PRUEBAS Y CALIDAD DEL SOFTWARE

9.1.- Introducción

La prueba de programas es un elemento importante para la garantía de calidad del software y representa una revisión final de las especificaciones, del diseño y de la codificación. No puede garantizar que no existan defectos en el programa, pero puede demostrar que existen defectos en el software.

Las pruebas constituyen un método mas de poder verificar y validar el software junto a las revisiones de los productos que preceden al código en el ciclo de vida. Puede definirse la verificación como "*el proceso de evaluación de un sistema o de uno de sus componentes para determinar si los productos de una fase dada satisfacen las condiciones impuestas al comienzo de dicha fase*".

Por su parte, la validación es "el proceso de evaluación de un sistema o de sus componentes durante o al final del proceso de desarrollo para determinar que satisface los requisitos especificados".

Puede definirse pues informalmente estos dos conceptos como:

- Verificación: **¿Se está construyendo correctamente el producto?**
- Validación: **¿Se está construyendo el producto correcto?**

Las pruebas pues permiten verificar y validar el software cuando ya está en forma de código ejecutable. Algunos conceptos relacionados con las pruebas son:

Pruebas.- Actividad por la cual un sistema o uno de sus componentes se ejecuta en circunstancias previamente especificadas, los resultados se observan y se registran y se realiza una evaluación de algún aspecto.

- ***Caso de prueba.-*** *Un conjunto de entradas, condiciones de ejecución y resultados esperados desarrollados para un objetivo particular, por ejemplo, ejercitar un camino concreto de un programa o verificar el cumplimiento de un determinado requisito.*

Defecto.- Una definición de datos o un paso de procesamiento incorrectos en un programa

Fallo.- La incapacidad de un sistema o de alguno de sus componentes para realizar las funciones requeridas dentro de los requisitos de rendimiento especificados.

Error.- Se consideran dos acepciones distintas:

- La diferencia entre el valor calculado, observado o medido y el valor verdadero, especificado o teóricamente correcto.
- Una acción humana que conduce a un resultado incorrecto

9.2.- Recomendaciones para la realización de pruebas

1°.- Cada caso de prueba debe definir el resultado de salida esperado, que debe compararse con el realmente obtenido de la ejecución de la prueba. La discrepancia entre ambos se considera síntoma de un posible defecto de software.

2°.- El programador debe evitar probar sus propios programas

3°.- Se debe inspeccionar a conciencia el resultado de cada prueba para así poder descubrir posibles síntomas de defectos.

4°.- Al generar casos de prueba deben incluirse tanto datos de entrada válidos y esperados como no válidos e inesperados

5°.- Las pruebas deben centrarse en dos objetivos:

- Probar si el software no hace lo que debe hacer
- Probar si el software hace lo que no debe hacer, esto es, si provoca efectos secundarios adversos.

6°.- Se deben evitar los casos desechables, esto es, los no documentados ni diseñados con cuidado

7°.- No deben hacerse planes de prueba suponiendo que prácticamente no hay defectos en los programas, y por tanto, dedicando pocos recursos a las pruebas.

8°.- La probabilifdad de descubrir nuevos defectos es proporcional al número de defectos ya descubiertos

9°.- Las pruebas sun una tarea tanto o mas creativa que el desarrollo del software.

9.3.- Técnicas de diseño de casos de pruebas.

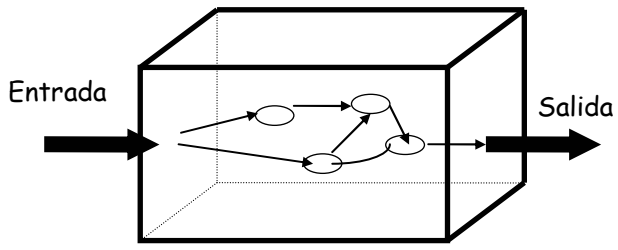
Las técnicas de diseño de pruebas tienen como objetivo conseguir una confianza aceptable en que se detectarán los defectos existentes (la seguridad total solo puede obtenerse de la prueba exhaustiva la cual es impracticable). Existen tres enfoques principales para el diseño de los casos:

- ***Enfoque Estructural o de caja blanca.***- Consiste en centrarse en la estructura interna (implementación) del programa para elegir los casos de prueba.
- ***Enfoque Funcional o de caja negra.*** – Consiste en estudiar la especificación de las funciones, la entrada y la salida para derivar los casos.
- ***Enfoque Aleatorio.***- Consiste en utilizar modelos (generalmente estadísticos que representen las posibles entradas al programa para crear a partir de ellos los casos de prueba.

Estos enfoque no son excluyentes entre sí ya que se pueden combinar para conseguir una detección de defectos mas eficaz.

9.4.- Pruebas estructurales

Conociendo el funcionamiento del producto. La prueba demostrará entonces si la operación interna se ajusta a las especificaciones y todos los componentes internos actúan de la forma adecuada.



