UF2:

Diseño y realización de pruebas
Optimización de documentos

Olimpia Olguín Tinoco

Innovación en Formación Profesional

Grupo Planeta

DISEÑO Y REALIZACIÓN DE PRUEBAS

Generación de batería de pruebas sobre los siguientes dos aspectos fundamentales:

Casuísticas habituales: las habituales que gestionará el programa.

Casuísticas críticas: en las que un fallo en el producto podría generar un grave problema.

PLANIFICACIÓN DE PRUEBAS

Calendario. ¿Cuándo y durante cuánto tiempo?

Normas. Objetivos de la compañía, ámbito de las pruebas, metodología a utilizar, etc.

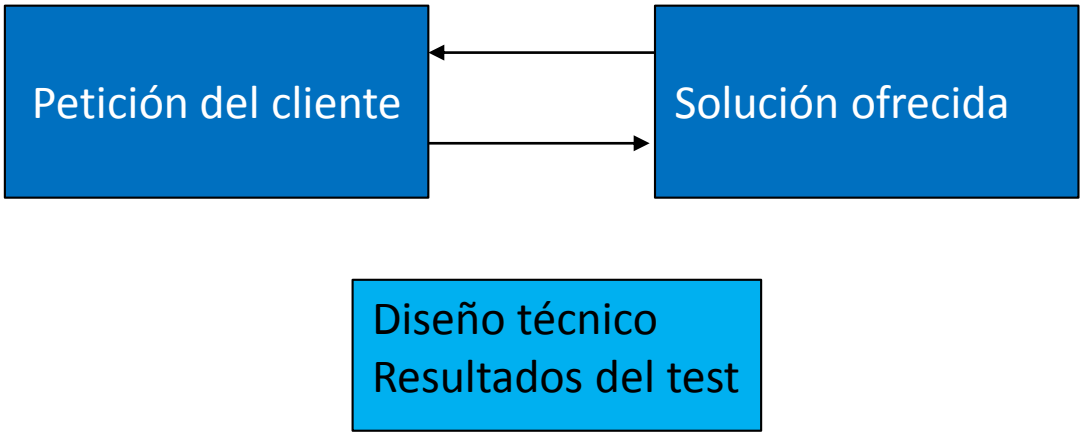
Recursos. Hardware y software necesarios para los test, herramientas que van a utilizarse y perfiles de los técnicos y usuarios que colaboraran.

Productos. Incluye todos los elementos que conforman la futura instalación en entorno real.

Contingencia. Alternativas de solución en caso de que falle la prueba principal del producto

