En BK, ya tienen el proyecto dividido en partes.
Ahora llega una parte clave: codificar los pasos y acciones a seguir para que el ordenador los ejecute. En otras palabras, programar la aplicación. Saben que no será fácil, pero afortunadamente cuentan con herramientas CASE que les van a ser de gran ayuda. A Ana el gustaría participar, pero cuando se habla de "código fuente", "ejecutable", etc. sabe que no tiene ni idea y que no tendrá más remedio que estudiarlo si quiere colaborar en esta fase del proyecto.

Durante la fase de codificación se realiza el proceso de programación.

Consiste en elegir un determinado lenguaje de programación, codificar toda la información anterior y llevarlo a código fuente.

Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.

Las características deseables de todo código son:

1. Modularidad: que esté dividido en trozos más pequeños.
2. Corrección: que haga lo que se le pide realmente.
3. Fácil de leer: para facilitar su desarrollo y mantenimiento futuro.
4. Eficiencia: que haga un buen uso de los recursos.
5. Portabilidad: que se pueda implementar en cualquier equipo.

Durante esta fase, el código pasa por diferentes estados:

- **Código Fuente**: es el escrito por los programadores en algún editor de texto. Se escribe usando algún lenguaje de programación de alto nivel y contiene el conjunto de instrucciones necesarias.
- **Código Objeto**: es el código binario resultado de compilar el código fuente.

  La compilación es la traducción de una sola vez del programa, y se realiza utilizando un compilador. La interpretación es la traducción y ejecución simultánea del programa línea a línea.

  El código objeto no es directamente inteligible por el ser humano, pero tampoco por la computadora. Es un código intermedio entre el código fuente y el ejecutable y sólo existe si el programa se compila, ya que si se interpreta (traducción línea a línea del código) se traduce y se ejecuta en un solo paso.

- **Código Ejecutable**: Es el código binario resultante de enlazar los archivos de código objeto con ciertas rutinas y bibliotecas necesarias. El sistema operativo será el encargado de cargar el código ejecutable en memoria RAM y proceder a ejecutarlo. También es conocido como código máquina y ya sí es directamente inteligible por la computadora.

Los programas interpretados no producen código objeto. El paso de fuente a ejecutable es directo.

5.4.- Fases en la obtención de código.

Caso práctico
Juan y María ya han decidido el Lenguajes de Programación que van a utilizar.
Saben que el programa que realicen pasará por varias fases antes de ser implementado en los equipos del cliente. Todas esas fases van a producir transformaciones en el código. ¿Qué características irá adoptando el código a medida que avanza por el proceso de codificación?

5.4.1.- Fuente.

El código fuente es el conjunto de instrucciones que la computadora deberá realizar, escritas por los programadores en algún lenguaje de alto nivel.

Este conjunto de instrucciones no es directamente ejecutable por la máquina, sino que deberá ser traducido al lenguaje máquina, que la computadora será capaz de entender y ejecutar.

Un aspecto muy importante en esta fase es la elaboración previa de un algoritmo, que lo definimos como un conjunto de pasos a seguir para obtener la solución del problema. El algoritmo lo diseñamos en pseudocódigo y con él, la codificación posterior a algún Lenguaje de Programación determinado será más rápida y directa.

Para obtener el código fuente de una aplicación informática:

1. Se debe partir de las etapas anteriores de análisis y diseño.
2. Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
3. Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.
4. Se procederá a la codificación del algoritmo antes diseñado.

La culminación de la obtención de código fuente es un documento con la codificación de todos los módulos (*Cada parte, con una funcionalidad concreta, en que se divide una aplicación*), funciones (*Parte de código muy pequeña con una finalidad muy concreta.*), bibliotecas y procedimientos (*Igual que las función, pero al ejecutarse no devuelven ningún valor.*) necesarios para codificar la aplicación.

Puesto que, como hemos dicho antes, este código no es inteligible por la máquina, habrá que TRADUCIRLO, obteniendo así un código equivalente pero ya traducido a código binario que se llama código objeto. Que no será directamente ejecutable por la computadora si éste ha sido compilado.