El diseño de los casos tiene que basarse en la elección de caminos importantes que ofrezcan una seguridad aceptable en descubrir defectos, para ello se utilizan los **criterios de cobertura lógica.** En definitiva, este tipo de prueba demostrará si:

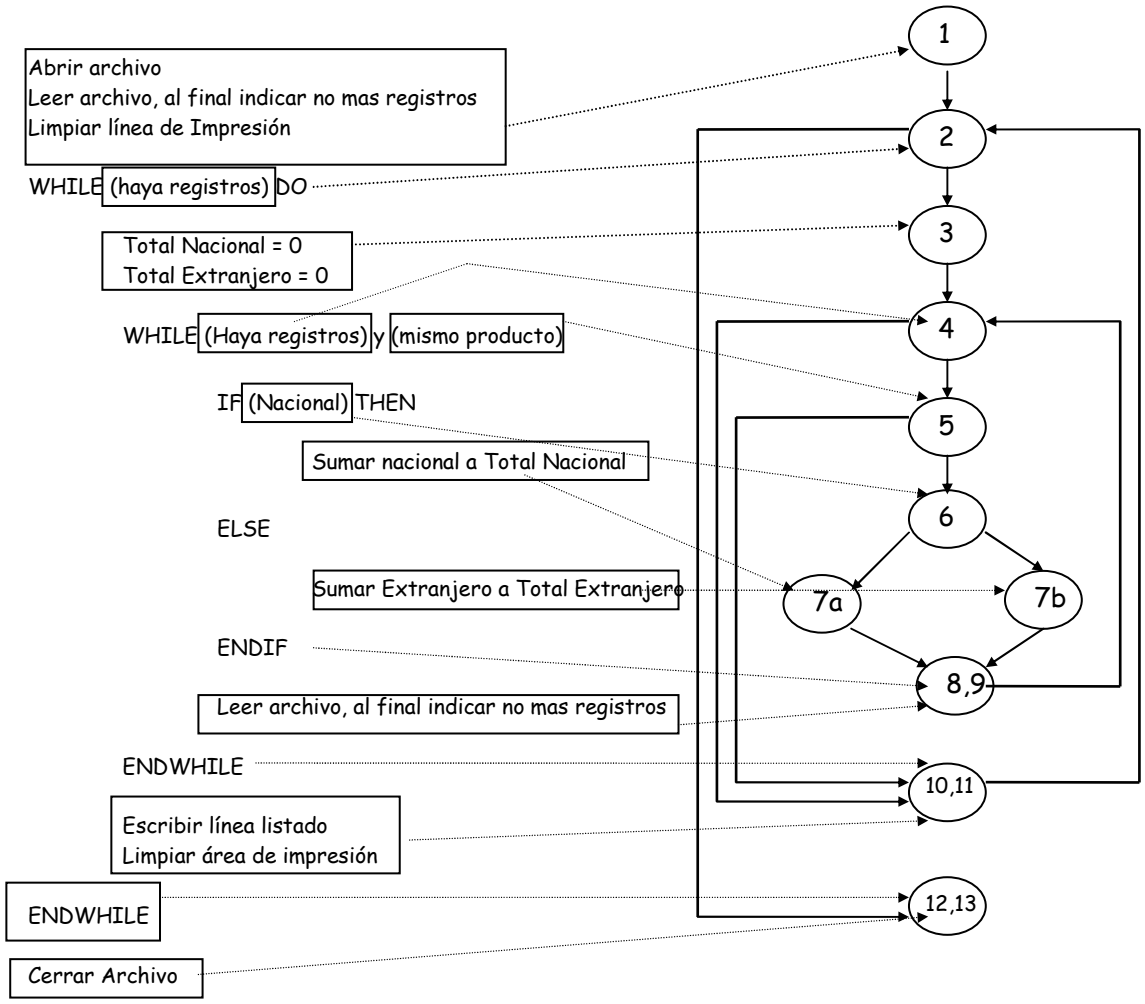
- Se ejercitan por lo menos una vez todos los caminos independientes de cada módulo.
- Se ejercitan todas las decisiones lógicas en sus posibilidades verdadera y falsa.
- Se ejecutan todos los bucles en sus límites y con sus límites operacionales.
- Se ejercitan las estructuras internas de datos que aseguran su validez.

En general, debe hacerse este tipo de pruebas en un producto debido a que:

- Los errores lógicos y las suposiciones incorrectas son inversamente proporcionales a la probabilidad de que se ejecute un camino del programa.
- Un camino lógico que parece tener pocas posibilidades de ejecutarse, de hecho, se puede ejecutar de forma regular
- Los errores tipográficos son aleatorios.

Habitualmente, aunque no existe ningún criterio específico de representación gráfica del software, suele tomarse como base los llamados **diagramas de flujo de control** como el de la figura.