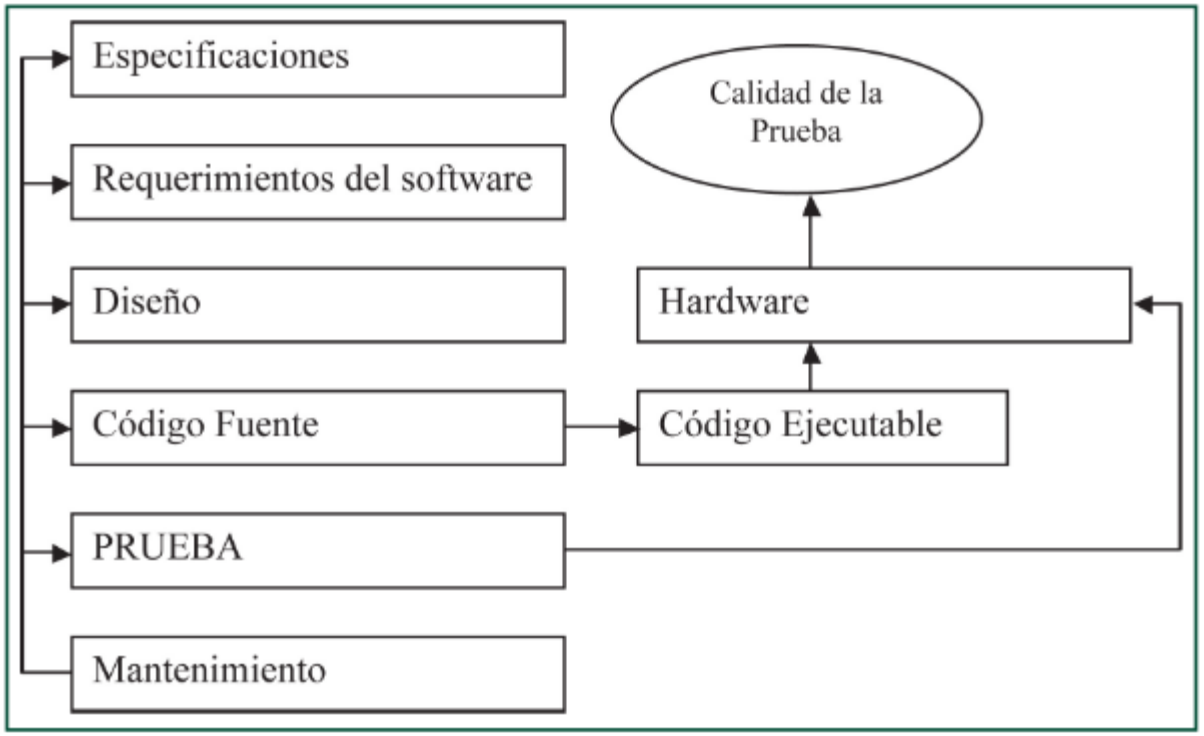
a) TIPOS DE PRUEBA

Prueba funcional: Este tipo de pruebas son externas al código del programa, es decir, solo miden resultados.



Prueba estructural: A partir del **estudio del código**, de **su estructura interna** y la comparación con el **diseño técnico** de puede llegar a la conclusión de las acciones **correctoras** si así fuera necesario.

Este tipo de prueba utiliza **casos de prueba relacionados con la lógica del código fuente, estructuras de datos, tablas, tiempos de ejecución y control correcto de bucles de sentencias**. Este tipo de pruebas entran en el propio código del programa.



Prueba de regresión: Se aplica en el mantenimiento de un software ya existente al que se le requiere realizar una modificación porque:

- Contenía un error
- Mejora del producto

Sí es un nuevo producto, este tipo de prueba se realiza después de haber realizado el test principal, implica la realización de cambios.

PROCEDIMIENTOS Y CASOS DE PRUEBA

Procedimientos: Son especificaciones formales de los casos de prueba, además de los requerimientos humanos y materiales.

Casos de prueba: indican las circunstancias en que deben ejecutarse.

Definen una ejecución específica del programa desarrollado examinando aspectos específicos de entrada y salida de datos.

Caso de Prueba:				
Descripción:				
Requerimiento de Datos:				
Número Paso	**Descripción**	**Resultado esperado**	**Nombre Transacción**	**Tiempo esperado**
01	Invocar aplicación desde escritorio	Se muestra la pantalla de menú principal	Usuario_login	5
02	Seleccionar 'Búsqueda' en el menú	Se muestra la pantalla de 'Búsqueda'	Seleccionar_búsqueda	3

EJEMPLOS:

Producte:	xcxxx
Responsable:	xxxxxx
Release inicial:	may-17
Release actual:	14/05/2017
Peticions Release actual	tickets
Codificació de	

Codi prova	Nom de la prova	Descripció prova	Resultat esperat	Tipus d'invocació	Reingección	Prova bloquejant	Observacions prova
CR_01							

INTEGRACIÓN

Resultados	Evidéncia de la prueba	Tiempos de respuesta	Tipos de execución: Manual/Automática	Probador	Fecha de ejecución	Defecto	Observaciones	

PREPRODUCCIÓN

Resultados	Reinjección	Evidéncia de prueba	Tiempo de respuesta	Tipos de execución: Manual/Automática	Probador	Fecha de ejecución	Defecto	Observacion
OK				Manual		23/09/2019		

