

Prueba y documentación de  
programas. Técnicas.

## **TEMA 30 (32 SAI)**

---

**ABACUS NT**

Oposiciones 2021

*Índice*

---

**1. Introducción.**

**1. Ciclo de Vida del Software**

**1. Prueba de programas**

1.1. Prueba de programas

**2. Técnicas para la realización de pruebas**

2.1. Técnicas estáticas

2.2. Técnicas de diseño de pruebas dinámicas

1.1.1. Pruebas de caja negra

1.1.2. Pruebas de caja blanca

2.3. El ciclo de pruebas

2.4. Niveles de prueba

1.1.3. Pruebas unitarias

1.1.4. Pruebas de integración

1.1.5. Pruebas de sistema

1.1.6. Pruebas de aceptación de usuario

**3. Documentación de programas**

1.2. Documentación del proyecto.

1.3. Documentación de especificaciones [IEEE 830].

1.4. Documentación de desarrollo.

1.5. Documentación de pruebas (IEEE 829).

**4. Técnicas para la realización de la documentación**

1.6. Características de la documentación

1.7. Técnicas

**5. Conclusión**

**5.1. Relación del tema con el sistema educativo actual**

**2. Bibliografía**

## 1. Introducción.

Cuando se desarrolla software, la prueba o testeo del mismo es una parte fundamental del ciclo de vida. La prueba es indispensable, puesto que a partir de ella se puede determinar la calidad de los productos implementados; a pesar de ello su importancia se suele desestimar.

La **Ley de Parkinson**, enunciada por el británico Cyril Northcote Parkinson en 1957, afirma que "el trabajo se expande hasta llenar el tiempo disponible para que se termine"; es decir: el tiempo asignado a un proyecto viene a ser exactamente el que efectivamente tarda en desarrollarse: si el tiempo es escaso el desarrollo será peor y se omitirán partes fundamentales como la prueba.

Por otro lado, tan importante como la prueba del software es la **documentación** que genera, ya que durante la etapa de mantenimiento es imprescindible para poder modificar el programa desarrollado.

Es necesario distinguir también entre los conceptos de verificación y validación dentro de la prueba del software:

- **Verificación:** el programa es correcto, acorde a sus especificaciones.
- **Validación:** El programa satisface las necesidades del usuario.

Por consiguiente, la prueba del software debe afrontarse siempre desde una perspectiva de búsqueda de la calidad, dentro de los límites impuestos por la planificación del proyecto.

## 1. Ciclo de Vida del Software

La creación de software consta generalmente de una serie de pasos claramente diferenciados:

1. **Análisis:** se recopila documentación, se estudian los **requisitos** del usuario final (qué se va a implementar, cuáles son las necesidades del usuario). Se suele empezar con una descripción en lenguaje natural de lo que quiere el usuario. Al final de esta etapa tendremos una descripción clara y precisa de qué producto vamos a construir, qué funcionalidades aportará y qué comportamiento tendrá.
2. **Diseño:** una vez que sabemos qué debemos hacer, debemos determinar cómo lo haremos. Estudiaremos el problema y lo descompondremos en problemas más pequeños; definiremos la estructura de la solución, identificando los módulos que hay que implementar y sus relaciones; definiremos los algoritmos a emplear y el lenguaje de programación a utilizar, etc.
3. **Implementación y codificación:** se escribe el programa fuente en el lenguaje de programación establecido y se genera el código ejecutable.
4. **Pruebas:** se comprueba que el programa no tenga errores, y que se cumplen los criterios de calidad.
5. **Mantenimiento:** el programador se asegura de que el programa siga funcionando, y lo va adaptando también a nuevos requisitos que puedan ir surgiendo



Muchas veces tratamos de acortar el proceso de programación yendo directamente desde el análisis del problema a la codificación del mismo.

Este atajo puede resultar muy tentador e incluso parecer una buena forma de ahorrar tiempo. Sin embargo, este método necesita realmente más tiempo y esfuerzo.

El desarrollo de una solución general antes de escribir realmente el programa puede ayudar al programador a manejar el problema, tomar el camino correcto y evitar errores innecesarios.

Además, puesto que la mayoría de los programas se usarán una y otra vez, la **documentación** y **mantenimiento** de programas son partes importantes de la programación.