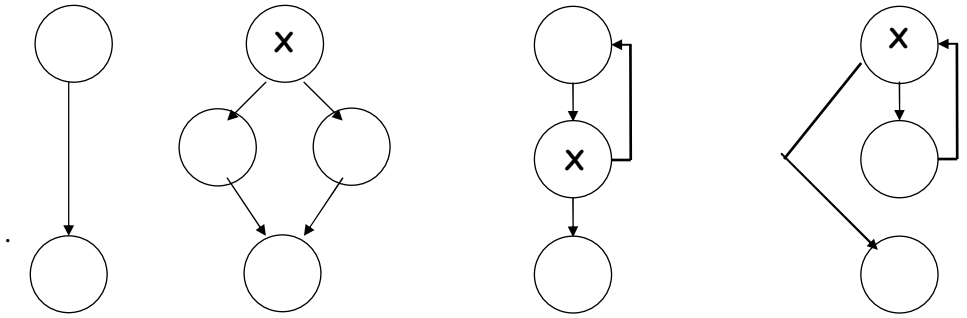
Así mismo, en el cuadro posterior se establecen algunas recomendaciones para dibujar grafos de flujo de programas para generar los casos de prueba correspondientes. (Actualmente existen herramientas que dibujan el grafo de flujo de un programa facilitando el código fuente del mismo como entrada, en concreto la herramienta **Logiscope**)



Para dibujar el grafo de un programa es recomendable seguir los siguientes pasos:

1°- Señalar sobre el código cada **condición** de cada **decisión** tanto en las sentencias IF-THEN y CASE-OF como en los bucles WHILE o UNTIL.

2°.- Agrupar el resto de las sentencias situadas entre cada dos condiciones según los esquemas de representación de estructuras básicas:



Secuencia **Condición IF** **Bucle HACER HASTA** **Bucle MIENTRAS**

3°.- Numerar tanto las condiciones como los grupos de sentencias, de forma que se les asigne un identificador único. Es recomendable alterar el orden en el que aparecen las condiciones en una decisión multicondicional, situándolas en orden decreciente de restricción, es decir, primero las mas restrictivas.

Una posible clasificación de criterios de cobertura lógica es la siguiente:

♦ ***Cobertura de sentencias.*** **-** Se trata de generar los casos de prueba necesarios para que cada sentencia o instrucción del programa se ejecute al menos una vez

♦ ***Cobertura de decisiones***.- Consiste en escribir los casos suficientes para que cada decisión tenga, por lo menos una vez, un resultado verdadero y al menos una vez, un resultado falso.

♦ ***Cobertura de condiciones.*** **-** Se trata de diseñar tantos caos como sea necesario para que cada condición de cada decisión tenga, por lo menos una vez, un resultado verdadero y al menos una vez, un resultado falso

♦ ***Criterio de decisión/condición.*** **-** Consiste en exigir el criterio de cobertura de condiciones obligando a que se cumpla también el criterio de decisiones.

♦ ***Criterio de condición múltiple.*** **-** Consiste en considerar que cada decisión multicondicional se descompone en varias decisiones unicondicionales, es decir, una decisión

IF (a=1) AND (c=4)

se convierte en una concatenación de dos decisiones:

IF (a=1) y **IF(c=4)**

exigiendo que todas las combinaciones posibles de resultados (verdadero/falso) de cada condición en cada decisión se ejecuten al menos una vez

En general, se define **camino** como la secuencia de sentencias encadenadas desde la sentencia inicial del programa hasta la sentencia final

Se define **camino de prueba** como un camino del programa que atraviesa, como máximo, una vez el interior de cada bucle que encuentra

Los bucles constituyen el elemento de los programas que genera un mayor número de problemas para la cobertura de caminos. Su tratamiento no es sencillo ni siquiera adoptando el concepto del camino de prueba. Por ello suelen utilizarse métricas para identificar los caminos independientes a utilizar, en particular la métrica de McCabe.

COMPLEJIDAD CICLOMÁTICA DE McCABE

Indica el número de caminos independientes que existen en un grafo, lo que implica que un buen criterio de prueba es la consecución del conjunto de caminos independientes igual a la métrica..

La complejidad de McCabe (**V(G)**) se puede calcular de tres maneras distintas:

1.- **V(G) = a - n + 2** siendo **a** el número de arcos o aristas del grafo y **n** el número de nodos

2.- **V(G) = r** siendo **r** el número de regiones cerradas del grafo

3.- **V(G) = c + 1** siendo **c** el número de nodos de condición.

Calculado el valor de V(G) puede afirmarse que éste es el número máximo de caminos independientes del grafo, que serán los que deban utilizarse como casos de prueba.

Para ayudar a la elección de dichos caminos, McCabe ideó el **Método del camino básico** que consiste en realizar variaciones sobre la elección de un primer camino de prueba típico, denominado **camino básico**:

En resumen, Los pasos a seguir para realizar pruebas estructurales son:

1º.- Dibujar un grafo de flujo tomando como base el diseño o el propio código del programa.

2º.- Determinar la complejidad ciclomática de dicho grafo.