Un aspecto importante a tener en cuenta es su licencia. Así, en base a ella, podemos distinguir dos tipos de código fuente:

- Código fuente abierto. Es aquel que está disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
- Código fuente cerrado. Es aquel que no tenemos permiso para editarlo.

Autoevaluación
Para obtener código fuente a partir de toda la información necesaria del problema:

☐ Se elige el Lenguaje de Programación más adecuado y se codifica directamente.

☐ Se codifica y después se elige el Lenguaje de Programación más adecuado.

Se elige el Lenguaje de Programación más adecuado, se diseña un algoritmo y se codifica.
Muy bien. El diseño del algoritmo (los pasos a seguir) nos ayudará a que la codificación posterior se realice más rápidamente y tenga menos errores.

5.4.2.- Objeto.

El código objeto es un código intermedio.

Es el resultado de traducir código fuente a un código equivalente formado por unos y ceros que aún no puede ser ejecutado directamente por la computadora.

Es decir, es el código resultante de la compilación del código fuente.

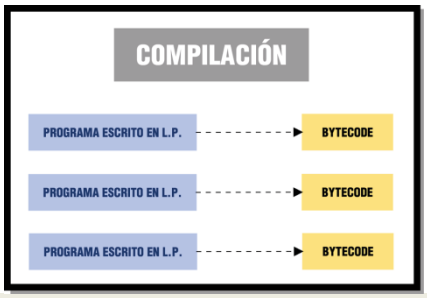
Consiste en un bytecode (*Código binario resultante de la traducción de código de alto nivel que aún no puede ser ejecutado.*) que está distribuido en varios archivos, cada uno de los cuales corresponde a cada programa fuente compilado.

Sólo se genera código objeto una vez que el código fuente está libre de errores sintácticos y semánticos.

El proceso de traducción de código fuente a código objeto puede realizarse de dos formas:

a. **Compilación**: El proceso de traducción se realiza sobre todo el código fuente, en un solo paso. Se crea código objeto que habrá que enlazar. El software responsable se llama compilador (*Software que traduce, de una sola vez, un programa escrito en un lenguaje de programación de alto nivel en su equivalente en lenguaje máquina*.).
b. **Interpretación**: El proceso de traducción del código fuente se realiza línea a línea y se ejecuta simultáneamente. No existe código objeto intermedio. El software responsable se llama intérprete (*Software que traduce, instrucción a instrucción, un programa escrito en un lenguaje de alto nivel en su equivalente en lenguaje máquina*). El proceso de traducción es más lento que en el caso de la compilación, pero es recomendable cuando el programador es inexperto, ya que da la detección de errores es más detallada.

El código objeto es código binario, pero no puede ser ejecutado por la computadora



Para saber más
En el siguiente enlace podrás visitar una página web, que te permitirá aprender más acerca de la generación de códigos objeto:
http://www.monografias.com/trabajos11/compil/compil2.shtml#co

5.4.3.- Ejecutable.

El código ejecutable, resultado de enlazar los archivos de código objeto, consta de un único archivo que puede ser directamente ejecutado por la computadora. No necesita ninguna aplicación externa. Este archivo es ejecutado y controlado por el sistema operativo.

Para obtener un sólo archivo ejecutable, habrá que enlazar todos los archivos de código objeto, a través de un software llamado linker (*Enlazador. Pequeño software encargado de unir archivos para generar un programa ejecutable.*) y obtener así un único archivo que ya sí es ejecutable directamente por la computadora.

Para saber más

En el siguiente enlace podrás visitar una página web, que te permitirá aprender más acerca de la generación de ejecutables:

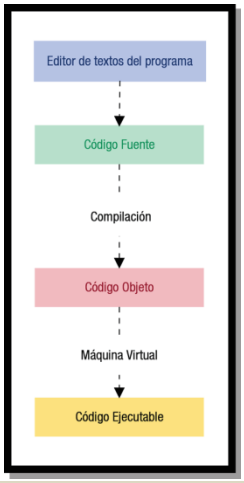
http://www.palomatica.info/juckar/sistemas/software/generacion_de_ejecutable.html

En el esquema de generación de código ejecutable, vemos el proceso completo para la generación de ejecutables.

A partir de un editor, escribimos el lenguaje fuente con algún Lenguaje de programación. (En el ejemplo, se usa Java).