## 1. Prueba de programas

### 1.1. Prueba de programas

Recordando la afirmación de Dijkstra en la que plantea que *el hecho de realizar una prueba no garantiza la ausencia de defectos, sino que solamente demuestra la presencia de estos*, se debe tener en cuenta que no se pueden probar todas las posibilidades de su funcionamiento ni siquiera en programas sencillos. Por tanto, debe realizarse la planificación y el diseño de los casos de prueba, que tiene como objetivo conseguir una confianza aceptable sin necesidad de consumir una gran cantidad de recursos.

Un caso de prueba es un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular. Cada caso de prueba debe definir el resultado de salida esperado, que se comparará con el realmente obtenido.

Recomendaciones en el diseño de los casos de prueba:

- El programador debe evitar probar sus propios programas, ya que es normal que las situaciones que olvidó al crear el programa se olviden también al crear los casos de prueba.
- Al generar casos de prueba, **se deben incluir tanto datos de entrada válidos y esperados como no válidos e inesperados**.
- Las pruebas deben centrarse en probar si el sistema **no hace** lo que debe hacer, **y si hace lo que no debe hacer**, es decir, si provoca efectos secundarios adversos.
- Se deben evitar los casos **no documentados** ni diseñados con cuidado, ya que es necesario probar muchas veces el software, y se debe tener claro lo que funciona y lo que no funciona.
- No deben hacerse planes de prueba suponiendo que prácticamente no hay defectos en los programas y dedicando pocos recursos a las pruebas, puesto que **siempre hay defectos**.
- **Donde hay un defecto hay otros**, es decir, la probabilidad de descubrir nuevos defectos en una parte del software es proporcional al número de defectos ya descubierto.
- El principio de **Pareto** aplicado a las pruebas del software implica que el 80% de todos los errores que se encuentren van a ser debidas a un 20% de todos los módulos del programa. Es necesario, por tanto, encontrar dichos módulos y probarlos exhaustivamente.

## 2. Técnicas para la realización de pruebas

### 2.1. Técnicas estáticas

Al contrario que las pruebas dinámicas, que exigen la ejecución de software, las técnicas de pruebas estáticas se basan en el examen manual (revisiones) y en el análisis automatizado (análisis estático) del código sin ejecutar el código. A este respecto, se pueden utilizar herramientas de software libre como **SonarQube**, que es un analizador estático de código fuente.

Las **métricas del software y la ingeniería inversa** también pueden ser descritas como forma de análisis estático de software.

### 2.2. Técnicas de diseño de pruebas dinámicas

Las pruebas dinámicas se clasifican, a su vez, en técnicas de pruebas de caja negra, **en las que se desconoce el código fuente del componente** y de caja blanca, en donde sí se conoce:

#### 1.1.1. Pruebas de caja negra

Este tipo de pruebas se realizan **interactuando con la interfaz del software, ya que no se tiene acceso al código fuente**. Los casos de prueba generados se diseñan a partir de valores de entrada/salida, determinando la validez de una salida en función de un conjunto determinado de entradas.

El diseño de los casos de prueba debe basarse en la elección de diferentes flujos de ejecución del programa con el fin de detectar el máximo número de errores. Para ello se utiliza el criterio de

cobertura lógica, que utiliza la representación de un grafo de flujo para establecer los caminos posibles desde la sentencia inicial hasta la final.

- **Partición de equivalencia:** En esta técnica se divide el dominio de los valores de entrada de un componente en un número finito de clases de equivalencia, de tal forma que, un único valor de una clase representa al resto de los valores de dicha clase. Así se reduce el número de casos que hay que probar. Por ejemplo, se puede segmentar el día en horas clave, los vehículos por tipos, etc.
- **Análisis de valores límites:** Los casos de prueba que exploran las **condiciones límites** de un programa producen buenos resultados, ya que existe una tendencia a que el software falle precisamente cuando las variables se aproximan o llegan a sus valores máximos y mínimos, por lo que se toman intervalos de valores alrededor de esos valores extremos.
- **Prueba de comparación.** Esta técnica es muy empleada en sistemas en tiempo real y/o embebidos. Consiste en crear sistemas redundantes pero distintos, desarrollados incluso por distintos equipos de trabajo. De esta forma comparamos la salida de ambos sistemas, para evitar completamente los errores.
- **Pruebas de transición de estado:** En este caso se considera el sistema como una máquina de estados, y se diseñan pruebas para cubrir secuencias típicas de estados, o cubrir todos los estados, transiciones o probar transiciones invalidas.
- **Pruebas por experiencia:** La generación de casos de prueba se realiza a partir de la intuición y la experiencia del programador. La idea básica es redactar una lista de posibles situaciones de error y basar los casos de prueba en dicha lista.
- **Tablas de decisión:** Se crea una tabla de condiciones/acciones y se utilizan conectores lógicos (and, or, not) entre diversas entradas (condiciones) para analizar la validez de la salida (acción).
- **Pruebas de caso de uso:** Consisten en ejecutar escenarios que describen el uso que los usuarios van a darle al software. Son de gran utilidad para diseñar las pruebas de aceptación con la participación del cliente.