3º.- Determinar un conjunto básico de caminos linealmente independientes

4º.- Preparar casos de prueba para cada uno de los caminos del conjunto anterior.

- **Prueba de Condiciones.-** Se centra en probar cada una de las condiciones del programa con el fin de dar una orientación para la generación de pruebas adicionales del programa.

- **Prueba de flujo de datos.-** Selecciona caminos de prueba de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

- **Prueba de Bucles.-** Este tipo de prueba se centra exclusivamente en la validez de las construcciones de bucles. Pueden definirse cuatro tipos de bucles: simples, concatenados, anidados y no estructurados.

Los pasos a seguir para realizar una prueba de este tipo son:

Para los ***bucles simples***:

1º.- Pasar por alto totalmente el bucle.

2º.- Pasar una sola vez por el bucle.

3º.- Pasar dos veces por el bucle.

4º.- Pasar m veces por el bucle, con m < n (n = nº máximo de veces que el bucle puede ciclarse)

5º.- Hacer n – 1, n y n + 1 pasos por el bucle.

Para el **resto de tipos de bucles**:

1º.- Comenzar con el bucle interior manteniendo los parámetros de iteración de los bucles externos en sus valores mínimos

2º.- Progresar hacia fuera mientras se mantienen en sus valores mínimos los parámetros de iteración de los bucles externos y en los valores "típicos" los parámetros de iteración de los bucles internos.

3º.- Continuar hasta que se hayan probado todos los bucles.

9.5.- Pruebas funcionales

La prueba funcional o de caja negra se centra en el estudio de la especificación del software, del análisis de las funciones que debe realizar, de las entradas y de las salidas.

Conociendo la función específica para la que fue diseñado el producto. La prueba entonces demostrará si cada función es completamente operativa.

Este tipo de pruebas se suele utilizar para demostrar si fiabilidad del software, esto es, si las funciones son operativas, si la entrada se acepta de forma adecuada, si se produce una salida correcta, si se mantiene la integridad de la información externa.

En resumen, este tipo de pruebas examina aspectos del modelo fundamental del sistema sin tener especialmente en cuenta la estructura lógica interna del software.

Intenta encontrar errores de las siguientes categorías:

- **Funciones incorrectas o ausentes**
- **Errores de interfaz**
- **Errores en estructuras de datos o en accesos a bases de datos externas**
- **Errores de rendimiento**
- **Errores de inicialización y terminación.**

Entre las técnicas de diseño de pruebas de caja negra mas usuales se encuentran:

♦ *PARTICIÓN O CLASE DE EQUIVALENCIA.*- Este método divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba.

Para diseñar un buen caso de prueba:

- Cada caso debe cubrir el máximo número de entradas
- Debe tratarse de un dominio de valores de entrada dividido en un número finito de clases de equivalencia que cumplan la siguiente condición: "*La prueba de un valor representativo de una clase permite suponer 'razonablemente' que el resultado obtenido (existan defectos o no) será el mismo que el obtenido probando cualquier otro elemento de la clase*".

El método consiste en:

- Identificación de las clases de equivalencia
- Creación de los casos de prueba correspondientes

Para identificar las posibles clases de equivalencia de un programa a partir de su especificación se siguen los siguientes pasos:

- Identificación de las condiciones de entrada del programa, esto es, las restricciones de formato o contenido de los datos de entrada
- Identificación de las clases de equivalencia
    - De datos válidos
    - De datos no válidos o erróneos

El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Las clases de equivalencia pueden definirse de acuerdo con las siguientes directrices:

1º.- Si una condición de entrada es de ***"rango"*** se definen entonces una clase de equivalencia válida y dos no válidas.

2º.- Si una condición de entrada requiere un ***"valor"*** específico, se definen entonces una clase de equivalencia válida y dos no válidas.

3º.- Si una condición de entrada especifica un ***miembro de un conjunto***, se definen entonces una clase de equivalencia válida y otra no válida.

4º.- Si una condición de entrada es ***lógica***, se define una clase de equivalencia válida y una inválida.

El último paso del método es el uso de las clases de equivalencia para identificar los casos de prueba correspondientes. Este proceso consta de las siguientes fases:

- Asignación de un número único a cada clase de equivalencia
- Hasta que todas las clases de equivalencia sean cubiertas por casos de prueba se tratará de escribir tantas clases válidas no incorporadas como sea posible.
- Hasta que todas las clases de equivalencia no válidas hayan sido cubiertas por casos de prueba escribir un caso de prueba para una única clase no valida sin cubrir.

- *ANÁLISIS DE VALORES LÍMITE.-* La experiencia constata que los casos de prueba que exploran condiciones límite de un programa producen un mejor resultado para la detección de defectos, es decir, es mas probable que los defectos del software se acumulen en estas condiciones

Esta técnica permite determinar la elección de los casos de prueba que ejerciten los valores límite. Son directrices para este método:

1º.- Si una condición de entrada especifica un **rango** delimitado por los valores A y B se deben diseñar casos de prueba para los valores A y B, para los valores justo por debajo de A y de B y para los valores justo por encima de A y de B.