HERRAMIENTAS DE DEPURACIÓN

Son **programas cuyo objetivo principal es el de probar el funcionamiento de otros programas informáticos**. Las serie de acciones que se utilizan para ello son las siguientes:

Puntos de ruptura(Breakpoint): El programador puede poner en el código fuente unas señales que la herramienta de depuración interpretará como un punto de parada de ejecución. Cuando el control del programa llegue a ese punto, se detendrá temporalmente, pero mantendrá los valores de variables, funciones y objetos en la memoria. Una vez analizados los datos es posible continuar la ejecución a partir de dicho punto.

Tipos de ejecución:

- **Ejecución línea a línea:** Cuando se desea una ejecución muy detallada.
- **Ejecución por grupos:** El programa se detiene hasta que termina una declaración completa**.**

PRUEBAS DE CÓDIGO

Es probar todo un programa, pero **comprobar todas las posibilidades de ejecución que puedan darse es técnicamente imposible**, para ello se necesita determinar:

Hasta que nivel de pruebas queremos llegar.

- Naturaleza del negocio
- En donde se sitúa el software
- Presupuesto económico

Prueba de cubrimiento: Para contabilizar de forma automática las líneas y las veces que se ejecutan. Se utiliza para comprobar el tanto por ciento de código utilizado. Se ejecuta al menos una vez cada sentencia.

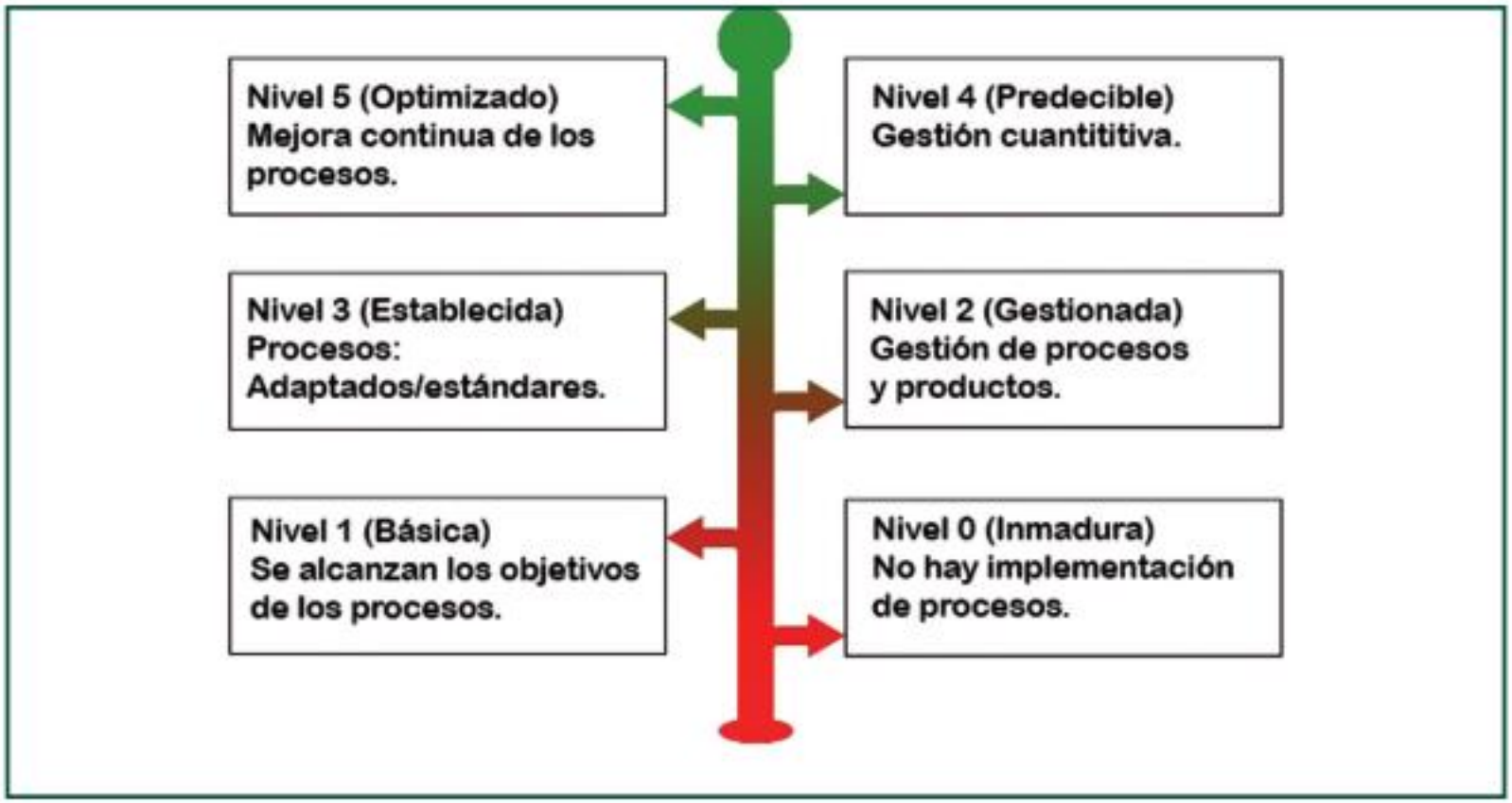
Clase de equivalencia: Se define como un modelo en el que unos datos son **equivalentes a otros**. De esta manera una gran cantidad de datos se reduce a unos pocos grupos de datos. Es importante probar los valores frontera o límite que están en el borde de cambio de una clase de equivalencia a otra.