### 1.1.2. Pruebas de caja blanca

Consiste en centrarse en la **estructura interna del programa para elegir los casos de prueba, analizando los caminos de ejecución**. El diseño de los casos de prueba debe basarse en la elección de diferentes flujos de ejecución del programa con el fin de detectar el máximo número de errores. Para ello se utiliza el criterio de cobertura lógica, que utiliza la representación de un grafo de flujo para establecer los caminos posibles desde la sentencia inicial hasta la final.

El diagrama de flujo consiste en un grafo dirigido en el que los nodos están formados por bloques de sentencias simples que incluyen alguna condición, y los arcos representan el flujo de ejecución del programa. Si existe una combinación de condiciones, se separan en el propio grafo.

Los criterios de cobertura lógica se clasifican en orden ascendente de exigencia y coste así:

- **Cobertura de sentencias.** La cobertura de sentencia viene determinada por el número de sentencias ejecutables cubiertas por los casos de prueba dividido entre el número de todas las sentencias ejecutables en el código objeto de la prueba. Con esta técnica se pretende diseñar casos de prueba que aumenten dicha cobertura.
- **Cobertura de decisiones.** Consiste en diseñar casos de prueba para que al menos cada decisión lógica tenga un resultado verdadero y otro falso.
- **Cobertura de condiciones.** Consiste en generar los casos de prueba para que cada condición de cada decisión tome al menos una vez un valor verdadero y otro falso.
- **Criterio de condición múltiple.** Consiste en diseñar casos de prueba para que se cumplan de forma independiente las condiciones de una condición múltiple.
- **Criterio de cobertura de caminos.** Se recorren absolutamente todos los caminos; esto normalmente es impracticable.

### 2.3. El ciclo de pruebas

De acuerdo con la IEEE [IEEE90] el concepto de prueba es “una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones específicas, se observan o almacenan los resultados y se realiza una evaluación de algún aspecto del sistema o componente”.

**La prueba es por tanto el proceso de ejecución de un programa con el intento deliberado de encontrar errores.**

El proceso de pruebas básico consta de las siguientes actividades principales:

- **Planificación y control:** Se establecen los **objetivos**, y el **plan de pruebas**.
- **Análisis y diseño:** los objetivos se transforman en **casos de prueba tangibles**
- **Implementación y ejecución:** Se configuran y ejecutan las pruebas.
- **Evaluación de los criterios de salida:** En este punto se **evalúa los resultados** de las pruebas respecto a los **objetivos** definidos.
- **Actividades de cierre de pruebas:** Aquí se realizan varias **tareas finales**, tales como analizar lecciones aprendidas, documentar la aceptación del sistema, etc.

La estrategia de ejecución de pruebas integra el diseño de los casos de prueba en una serie de casos coordinados a través de la creación de distintos niveles de prueba con distintos objetivos cada uno. Existe una correspondencia entre cada nivel de prueba y cada etapa del desarrollo del ciclo de vida del software (Modelo en V)

- **Especificación y lógica de módulos → Pruebas unitarias.** Centran sus actividades en ejercitando la lógica del módulo (caja blanca) y los distintos aspectos de la especificación de las funciones que debe realizar el módulo (caja negra).
- **Diseño modular → Pruebas de integración.** Deben tener en cuenta los mecanismos de agrupación de módulos, fijados en la estructura del programa, y en general los interfaces entre los componentes de la arquitectura del software.