2º.- Si una condición de entrada especifica un número de **valores**, deben desarrollarse casos de prueba que ejerciten los valores máximo y mínimo, así como para los valores justo por debajo del máximo y del mínimo y para los valores justo por encima del máximo y del mínimo.

3º.- Aplicación de las directrices 1º y 2º a las condiciones de salida, ya que los valores límite de entrada no generan necesariamente valores límite de salida .

4º.- Si las estructuras de datos tiene **límites preestablecidos** (p.e. matrices) deben realizarse pruebas en sus límites.

♦ *PRUEBA DE COMPARACIÓN.-* En situaciones donde la fiabilidad del software sea algo absolutamente crítico suele realizarse el software de forma redundante para minimizar las posibilidades de error. En estas situaciones se prueba cada versión con los mismos datos de prueba para todas las versiones, para asegurar que todas tienen una salida idéntica.

Posteriormente se ejecutan todas las versiones en paralelo y se hace una comparación en tiempo real de los resultados para garantizar la consistencia.

9.6.- Pruebas aleatorias

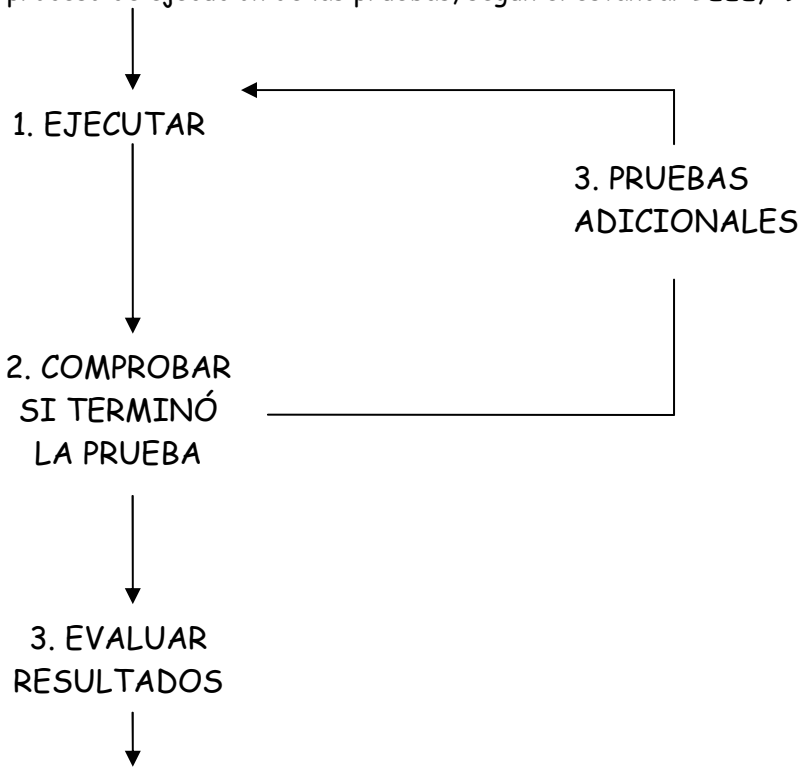
En las pruebas aleatorias se simula la entrada habitual del programa creando datos de entrada en la secuencia y con la frecuencia que podrían aparecer en la práctica, de forma continua, sin parar: esto implica el uso de una **herramienta** denominada **generador de pruebas** a la que se alimenta con una descripción de las entradas, de las secuencias de entrada posibles y su probabilidad de ocurrir en la práctica.

Si el proceso de generación se ha realizado correctamente, se crearán eventualmente todas las posibles entradas del programa en todas las posibles combinaciones y permutaciones.

Puede conseguirse también, indicando la distribución estadística que siguen, que la frecuencia de entradas sea la apropiada para orientar correctamente las pruebas a realizar hacia lo que es probable que suceda en la práctica.