NORMAS DE CALIDAD

Son documentos en los que se especifican los requisitos, las reglas y las formas de trabajo que una empresa debe seguir para poder ser considerada "certificada" en dicha norma.

- **ISO/IEC 15504 SPICE - Software Process Improvement Capability Determination.**

Facilitan **la mejora continua de los procesos de las organizaciones, especialmente aquellos relacionados con le desarrollo y mantenimiento del software**. Se basa en la **evaluación**, a través de niveles **de madurez de la empresa**.
Los niveles de madurez de una empresa:


Nivel 5 (Optimizado)
Mejora continua de los procesos.
Nivel 4 (Predecible)
Gestión cuantititiva.
Nivel 3 (Establecida)
Procesos:
Adaptados/estándares.
Nivel 2 (Gestionada)
Gestión de procesos
y productos.
Nivel 1 (Básica)
Se alcanzan los objetivos
de los procesos.
Nivel 0 (Inmadura)
No hay implementación
de procesos.

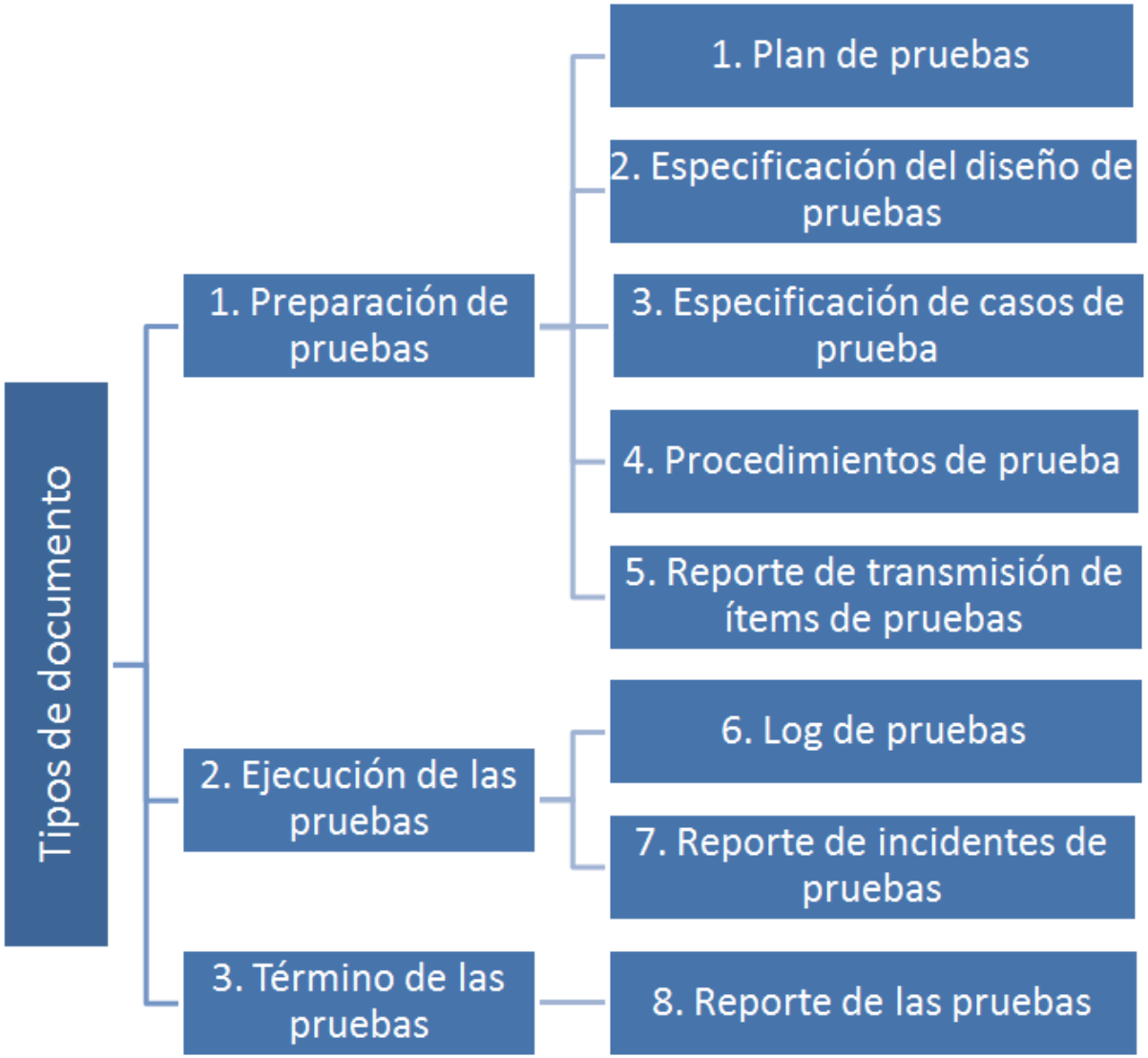
- **ISO 9001.** Es la solución más eficaz para las organizaciones que pretendan el **desarrollo e implantacion de Sistemas de gestión de calidad**.

Además de abarcar:

  - El desarrollo del software,
  - Servicios al cliente
  - Sistemas de mantenimiento y flujos administrativos

- **ISO IEC 90003 2004**. Esta normativa sirve de guía para una **constante mejora**. Se aborda desde diferentes ámbitos:
  - Requerimientos del sistema empresarial.
  - Requerimientos de gestión.
  - Requerimientos de recursos.
  - Requerimientos de control y realización.
  - Requerimientos de contingencia y reparación.
- **Normativa IEEE 829-2008.** También **conocida como norma 829 para la documentación de test de sistemas y software**.
  - Proporciona un conjunto estandarizado de documentos para las pruebas de software.
  - Las tareas del proceso de prueba se especifican para diferentes niveles de integridad

- Determinan la amplitud y profundidad apropiadas de la documentación de prueba.
- El alcances de las pruebas abarca sistemas basados en software, hardware y sus interfaces.
- Esta norma se aplica a los sistemas basados en software que se desarrollan, mantienen o reutilizan:
  - Elementos heredados
  - Comerciales
  - No evolutivos
- Existen 8 tipos de documentos que se pueden usarse en 3 etapas distintas de las pruebas de software.