- **Requisitos de usuario → Pruebas de validación.** Sirven para que el usuario pueda verificar si el producto se ajusta a los requisitos establecidos por él mismo.
- **Especificación de requisitos → Pruebas de sistema.** Comprueban los desajustes entre el software y los requisitos de funcionamiento, y el grado de cumplimiento de los objetivos.

La **depuración** es el resultado de una prueba efectiva. Su objetivo es encontrar y corregir la causa del error. Como estrategia se puede utilizar:

- **Fuerza bruta.** El error se encuentra mediante la traza de la ejecución y sentencias WRITE.
- **Vuelta atrás.** Se recorre manualmente el código hacia atrás hasta encontrar el error.
- **Eliminación de causas.** Se establecen los datos relacionados con el error y se agrupan para determinar una hipótesis de causa. Se establecen pruebas para desechar grupos de datos y se van probando hasta aislar los datos que producen el error.

## 2.4. Niveles de prueba

Según el **ISTQB** (International Software Testing Qualifications Board), el proceso de prueba se puede ejecutar en los siguientes niveles:

### 1.1.3. Pruebas unitarias

Tienen como objeto comprobar el funcionamiento de un **único componente**.

Se trata de las pruebas formales que permiten asegurar que cada módulo funciona individualmente de forma adecuada, y son de tipo **caja blanca**. Puede abarcar desde un módulo hasta un grupo de módulos, incluso un programa completo. Estas pruebas suelen realizarlas el **propio personal de desarrollo**, pero evitando que sea el propio programador del módulo.

Los aspectos a evaluar son los siguientes:

- **Interfaz.** Comprobar la información que entra y sale del módulo.
- **Estructuras de datos locales.** Comprobar la integridad de los datos locales.
- **Caminos independientes.** Asegurar que todos los caminos y cálculos funcionan bien.
- **Caminos de manejo de errores.** Disponer de unos caminos de manejo de errores que informen y gestionen el procesamiento cuando se detecta un error.
- **Condiciones límite.** Comprobar que el módulo funciona correctamente en los límites establecidos como restricciones de procesamiento.

### 1.1.4. Pruebas de integración

Su objetivo principal es comprobar la interacción de diferentes partes de un sistema, validando la comunicación entre sus interfaces.

Su objetivo es verificar el diseño y la construcción del programa, y son del tipo caja negra más algunas de caja blanca. Implican una progresión ordenada de pruebas que van desde los componentes o módulos culminando en el sistema completo. Normalmente se solapan con las pruebas unitarias y las realiza el desarrollador.

Hay dos tipos fundamentales de integración:

**Integración no incremental** Se prueba cada módulo por separado y luego se integran todos de una vez y se prueba el programa completo.

**Integración incremental** Se combina el siguiente módulo que se debe probar con el conjunto de módulos que ya han sido probados. Hay tres maneras de realizar la integración de la jerarquía arborescente de módulos:

- **Ascendente (bottom-up).** Comenzando por los módulos hoja, se utilizan módulos denominados conductores que simulan la llamada para introducir los datos de prueba y recoger los resultados. Una vez probados, los módulos conductores se van sustituyendo sucesivamente por los módulos del nivel superior de jerarquía. El problema es que el programa como entidad no existe hasta el final.
- **Descendente (top-down).** Comienza por el módulo de control principal y va incorporando módulos subordinados, descendiendo progresivamente en profundidad o en amplitud. Se utilizan módulos denominados resguardo que simulan los módulos dependientes que aún faltan por probar, los cuales implican más esfuerzo. Si hay secciones críticas especialmente complejas deben integrarse lo antes posible. El orden de integración debe incorporar cuanto antes los módulos de entrada/salida para facilitar la ejecución de pruebas.
- **Combinada o sándwich.** Es una combinación de la estrategia descendente para los módulos de jerarquía superior, y de la estrategia ascendente para los módulos de jerarquía inferior.

### 1.1.5. Pruebas de sistema

Estas pruebas buscan probar si los componentes de un sistema son compatibles, e interactúan correctamente. Es el proceso de prueba de un sistema integrado de hardware y software.

Son de tipo **caja negra** y permiten comprobar el cumplimiento de todos los requisitos funcionales, considerando el producto software final al completo en un entorno de sistema.

Permiten verificar además la seguridad del sistema y su capacidad de recuperación de fallos, y el funcionamiento y rendimiento en las interfaces hardware, software, de usuario y de operador en condiciones límite y de sobrecarga.