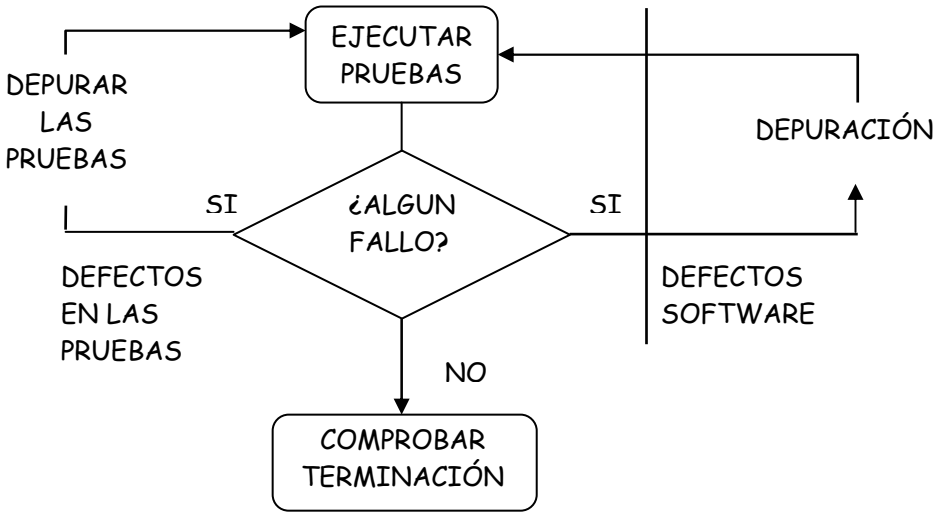
9.7.- Ejecución de las pruebas

El proceso de ejecución de las pruebas, según el estándar IEEE, 1986ª es:

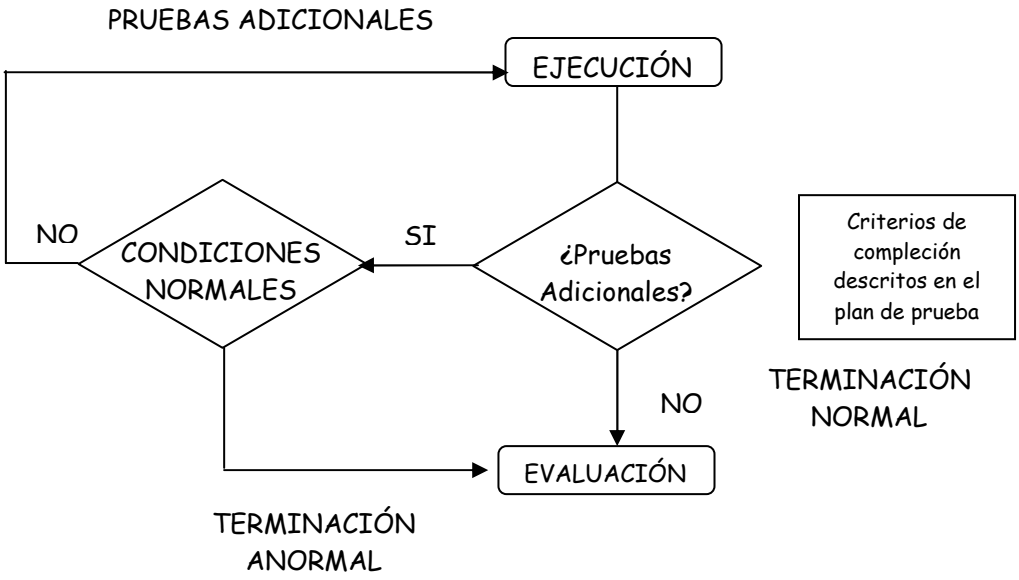


El proceso abarca las siguientes fases:

1. **Ejecutar las pruebas**, cuyos casos y procedimientos han sido ya diseñados previamente



2. **Comprobar si se ha concluido el proceso de prueba** según los criterios especificados en el plan de prueba



3. En el caso en el que se hayan concluido las pruebas se evalúan los resultados; en caso contrario se han de generar pruebas adicionales para que satisfagan los criterios de compleción de pruebas.

9.8.- Calidad del Software

En la vida cotidiana la **calidad** representa las propiedades inherentes a un objeto que permiten apreciarlo como mejor, igual o peor que otros objetos de su especie.

La calidad no es pues un concepto **absoluto** y puede considerarse como un concepto **multidimensional** (referida a muchas cualidades), sujeta a **restricciones** (por ejemplo, dependiente del presupuesto disponible), ligada a **compromisos aceptables** (por ejemplo, plazos de fabricación) ni es totalmente **subjetiva** (ciertos aspectos pueden medirse) ni totalmente **objetiva** (existen cualidades cuya evaluación solo puede hacerse subjetivamente). Además la calidad suele ser transparente cuando está presente pero resulta fácilmente reconocible cuando está ausente.

La definición de calidad mas completa es la de la norma ISO 8402 (admitida como norma española por AENOR con la denominación UNE 66-001-92) que define la calidad en general como:

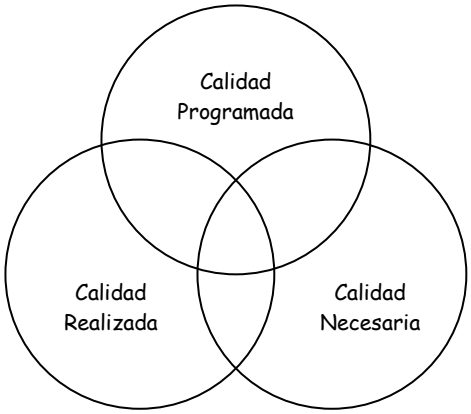
"*Totalidad de las características de un producto o servicio que le confieren su aptitud para satisfacer unas necesidades expresadas o implícitas*"

Esta definición permite comprender que la consecución de la calidad puede tener tres orígenes:

• ***La calidad realizada:*** *La que es capaz de obtener la persona que realiza el trabajo.(El ejemplo mas típico es el del programador que trabaja sin especificaciones), pero se refiere también al grado de cumplimiento de la especificación que el responsable de un trabajo es capaz de conseguir.*

• ***La calidad programada:*** *La que se ha pretendido obtener. Es la que aparece descrita en el documento de diseño, en una especificación etc.. Es, por tanto, la que se ha encomendado conseguir al responsable de ejecutar el trabajo.*

• ***La calidad necesaria:*** *La que el cliente exige con mayor o menor grado de concreción o, al menos, la que le gustaría obtener.*



La gestión de la calidad pretenderá conseguir que estos tres círculos coincidan entre sí. Todo lo que se encuentre fuera de dicha coincidencia será motivo de derroche, de gasto superfluo o de insatisfacción.

9.9.- Calidad en Ingeniería de Software

El software constituye un producto de características muy peculiares lo que provoca que las ideas sobre calidad creadas y aplicadas en otros sectores industriales tengan que adaptarse a esta situación. Así el software:

• ***Se desarrolla, no se fabrica en el sentido clásico del término.*** *Todo el coste de producción se centra en su diseño, ya que la replicación de un programa es una tarea trivial.*

• ***Se trata de un producto lógico, sin existencia física****. El verdadero producto del software es el diseño de una serie de instrucciones para el ordenador.*

• ***No se degrada con el uso****. Permanece inalterable por muy intensa que sea su utilización*

• *Su complejidad y la ausencia de controles lleva a que sea un producto que* ***se entrega conscientemente con defectos,*** *incluso públicamente declarados, algo inaceptable en el resto de los sectores productivos*

• *Un porcentaje muy grande de la producción* ***se hace a medida*** *en vez de emplear componentes existentes y ensamblarlos*

• *Es* ***extraordinariamente flexible,*** *pudiéndose cambiar con facilidad e incluso pudiéndose reutilizar trozos de un producto para construir otro.*

Dado que la definición estándar de software en ingeniería de software es "*los programas de ordenador, los procedimientos, la documentación asociada y los datos relativos a la operación del sistema informático*", la calidad no debe sólo limitarse al código. Debe tenerse en cuenta el software **en cualquier estado de evolución**: diseño, especificaciones, datos de prueba, etc. si se quiere un producto de calidad. De hecho, la calidad del software se debe obtener a medida que éste se construye.

9.9.1.- Definición de calidad del software

La definición oficial de calidad del software es la del estándar IEEE Std 610-1990:

Grado con el que un sistema, componente o proceso cumple:

- Los requisitos especificados
- Las necesidades o expectativas del cliente o usuario

Hay que recordar que:

- Los requisitos explícitamente establecidos se reflejan en la ERS (Especificación de Requisitos de Software), documento que constituye la culminación de la etapa de análisis dentro del proceso de desarrollo.
- Los requisitos establecidos en la ERS pueden ser:
  - Requisitos funcionales, *que determinan funciones a realizar por el software*
  - Requisitos de otros tipos*: de rendimiento, de seguridad, de interfaz, etc.*
- Los estándares y las normas de desarrollo determinan cómo se debe realizar el proceso de desarrollo de software. Su seguimiento permite que se consiga una calidad **técnica** el en software producido que influye en la calidad **de cara al usuario** o conformidad con los requisitos.
- Existen requisitos implícitos, no expresamente declarados, pero que el usuario del software desea obtener.

9.9.2.- Terminología sobre calidad

Términos normalmente aceptados en este área son:

• ***Gestión de calidad del software.-*** Aspecto de la función general de la gestión que determina y aplica la política de calidad (objetivos y directrices generales de calidad de una empresa).

Normalmente, la gestión de calidad se aplica a nivel de Empresa, por lo que incluye planificación estratégica, asignación de recursos, etc. aunque puede haber una gestión de calidad dentro de la gestión de cada proyecto.

• ***Aseguramiento de la calidad del software.-*** Conjunto de actividades planificadas y sistemáticas necesarias para aportar la confianza en que el producto (software) satisfará los requisitos dados de calidad.

Puede referirse también al *conjunto de actividades para evaluar el proceso mediante el cual se desarrolla el producto.*

En cualquier caso se debe evitar el término ***garantía de calidad*** ya que puede llevar a confusión con el concepto tradicional de garantía de los productos. El aseguramiento pretende dar confianza en que el producto tiene calidad.

• ***Control de calidad del software.-*** Técnicas y actividades de carácter operativo, utilizadas para satisfacer los requisitos relativos a la calidad, centradas en dos objetivos fundamentales: mantener bajo control un proceso y eliminar las causas de defectos en las diferentes fases del ciclo de vida..