A continuación, el código fuente se compila obteniendo código objeto o bytecode.

Ese bytecode, a través de la máquina virtual (se verá en el siguiente punto), pasa a código máquina, ya directamente ejecutable por la computadora.



Autoevaluación

Relaciona los tipos de código con su característica más relevante, escribiendo el número asociado a la característica en el hueco correspondiente.

Ejercicio de relacionar

Tipo de código.	Relación.	Características.
Código Fuente	2	1. Escrito en Lenguaje Máquina pero no ejecutable.
Código Objeto	1	2. Escrito en algún Lenguaje de Programación de alto nivel, pero no ejecutable.
Código Ejecutable	3	3. Escrito en Lenguaje Máquina y directamente ejecutable.

El código fuente escrito en algún lenguaje de programación de alto nivel, el objeto escrito en lenguaje máquina sin ser ejecutable y el código ejecutable, escrito también en lenguaje máquina y ya sí ejecutable por el ordenador, son las distintas fases por donde pasan nuestros programas.

5.5.- Máquinas virtuales.

Una máquina virtual es un tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.

Esta capa de software desempeña un papel muy importante en el funcionamiento de los lenguajes de programación, tanto compilado como interpretado.

Con el uso de máquinas virtuales podremos desarrollar y ejecutar una aplicación sobre cualquier equipo, independientemente de las características concretas de los componentes físicos instalados. Esto garantiza la portabilidad (*Capacidad de un programa para ser ejecutado en cualquier arquitectura física de un equipo.*) de las aplicaciones.

Las funciones principales de una máquina virtual son las siguientes:

- Conseguir que las aplicaciones sean portables.
- Reservar memoria para los objetos que se crean y liberar la memoria no utilizada.
- Comunicarse con el sistema donde se instala la aplicación (huésped), para el control de los dispositivos hardware implicados en los procesos.
- Cumplimiento de las normas de seguridad de las aplicaciones.

CARACTERÍSTICAS DE LA MÁQUINA VIRTUAL

Cuando el código fuente se compila se obtiene código objeto (bytecode, código intermedio).

Para ejecutarlo en cualquier máquina se requiere tener independencia respecto al hardware concreto que se vaya a utilizar.

Para ello, la máquina virtual aísla la aplicación de los detalles físicos del equipo en cuestión.

Funciona como una capa de software de bajo nivel y actúa como puente entre el bytecode de la aplicación y los dispositivos físicos del sistema.

La Máquina Virtual verifica todo el bytecode antes de ejecutarlo.

La Máquina Virtual protege direcciones de memoria.

La máquina virtual actúa de puente entre la aplicación y el hardware concreto del equipo donde se instale.

Para saber más

En el siguiente enlace te presentamos el proceso de instalación de la JVM (Máquina Virtual de Java) y su apariencia.

http://www.unabvirtual.edu.co/ayuda/manuales_pdf/maquinavirtualjava.pdf

5.5.1.- Frameworks.

Un framework (*Plataforma, entorno, marco de trabajo del desarrollo rápido de aplicaciones*) es una estructura de ayuda al programador, en base a la cual podemos desarrollar proyectos sin partir desde cero.

Se trata de una plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.

Con el uso de framework podemos pasar más tiempo analizando los requerimientos del sistema y las especificaciones técnicas de nuestra aplicación, ya que la tarea laboriosa de los detalles de programación queda resuelta.

- Ventajas de utilizar un framework:
  - **Desarrollo rápido** de software.
  - **Reutilización** de partes de código para otras aplicaciones.
  - **Diseño** uniforme del software.
  - **Portabilidad** de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.
- Inconvenientes:
  - Gran dependencia del código respecto al framework utilizado (sin cambiamos de framework, habrá que reescribir gran parte de la aplicación).
  - La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.