### 1.1.6. Pruebas de aceptación de usuario

El objetivo de estas pruebas es validar la disposición de un sistema para su despliegue y uso. Son pruebas en las que participa el usuario, y están enfocadas hacia la comprobación de los requisitos especificados. En el caso del software a medida, son pruebas de tipo caja negra planificadas y organizadas formalmente para determinar si se cumplen los requisitos de aceptación marcados por el cliente.

En el caso de productos comerciales podemos destacar las pruebas alfa y beta.

- **Pruebas alfa:**

Se efectúan bajo la vigilancia continua del equipo de desarrollo, en un entorno controlado, registrando continuamente los errores y problemas de uso.

- **Pruebas beta:**

Se llevan a cabo en varios lugares por usuarios finales en un entorno normal sin apoyo ninguno. Es el propio cliente o usuario final el que notifica los problemas que surjan.

### 3. Documentación de programas

Cada una de las fases del ciclo de vida del software tiene asociado una documentación descriptiva, la cual es utilizada como entrada para la siguiente fase. La documentación es de suma importancia para el mantenimiento de los programas, y para su evolución.

#### 1.2. Documentación del proyecto.

**Oferta de desarrollo.** Recoge una descripción del acuerdo alcanzado por la empresa proveedora y la empresa cliente. Deben especificarse valoraciones económicas, valoraciones temporales, disposiciones técnicas y una referencia al documento de especificaciones de requisitos.

**Documentación de las especificaciones.** Recoge qué es lo que hay que hacer, con qué se debe hacer, cuándo debe estar finalizado y cómo hacerlo.

**Documentación del desarrollo.** Recoge toda la información necesaria que se va generando durante el desarrollo de la aplicación. No se entrega al cliente.

**Documentación de las pruebas.** Recoge todos los casos de prueba que se han generado para toda la aplicación. Se entrega al cliente cuando finaliza el desarrollo de la aplicación y deberá contener los estados del control de cambios (solucionadas, en observación, rechazadas y motivo) y de las mejoras impuestas por el cliente fuera del documento de especificaciones.

**Documentación del cliente.** Deberá entregarse:

- **Manual de explotación.** Recoge la información necesaria para poner en explotación la aplicación. Va dirigido a la instalación y configuración fundamentalmente, por lo que su estructura depende de la organización donde se vaya a implantar la aplicación.
- **Manual de administración.** Es el documento que recoge las tareas que deben realizarse para el mantenimiento y administración de aplicación.
- **Manual de usuario.** Reúne la información, normas y documentación necesaria para que el usuario conozca y utilice adecuadamente la aplicación desarrollada. El usuario debe comprender la información, ya que debe utilizarlo como guía de consulta y referencia. Los objetivos que se persiguen son el correcto uso de la aplicación y que permita detectar y corregir errores. Su redacción debe ser concisa y clara, con los apoyos gráficos necesarios.

- Auditoria. Documento final que se genera para comprobar la calidad de la aplicación además de ver si cumple con la metodología de desarrollo y especificaciones de requisitos.

### 1.3. Documentación de especificaciones [IEEE 830].

Este documento tiene como objeto asegurar que tanto el desarrollador como el cliente tienen la misma idea sobre las funcionalidades del sistema. Es muy importante que esto quede claro ya que de lo contrario el desarrollo software no será aceptable.

Debe contener los siguientes apartados:

- Fines y objetivos del software.
- Descripción detallada del flujo y contenido de la información y de la interfaz del sistema.
- Descripción funcional incluidas las restricciones de diseño y los requisitos de rendimiento.
- Descripción del comportamiento del software ante sucesos externos y controles internos.
- Criterios de validación, con clases de pruebas y respuesta esperada del software.

### 1.4. Documentación de desarrollo.

Por norma general se debe documentar el código fuente, cómo se producirán las entradas y salidas, y la declaración y el uso de los datos.

**Especificaciones de operaciones de entrada/salida.** Hay que establecer cuáles son los dispositivos de E/S que se van a utilizar, y el formato de los datos de entrada y de salida. Si el usuario tiene un alto grado de interacción con la aplicación, el interfaz ofrecido al usuario será simple e intuitivo. En cualquier operación de E/S es importante que se pueda dar marcha atrás y si se produce un error, saber cuál es el módulo que lo produjo y su posible solución.