Tipos de documento
1. Preparación de pruebas
1. Plan de pruebas
2. Especificación del diseño de pruebas
3. Especificación de casos de prueba
4. Procedimientos de prueba
5. Reporte de transmisión de ítems de pruebas
2. Ejecución de las pruebas
6. Log de pruebas
7. Reporte de incidentes de pruebas
3. Término de las pruebas
8. Reporte de las pruebas

DOCUMENTACIÓN DE PRUEBA

Consta de tres grupos de documentos, cada uno de ellos con sus propios formularios estándar, que deberán rellenarse con la información adecuada:

Preparación de las pruebas:

- Declaración del Plan.
- Especificaciones del diseño.
- Especificaciones de los casos de test.
- Procedimiento.
- Detalle de los elementos a probar.

Ejecución de las pruebas:

- Registro detallado de cada test. → evidencias
- Incidentes acaecidos durante las pruebas.

Finalización de las pruebas:

- Informe resumen.

Por último, es preciso utilizar una gestión documental aceptable

PRUEBAS UNITARIAS:

Es aquella que se realiza en un sólo componente del programa de forma aislada, que puede ser un módulo o una clase. Algunas veces el aislamiento del componente para su prueba suele ser complicado, para ellos se utilizan las siguientes herramientas: Junit, PHPUnit (PHP), NUNit (.NET)

AUTOMATIZACIÓN DE PRUEBAS

Los asistentes informáticos de pruebas de software son muy adecuados para los entornos de desarrollo. La automatización puede orientarse hacia dos áreas concretas:

- Prueba de código:
- Interfaz gráfica de usuario: comprobación manual de la interfaz.

Ejemplo de algunas herramientas:

HP Unified Functional Testing (UFT). Este software permite la automatización de pruebas funcionales y de regresión.

Microsoft Test Manager. Esta herramienta permite planificar, crear y ejecutar casos de test, además de su automatización.

Selenium. Automatización de pruebas para aplicaciones web. Permite pruebas de regresión.

CONCEPTO DE REFACTORIZACIÓN

Refactorizar (Refactoring) es una técnica que consiste en modificar la estructura interna de un programa para mejorar su comprensión.

Los cambios realizados en el código no afectan al funcionamiento del programa, es decir, el programa debe hacer exactamente lo mismo que hacía antes de los cambios.

Limitaciones suelen se en plazos, presupuestos y recursos finitos, por ejemplo:

- Los proyectos suelen estar **condicionados por el aspecto económico**.
- La refactorización **no puede facturarse** a un cliente inmediatamente.
- El programador **que ”limpia” el código puede no ser el mismo que el creador**, conlo que se corre el riesgo de que entren nuevos errores.
- Las modificaciones o **cambios han de poder probarse sin esfuerzo**, por lo que hay que tener preparadas unas pruebas funcionales o unitarias.
- Si se trata de un **código antiguo en el que han trabajado muchos programadores**, algunos de ellos noveles, podemos encontrarnos con un **auténtico lío** cuya mejora puede costar cara.
- A veces **no podemos cambiar la interfaz de usuario** y **complica la refactorización**.

a) PATRONES DE FACTORIZACIÓN MÁS USADOS

Los patrones pueden clasificarse en tres grandes grupos:

1. **Patrones de creación**. Ayudan a la creación y configuración de objetos.
   - **Simple Factory (Fábrica simple**). Clase para la **creación de instancias** de objetos.
   - **Factory Method (Método de Fábrica).** Define **una interfaz para crear objetos** y permite a las subclases decidir la clase a instanciar.
   - **Abstract Factory (Fábrica Abstracta)**. Interfaz para crear familias de objetos relacionados o dependientes sin especificar sus clases.
   - **Builder (Constructor).** La construcción de un objeto se separa de su representación, por lo que pueden crearse diferentes representaciones con el mismo proceso de construcción.