Para saber más

El uso creciente de frameworks hace que tengamos que estar reciclándonos constantemente. En el siguiente enlace, hay un documento muy interesante de sus principales características, ventajas y formas de uso:

http://www.maestrosdelweb.com/editorial/los-frameworks-de-php-agilizan-tu-trabajo/

Ejemplos de Frameworks:

- **.NET** es un framework para desarrollar aplicaciones sobre Windows. Ofrece el "Visual Studio .net" que nos da facilidades para construir aplicaciones y su motor es el ".Net framework" que permite ejecutar dichas aplicaciones. Es un componente que se instala sobre el sistema operativo.
- Spring de Java. Son conjuntos de bibliotecas (API's) para el desarrollo y ejecución de aplicaciones.

Debes conocer

El proceso de instalación y configuración del framework Spring de Java, así como varios ejemplos de uso. En el siguiente enlace encontrarás una guía muy útil detallando los pasos a seguir:

http://pablolg.wikispaces.com/file/view/spring_tutorial_v0.271.pdf

5.5.2.- Entornos de ejecución.

Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas. En ocasiones pertenece al propio sistema operativo, pero también se puede instalar como software independiente que funcionará por debajo de la aplicación.

Es decir, es un conjunto de utilidades que permiten la ejecución de programas.

Se denomina runtime al tiempo que tarda un programa en ejecutarse en la computadora.

Durante la ejecución, los entornos se encargarán de:

- Configurar la memoria principal disponible en el sistema.
- Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados. Considerando que las bibliotecas son el conjunto de subprogramas que sirven para desarrollar o comunicar componentes software pero que ya existen previamente y los subprogramas serán aquellos que hemos creado a propósito para el programa.
- Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).

Funcionamiento del entorno de ejecución:

El Entorno de Ejecución está formado por la máquina virtual y los API's (bibliotecas de clases estándar, necesarias para que la aplicación, escrita en algún Lenguaje de Programación pueda ser ejecutada). Estos dos componentes se suelen distribuir conjuntamente, porque necesitan ser compatibles entre sí.

El entorno funciona como intermediario entre el lenguaje fuente y el sistema operativo, y consigue ejecutar aplicaciones.

Sin embargo, si lo que queremos es desarrollar nuevas aplicaciones, no es suficiente con el entorno de ejecución.

Adelantándonos a lo que veremos en la próxima unidad, para desarrollar aplicaciones necesitamos algo más. Ese "algo más" se llama entorno de desarrollo.

Autoevaluación
Señala la afirmación falsa respecto de los entornos de ejecución:

- ◉ Su principal utilidad es la de permitir el desarrollo rápido de aplicaciones.
- ○ Actúa como mediador entre el sistema operativo y el código fuente.
- ○ Es el conjunto de la máquina virtual y bibliotecas necesarias para la ejecución.

5.5.3.- Java runtimeenvironment.

En esta sección de va a explicar el funcionamiento, instalación, configuración y primeros pasos del RuntimeEnvironment del lenguaje Java (se hace extensible a los demás lenguajes de programación).

Concepto.

Se denomina JRE al Java RuntimeEnvironment (entorno en tiempo de ejecución Java).

El JRE se compone de un conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.

Componentes.

JRE está formado por:

- Una Máquina virtual Java (JMV o JVM si consideramos las siglas en inglés), que es el programa que interpreta el código de la aplicación escrito en Java.
- Bibliotecas de clase estándar que implementan el API de Java.
- Las dos: JMV y API de Java son consistentes entre sí, por ello son distribuidas conjuntamente.

Lo primero es descargarnos el programa JRE. (Java2 RuntimeEnvironment JRE 1.6.0.21). Java es software libre, por lo que podemos descargarnos la aplicación libremente.

Una vez descargado, comienza el proceso de instalación, siguiendo los pasos del asistente.

Debes conocer
El proceso de descarga, instalación y configuración del entorno de ejecución de programas. En el siguiente enlace, se explican los pasos para hacerlo bajo el sistema operativo Linux.
http://www.java.com/es/download/help/linux_install.xml

Para saber más
En el siguiente enlace encontrarás un tutorial del lenguaje Java, con sus principales características y órdenes y comandos principales.
http://www.tecnun.es/asignaturas/Informat1/AyudaInf/aprendainf/Java/Java2.pdf

5.6.- Pruebas.