**Documentación del código fuente.** El nombre del programa o módulo y los nombres de las variables y estructuras de datos deben guardar relación con el contenido que tendrán para mejorar la legibilidad, manteniendo un diccionario de datos. También es importante introducir comentarios que expresen las funciones desarrolladas en el código. Deben incluirse líneas en blanco, tabuladores y sangrías que faciliten la lectura. Todo programa debe llevar la fecha de creación y fecha de las modificaciones, así como los cambios que se produzcan y su autor.

### 1.5. Documentación de pruebas (IEEE 829).

Es necesaria para una buena organización de las mismas, así como para asegurar su reutilización. También contribuye a la eficacia y eficiencia de la aplicación generada.

Los siguientes documentos están asociados a la fase de diseño de las pruebas:

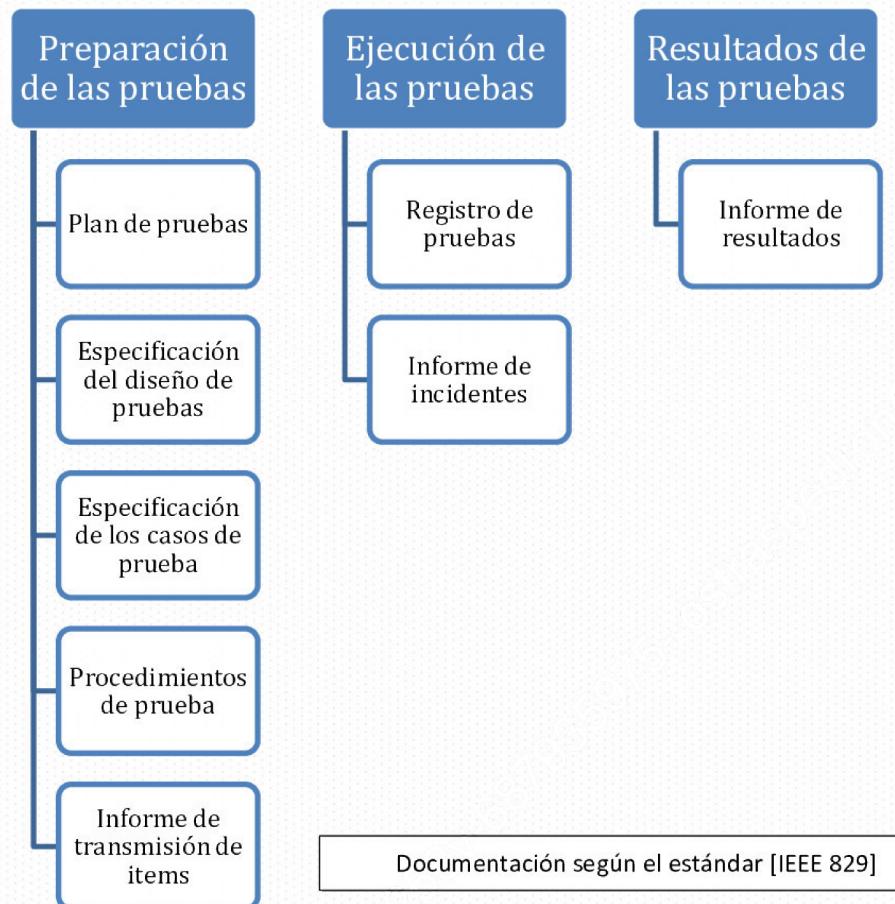
- **Plan de pruebas.** El objetivo del documento es señalar el enfoque, los recursos y el esquema de actividades de prueba, así como los elementos a probar, las características, los criterios, las actividades de prueba, el personal responsable y los riesgos asociados.
- **Especificación del diseño de pruebas.** El objetivo del documento es especificar los refinamientos necesarios sobre el enfoque general reflejado en el plan de pruebas e identificar

las características de los elementos software que se deben probar y los criterios de paso/fallo de la prueba.