• ***Verificación y validación del software.-*** Actividad ligada al control de la calidad en el ámbito del software:

> • ***Verificación***: Comprobación de que los productos construidos en una fase del ciclo de vida satisfacen los requisitos establecidos en la fase anterior. Se corresponde con las actividades para comprobar si un producto software está técnicamente bien construido, es decir, si funciona.
>
> • **Validación:** Comprobación de que el software construido satisface los requisitos del usuario. Se corresponde con las actividades de comprobación de que el producto software construido es el que se deseaba construir, esto es, si el producto funciona como el usuario quiere y hace las funciones que se habían solicitado

9.10.- Modelos tradicionales de evaluación de calidad del software

El modelo McCall se basa en descomponer el concepto de calidad en tres usos o **capacidades** importantes para un producto software desde el punto de vista del usuario del software:

* Capacidad de operación
* Capacidad para ser modificado o capacidad de revisión
* Capacidad de transición o adaptación a otros entornos.

Cada capacidad o uso se descompone en una serie de **factores** que determinan la calidad de cada una de las capacidades antes mencionadas. Para tener una visión apropiada a la calidad existen una serie de factores que pueden evaluarse mas fácilmente, cuyas definiciones son:

FACILIDAD DE USO: Grado de esfuerzo necesario para aprender a utilizar el software, preparar la entrada de datos e interpretar la salida del mismo.

INTEGRIDAD: Grado en el que se puede controlar el acceso del personal al software o a los datos que utiliza

EFICIENCIA: Necesidades de recursos hardware y software requeridos por el software evaluado para realizar sus funciones

FIABILIDAD: Grado o probabilidad de que el software no falle al realizar sus funciones

CORRECCIÓN O EXACTITUD: Grado en el que el software cumple sus especificaciones

FLEXIBILIDAD: Facilidad o grado de esfuerzo necesario para modificar el software en funcionamiento

FACILIDAD DE PRUEBA: Esfuerzo necesario para probar el software de modo que se tenga un cierto grado de confianza en que realiza adecuadamente sus funciones.

FACILIDAD DE MANTENIMIENTO: Facilidad o grado de esfuerzo para mantener operativo el software mediante la corrección o depuración de los problemas que puedan aparecer durante su funcionamiento.

TRANSPORTABILIDAD: Facilidad o grado de esfuerzo necesario para transportar o migrar el software de un entorno de operación a otro.

CAPACIDAD DE REUTILIZACIÓN: Capacidad o grado de esfuerzo para que el software o alguna de sus partes puedan ser utilizadas en otros desarrollos de software

CAPACIDAD DE INTEROPERACIÓN: Capacidad o grado de esfuerzo necesario para que el software o un sistema puedan operar conjuntamente con otros sistemas o aplicaciones de software.

Cada factor determinante anterior puede, a su vez, descomponerse en **criterios** o propiedades mas concretas que determinan la calidad.

Estas propiedades elementales o criterios son, en muchos casos, propiedades internas del software que no dependen en su apreciación de quien está observándolas y que los desarrolladores de software consideran que influyen en la calidad:

- FACILIDAD DE OPERACIÓN
- FACILIDAD DE COMUNICACIÓN
- FACILIDAD DE FORMACIÓN O APRENDIZAJE
- CONTROL DE ACCESOS (SEGURIDAD)
- FACILIDAD DE AUDITORÍA
- EFICIENCIA DE EJECUCIÓN
- EFICIENCIA DE ALMACENAMIENTO
- EXACTITUD O PRECISIÓN
- CONSISTENCIA
- TOLERANCIA A FALLOS
- MODULARIDAD
- SIMPLICIDAD
- COMPLECIÓN
- RASTREABILIDAD O FACILIDAD DE TRAZA
- AUTODESCRIPCIÓN
- CAPACIDAD DE EXPANSIÓN
- GENERALIDAD
- INSTRUMENTACIÓN
- INDEPENDENCIA ENTRE SISTEMA Y SOFTWARE
- INDEPENDENCIA DE LA MÁQUINA (DEL HARDWARE)
- NORMALIZACIÓN (O COMPATIBILIDAD) DE COMUNICACIONES
- NORMALIZACIÓN (O COMPATIBILIDAD) DE DATOS

Estos criterios pueden ser evaluados mediante un conjunto de **métricas o medidas** que se pueden calcular directamente observando el software o bien su proceso de desarrollo.

9.10.1.- Métricas de software

En general, para la evaluación de la calidad, lo mas habitual es centrarse en medidas del producto mas que en medidas de procesos. Entre las métricas mas conocidas se encuentran:

* **Métricas basadas en el texto del código:**
  - Número de líneas de código
  - Número de líneas de comentario
  - Número de instrucciones
  - Porcentaje de líneas de comentario dentro del total de líneas de código
  - Número de instrucciones GOTO
  - etc.
* **Métricas basadas en la estructura de control del código:**
  - Relacionadas con el control intramodular, basadas en su grafo de control (métrica de McCabe y otras)
  - Relacionadas con la arquitectura de módulos, basadas en el grafo de llamadas o en el diagrama de estructuras (acoplamiento y cohesión).