Caso práctico
María reúne todos los códigos diseñados y los prepara para implementarlos en el equipo del cliente. Juan se percata de ello, y le recuerda a su amiga que aún no los han sometido a pruebas. Juan se acuerda bien de la vez que le pasó aquello: hace dos años, cuando fue a presentar una aplicación a sus clientes, no paraba de dar errores de todo tipo… los clientes, por supuesto, no la aceptaron y Juan perdió un mes de duro trabajo y estuvo a punto de perder su empleo…
"—No tan deprisa María, tenemos que PROBAR la aplicación".

Una vez obtenido el software, la siguiente fase del ciclo de vida es la realización de pruebas.

Normalmente, éstas se realizan sobre un conjunto de datos de prueba, que consisten en un conjunto seleccionado y predefinido de datos límite a los que la aplicación es sometida.

La realización de pruebas es imprescindible para asegurar la validación y verificación del software construido.

Entre todas las pruebas que se efectúan sobre el software podemos distinguir básicamente:

PRUEBAS UNITARIAS

Consisten en probar, una a una, las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente). JUnit es el entorno de pruebas para Java.

PRUEBAS DE INTEGRACIÓN

Se realizan una vez que se han realizado con éxito las pruebas unitarias y consistirán en comprobar el funcionamiento del sistema completo: con todas sus partes interrelacionadas.

La prueba final se denomina comúnmente Beta Test, ésta se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente (a ser posible, en los equipos del cliente y bajo un funcionamiento normal de su empresa).

El período de prueba será normalmente el pactado con el cliente.

Autoevaluación
Si las pruebas unitarias se realizan con éxito, ¿es obligatorio realizar las de integración?

- [ ] Sí, si la aplicación está formada por más de cinco módulos diferentes.
- [x] Sí, en cualquier caso.

Para saber más
Puedes visitar la siguiente página web, donde se detallan los tipos de pruebas que suelen hacer al software y la función de cada una.
http://www.sistedes.es/TJISBD/Vol-1/No-4/articles/pris-07-raja-ctps.pdf

5.7.- Documentación.

Caso práctico
Ada ha quedado dentro de dos días con su cliente. Pregunta a María por todos los dossiers de documentación. La pálida expresión de la joven hace que Ada arda en desesperación: "—¿No habéis documentado las etapas? ¿Cómo voy a explicarle al cliente y sus empleados el funcionamiento del software? ¿Cómo vamos a realizar su mantenimiento?".

Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.

¿Por qué hay que documentar todas las fases del proyecto?

Para dar toda la información a los usuarios de nuestro software y poder acometer futuras revisiones del proyecto.

Tenemos que ir documentando el proyecto en todas las fases del mismo, para pasar de una a otra de forma clara y definida. Una correcta documentación permitirá la reutilización de parte de los programas en otras aplicaciones, siempre y cuando se desarrollen con diseño modular.

Distinguimos tres grandes documentos en el desarrollo de software:

Documentos a elaborar en el proceso de desarrollo de software			
	GUÍA TÉCNICA	**GUÍA DE USO**	**GUÍA DE INSTALACIÓN**

Quedan reflejados:	• El diseño de la aplicación. • La codifica-ción de los programas. • Las pruebas realizadas.	• Descripción de la funciona-lidad de la aplicación. • Forma de comenzar a ejecu-tar la aplicación. • Ejemplos de uso del progra-ma. • Requerimientos software de la aplicación. • Solución de los posibles problemas que se pueden presentar.	Toda la información necesaria para: • Puesta en marcha. • Explotación. • Seguridad del sistema.
¿A quién va dirigido?	Al personal técni-co en informática (analistas y pro-gramadores).	A los usuarios que van a usar la aplicación (clientes).	Al personal informático res-ponsable de la instalación, en colaboración con los usuarios que van a usar la aplicación (clientes).
¿Cuál es su objetivo?	Facilitar un co-rrecto desarrollo, realizar correc-ciones en los pro-gramas y permitir un mantenimiento futuro.	Dar a los usuarios finales toda la información necesaria para utilizar la aplicación.	Dar toda la información nece-saria para garantizar que la implantación de la aplicación se realice de forma segura, confiable y precisa.

5.8.- Explotación.