- **Especificación de caso de prueba.** El objetivo del documento es definir uno de los casos de prueba identificado por una especificación del diseño de las pruebas, detallando los elementos software a probar, las entradas y salidas del caso de prueba y los requisitos especiales del procedimiento de prueba.
- **Especificación de procedimiento de prueba.** El objetivo del documento es especificar las acciones necesarias para la ejecución de un conjunto de casos de prueba o, más generalmente, los pasos utilizados para analizar un elemento software con el propósito de evaluar un conjunto de características del mismo.
- **Informe de transmisión de ítems.** Describe los ítems para prueba, dónde encontrarlos y da la aprobación para su liberación. Garantiza al tester de que los ítems están listos para ser robados

Los siguientes documentos están asociados a la fase de ejecución de las pruebas:

- **Histórico de pruebas.** El objetivo del documento es documentar todos los hechos relevantes ocurridos durante la ejecución de las pruebas.
- **Informe de incidente.** El objetivo del documento es documentar cada incidente (por ejemplo, una interrupción en las pruebas debido a un corte de electricidad, bloqueo del teclado, etc.) ocurrido en la prueba y que requiera una posterior investigación.
- **Informe resumen de pruebas.** El objetivo del documento es resumir los resultados de las actividades de prueba (las señaladas en el propio informe) y aportar una evaluación del software basada en dichos resultados.



## 4. Técnicas para la realización de la documentación

### 1.6. Características de la documentación

Las características de la documentación del software son:

- Mejora la usabilidad del programa, ya que documenta lo que se hace o se puede hacer en cada momento.
- Facilita el mantenimiento del software, ya que pasado un tiempo es imposible entender características que no estén documentadas.
- Es un factor importante para medir la calidad del software.
- Indica el avance del proyecto, al estar presente en todas las fases del ciclo de vida del software.

### 1.7. Técnicas

Algunas de las técnicas que se pueden seguir para elaborar la documentación de un programa son:

- **Planteamiento top-down:** Primero se define y documentan los aspectos más generales de un sistema, como por ejemplo objetivos, arquitectura, etc., y se termina documentando aquellos aspectos más detallados y específicos.
- **Planteamiento bottom-up:** En este caso se empieza documentando desde lo más específico hasta terminar en lo más general. Favorece la aplicación de pruebas tempranas.
- **Generar documentación de forma automática:** También se puede elaborar la documentación de un programa de forma automática a partir de la codificación, así, por ejemplo, utilizando el plugin eUML2 en eclipse se pueden generar diagramas UML.

## 5. Conclusión

La prueba es una actividad fundamental en muchos procesos de desarrollo, incluyendo el del software. De manera general, se puede decir que la prueba de software permite al desarrollador determinar si el producto generado satisface las especificaciones establecidas. Así mismo, una prueba de software permite detectar la presencia de errores que pudieran generar salidas o comportamientos inapropiados durante su ejecución.

A modo de síntesis, se podría destacar que la creación de sistemas de información, no sólo conlleva el desarrollo de código, sino que también requiere, entre otras actividades, la realización de pruebas, y la elaboración de documentación, siendo, estas dos últimas actividades, esenciales para garantizar la calidad y el mantenimiento de los mismos.

### 5.1. Relación del tema con el sistema educativo actual

Este tema puede ser desarrollado en los siguientes módulos formativos (para atribución docente de PES)

- Bachillerato – Tecnologías de la Información y la Comunicación II (PES)
- GS- GS – DAW – Desarrollo Web en Entorno Servidor DAW/DAM – Programación (PES)

Aunque los profesores de SAI no tienen una atribución docente en ningún módulo con aplicación directa del tema, siempre es probable que se diseñe algún pequeño programa en clase, para lo que recurrir a la notación de pseudocódigo o de ordinograma sería un recurso esclarecedor.

## 2. Bibliografía

- Introduction to Java Programming and Data Structures, Comprehensive Version. Y. Daniel Liang. Pearson, 11<sup>a</sup> edición. 2017.
- Java 9. Francisco Javier Moldes Teo. Anaya. 2017.

- Introducción a la computación. J, Glenn Brookshear, Pearson, 11º edición. 2012
- Fundamentos de programación. Algoritmos, estructuras de datos y objetos. Luis Joyanes Aguilar, McGraw-Hill, 4º edición. 2008.
- Langsam, Augenstein y Tanembaum: “Estructuras de Datos con C y C++”, Prentice-Hall 1997
- Prieto A., Lloris A. y Torres J.C.: Introducción a la Informática, 4ª ed (2006) McGraw-Hill
- Lenguajes de programación, principios y práctica. 2ª Ed (2004). Kenneth C.Louden