2. **Patrones de estructura**. Asisten en la generación de grupos de objetos en estructuras mayores. Se adecuan a la forma de combinar clases y objetos en el montaje de dichas estructuras:
   - **Adapter (Adaptador).** Hace de intermediario entre dos clases que posean interfaces incompatibles y, por lo tanto, de difícil comunicación

- **Bridge (Puente).** Descompone un componente en dos jerarquías de clase: una abstracción funcional (algo que hace) y una implementación (la cosa en sí).
- **Decorator (Decorador).** Agrega o limita competencias a un objeto.

Patrones de comportamiento. Conforman el flujo y tipo de comunicación entre los objetos de un programa.

- **Chain of Responsibility (Cadena de responsabilidad**). La cadena de mensajes puede ser respondida por más de un receptor.
- **Command (Orden).** Representa una petición con un objeto.
- **Mediator (Mediador).** Se introduce un nuevo objeto que administra los mensajes entre objetos.

b) ANALIZADORES DE CODIGO

Un análisis de código consiste en verificar el código fuente con la intención de realizar posteriormente trabajos de mantenimiento (refactorización).

- **Análisis dinámico**. Se ejecuta el programa para detectar defectos de ejecución:
  - Valgrind Tool Suite (C / C++).
  - IBM Rational AppScan. Chequea las vulnerabilidades de seguridad.
- **Análisis estático**. Se examina el código sin ejecutar el programa:
  - FxCop .NET Framework 2.0. Ofrece información acerca del diseño, localización, rendimiento y mejoras de la **seguridad.**
  - PMD Rule Designer. Chequea código fuente Java encontrando errores, variables no usadas y código duplicado

c) HERRAMIENTAS DE AYUDA A LA REFACTORIZACIÓN

En las dos plataformas más utilizadas, tenemos los siguientes:

Java con los dos IDE's más utilizados NetBeans y Eclipse(contiene un set de refactorings automáticos)

- **RefactorIt** contiene un reestructurador de diseño. Puede usarse sólo o como plugin de entornos de desarrollo, como Eclipse o NetBeans entre otros.

.NET La plataforma de Microsoft multilenguaje también posee potentes herramientas de optimización de código como:

- **ReSharper**. Inspección de código y refactoring.
- **Visual Estudio** para los lenguajes soportados.

CONTROL DE VERSIONES

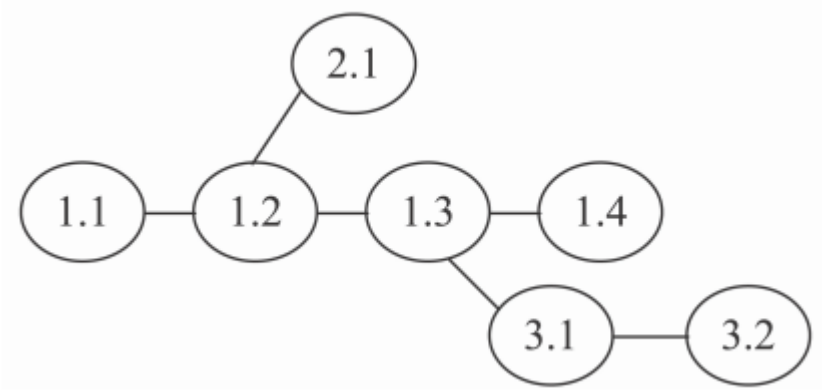
Para una correcta gestión del código fuente, de los documentos, de los gráficos y de los ficheros que conforman un proyecto informático se necesita de un software de control de versiones que provea de una base de datos que incluya la trazabilidad de las revisiones hechas por todos los programadores. Las más usuales es:

La unión de números (y, en ocasiones, también letras), que indican el nivel que definen, por ejemplo: (1.1, 1.2…)

a) ESTRUCTURA DE LAS HERRAMIENTAS DE CONTROL DE VERSIONES

El sistema gestiona una base de datos en forma de árbol en el que se van apilando los diferentes cambios que se realizan en el software. Los elementos que los componen son:

- **Tronco.** Es el camino de cambio principal (1.1, 1.2, 1.3).
- **Cabeza.** Es la última versión del tronco (1.4).
- **Ramas.** Son los caminos secundarios (2.1, 3.1).
- **Delta.** Son los cambios de una versión respecto a otra. Es la modificación en un archivo bajo control de versiones.



b) REPOSITORIO

Es el lugar en donde se almacena una copia completa de todos los ficheros y directorios que están bajo la gestión del software de control de versiones.

Los proyectos diferentes suelen estar ubicados en distintos repositorios. En cualquier caso, éstos no deberán atenderse entre ellos.

Un mismo repositorio controla la concurrencia de varios programadores que intentan acceder al mismo elemento y a la misma versión de código mediante avisos y posibilidad de lectura sin grabación de código.

c) HERRAMIENTAS DE CONTROL DE VERSIONES

Existen muchas herramientas que gestionan el control de versiones. Alguna de ellas son:

- **CVS (Concurrent Versions System).** Control exhaustivo de la trazabilidad de nuevas funcionalidades, así como de la introducción involuntaria de errores en el código.
- **Mercurial.** Es un gestor de código fuente que gestiona el desarrollo colaborativo entre varios programadores y mantiene una traza de los cambios que se realizan.
- **Subversio (Apache Subversion).** Es es una herramienta de control de versiones open source basada en un repositorio cuyo funcionamiento se asemeja enormemente al de un sistema de ficheros .
- **Git.** Esta pensado en la eficiencia y la confiablidad del mantenimiento de versiones de aplicaciones cuando éstas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora y coordinar el trabajo que varias personas realizan sobre archivos compartidos.
- **Perforce, Clearcase,** etc, etc

DOCUMENTACION

La documentación acompañan al desarrollo de software y aportan información del flujo de datos del desarrollo. Desde un punto de vista de la gestión del negocio y de la gestión tecnológica.

- **Documentación de gestión.** Documentos destinados a informar a los responsables del proyecto (presupuestos, desviaciones, roles a adoptar por el personal, recursos y plazos de entrega).
- **Documentación de requerimientos.** Información relativa a las necesidades planteadas de cambio. Pueden mostrarse descripciones del actual sistema y los motivos del cambio. Requerimientos Inicio del proyecto y cambios
- **Diseño de función y tecnológico.** A partir del sistema que hay que cambiar se describen las nuevas funcionalidades, es decir, ”lo que se debe hacer”. Incluye los diseños tecnológicos, es decir, ”cómo lo vamos a hacer”.
- **Documentación de usuario**. Manuales, diagramas explicativos, normativa de trabajo, etc.

a) USO DE COMENTARIOS

Los comentarios son textos que se introducen dentro del código fuente para ofrecer información adicional cuando sea necesario.

- No son sentencias ejecutables.
- Utilizar una **metodología rigurosa** acerca **de cómo documentar** los programas para que cualquier desarrollador utilice una forma uniforme y sistemática
- Insertar código de comentario en **cabecera de fuente**, y especificar la información relativa a la versión, el programador, fechas, la compilación y todo aquello que resulte necesario para el desarrollo del proyecto.

b) HERRAMIENTAS

Existen herramientas externas que generan la documentación, un ejemplo de estas son las siguientes:

- **Javadoc.** Genera documentación en formato HTML de programas en lenguaje Java.
- **Doxygen**. Es un sistema de documentación para C++, C, Java, Python, PHP y otros

Innovación
en Formación
Profesional

La mejor forma de aprender es "haciendo".
Educación y empresas forman un binomio inseparable