Caso práctico
Llega el día de la cita con la cadena hotelera. Ada y Juan se dirigen al hotel sonde se va a instalar y configurar la aplicación. Si todo va bien, se irá implementando en los demás hoteles de la cadena. Ada no quiere que se le pase ni un detalle: lleva consigo la guía de uso y la guía de instalación.

Después de todas las fases anteriores, una vez que las pruebas nos demuestran que el software es fiable, carece de errores y hemos documentado todas las fases, el siguiente paso es la explotación.

Aunque diversos autores consideran la explotación y el mantenimiento como la misma etapa, nosotros vamos a diferenciarlas en base al momento en que se realizan.

La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.

La explotación es la instalación, puesta a punto y funcionamiento de la aplicación en el equipo final del cliente.

En el proceso de instalación, los programas son transferidos al computador del usuario cliente y posteriormente configurados y verificados.

Es recomendable que los futuros clientes estén presentes en este momento e irles comentando cómo se va planteando la instalación.

En este momento, se suelen llevan a cabo las Beta Test, que son las últimas pruebas que se realizan en los propios equipos del cliente y bajo cargas normales de trabajo.

Una vez instalada, pasamos a la fase de configuración.

En ella, asignamos los parámetros de funcionamiento normal de la empresa y probamos que la aplicación es operativa. También puede ocurrir que la configuración la realicen los propios usuarios finales, siempre y cuando les hayamos dado previamente la guía de instalación. Y también, si la aplicación es más sencilla, podemos programar la configuración de manera que se realice automáticamente tras instalarla. (Si el software es "a medida", lo más aconsejable es que la hagan aquellos que la han fabricado).

Una vez se ha configurado, el siguiente y último paso es la fase de producción normal. La aplicación pasa a manos de los usuarios finales y se da comienzo a la explotación del software.

Es muy importante tenerlo todo preparado antes de presentarle el producto al cliente: será el momento crítico del proyecto.

Reflexiona

Realizas un proyecto software por vez primera y no te das cuenta de documentarlo. Consigues venderlo a buen precio a una empresa. Al cabo de un par de meses te piden que actualices algunas de las funciones, para tener mayor funcionalidad. Estás contento o contenta porque eso significa un ingreso extra. Te paras un momento...¿Dónde están los códigos? ¿Qué hacía exactamente la aplicación? ¿Cómo se diseñó? No lo recuerdas... Probablemente hayas perdido un ingreso extra y unos buenos clientes.

5.9.- Mantenimiento.

Caso práctico

Ada reúne por última vez durante estas semanas a su equipo. Todos celebran que el proyecto se ha implementado con éxito y que sus clientes han quedado satisfechos.
—Esto aún no ha terminado —comenta Ada—, nos quedan muchas cosas por hacer. Esta tarde me reúno con los clientes. ¿Cómo vamos a gestionar el mantenimiento de la aplicación?

Sería lógico pensar que con la entrega de nuestra aplicación (la instalación y configuración de nuestro proyecto en los equipos del cliente) hemos terminado nuestro trabajo.

En cualquier otro sector laboral esto es así, pero el caso de la construcción de software es muy diferente.

La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.

Por su naturaleza, el software es cambiante y deberá actualizarse y evolucionar con el tiempo. Deberá ir adaptándose de forma paralela a las mejoras del hardware en el mercado y afrontar situaciones nuevas que no existían cuando el software se construyó.

Además, siempre surgen errores que habrá que ir corrigiendo y nuevas versiones del producto mejores que las anteriores.

Por todo ello, se pacta con el cliente un servicio de mantenimiento de la aplicación (que también tendrá un coste temporal y económico).

El mantenimiento se define como el proceso de control, mejora y optimización del software.

Su duración es la mayor en todo el ciclo de vida del software, ya que también comprende las actualizaciones y evoluciones futuras del mismo.

Los tipos de cambios que hacen necesario el mantenimiento del software son los siguientes:

- **Perfectivos**: Para mejorar la funcionalidad del software.
- **Evolutivos**: El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
- **Adaptativos**: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
- **Correctivos**: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).

Autoevaluación

¿Cuál es, en tu opinión, la etapa más importante del desarrollo de software?

- (•) El análisis de requisitos.
- ( ) La codificación.
- ( ) Las pruebas y documentación.
- ( ) La explotación y el mantenimiento.