# CAPÍTULO 3. ASEGURAMIENTO DE CALIDAD: PRUEBAS DE SOFTWARE

## 3.1. PRINCIPIOS DE PRUEBAS. ENFOQUE ESTRATEGICO.

**0. CONSIDERACIONES INICIALES.** Las pruebas son un conjunto de actividades que se pueden planificar y ejecutar sistemáticamente. Se han propuesto varias estrategias de prueba del software en distintos libros. Todas proporcionan al ingeniero del software una plantilla para la prueba y todas tienen las siguientes características genéricas:

- Para realizar pruebas efectivas un equipo de software debe ejecutar Revisiones Técnicas Formales y efectivas. Esto eliminará muchos errores antes que empiece una prueba.
- Las pruebas comienzan a nivel de módulo/clase/componente y trabajan "hacia fuera", hacia la integración de todo el sistema.
- Según el momento, son apropiadas diferentes técnicas de prueba.
- La prueba la lleva a cabo el responsable del desarrollo del software y (para grandes proyectos) un grupo independiente de pruebas.
- La prueba y la depuración son actividades diferentes, pero la depuración se debe incluir en cualquier estrategia de prueba.

# 1. VERIFICACIÓN Y VALIDACIÓN

La prueba del software es un elemento de un tema más amplio que, a menudo, es conocido como verificación y validación (V & V). La verificación se refiere al conjunto de actividades que aseguran que el software implementa correctamente una función específica. La validación se refiere a un conjunto diferente de actividades que aseguran que el software construido se ajusta a los requisitos del cliente. Algunos lo definen así:

***Verificación: ¿Estamos construyendo el producto correctamente?***
***Validación: ¿Estamos construyendo el producto correcto?***

La verificación y la validación abarcan una amplia lista de actividades SQA que incluye: revisiones técnicas formales, auditorías de calidad y de configuración, monitorización de rendimientos, simulación, estudios de factibilidad, revisión de la documentación, revisión de la base de datos, análisis algorítmico, pruebas de desarrollo, pruebas de validación y pruebas de instalación.

Las pruebas constituyen el último bastión para evaluar la calidad y descubrir los errores. Pero las pruebas no deben ser vistas como una red de seguridad. Ojo a esto: "Si la calidad no está ahí antes de comenzar la prueba, no estará cuando se termine."

La calidad se incorpora en el software durante el proceso de ingeniería del software. La aplicación adecuada de los métodos y de las herramientas, las revisiones técnicas formales efectivas y una sólida gestión y medición, conducen a la calidad, que se confirma durante las pruebas.

## 2. ORGANIZACIÓN DE PRUEBAS.

| DISCIPLINA | CATEGORIA DE PRUEBA ASOCIADA |
|---|---|
| Construcción | ***Prueba de Unidad.*** Se centra en cada unidad del software (módulo/clase/componente), tal como está implementada en código fuente. Su naturaleza tiende a ser exhaustiva, y se utilizan los enfoques de caja negra y de caja blanca. |
| Diseño | ***Prueba de Integración.*** El foco de atención es el diseño y la construcción de la arquitectura del software. Hay varios estilos (ascendente / descendente); aquí también se prueban interfaces. Normalmente hay que re-ejecutar o combinar algunas de las pruebas que se hicieron en el estado unitario. Se utiliza enfoque de caja negra y ocasionalmente de caja blanca. |
| Requisitos. | **Prueba de validación.** Se validan los requisitos establecidos como parte del análisis de requisitos del software, comparándolos con el sistema que ha sido construido. Se utiliza exclusivamente enfoque de caja negra. Se responde a la pregunta de si satisface todos los requisitos funcionales, además de los de comportamiento y de rendimiento.<br><br>Incluye las denominadas "pruebas de aceptación" en sus modalidades alfa y beta. |
| | ***Prueba del sistema.*** El software, una vez validado, se debe combinar con otros elementos del sistema (por ejemplo, hardware, gente, bases de datos). Se verifica que cada elemento encaja de forma adecuada y que se alcanza la funcionalidad y el rendimiento del sistema total.<br><br>Abarca una serie de pruebas diferentes como son: prueba de recuperación, prueba de seguridad, prueba de resistencia (o de stress) y prueba de desempeño. |

## 3. PRINCIPIOS DEL PROCESO DE PRUEBAS DE SOFTWARE.

Desde un punto de vista psicológico, el análisis y el diseño del software (junto con la codificación) son tareas constructivas. El ingeniero del software crea un programa de computadora, su documentación y sus estructuras de datos asociadas.

Cuando comienza la prueba, aparece una sutil, aunque firme intención de "romper" lo que el ingeniero del software ha construido. Desde el punto de vista del constructor, la prueba se puede considerar (psicológicamente) destructiva. Por tanto, el constructor anda con cuidado, diseñando y ejecutando pruebas que demuestren que el programa funciona, en lugar de detectar errores. ***Desgraciadamente, los errores seguirán allí. Y si el ingeniero del software no los encuentra, ¡el cliente lo hará!***

| ***A menudo, existen ciertos malentendidos que se pueden deducir...*** | ***A las que podemos responder o complementar lo siguiente:*** |
|---|---|
| [1] El responsable del desarrollo no debería entrar en el proceso de prueba; | …El error está en que no definimos en cuáles etapas no debe entrar y en cuáles sí. |
| [2] El software debe ser "puesto a salvo" de extraños que puedan probarlo de forma despiadada. | Esta idea es la que debemos combatir: si no lo probamos, ¿cómo reducimos el riesgo de aparición de errores? |
| [3] Los encargados de la prueba sólo aparecen en el proyecto cuando comienzan las etapas de prueba. | …No hay que esperar a las pruebas, el aseguramiento de calidad es permanente y desde el primer día. |

El responsable del desarrollo del software siempre es responsable de probar las unidades individuales (módulos) del programa, asegurándose de que cada una lleva a cabo la función para la que fue diseñada. En muchos casos, también se encargará de la prueba de integración, el paso de las pruebas que lleva a la construcción (y prueba) de la estructura total del sistema.

Sólo una vez que la arquitectura del software esté completa entra en juego un grupo independiente de prueba también denominado GIP. (Bueno... de todas formas siempre es bueno que los módulos/componentes/clases/ más críticos sí sean auditados por un GIP). El noble propósito de un GIP es el de eliminar el conflicto de intereses que, de otro modo, estaría presente. Después de todo, al personal del equipo que forma el grupo independiente se le paga para que encuentre errores.

Sin embargo, el responsable del desarrollo del software no entrega simplemente el programa al GIP y se desentiende. El responsable del desarrollado y el GIP **trabajan estrechamente a lo largo del proyecto de software** para asegurar que se realizan pruebas "exhaustivas" (y ojo que lo escribo entre comillas). Mientras se realiza la prueba, el desarrollador debe estar disponible para corregir los errores que se van descubriendo.

OJO a la frase:

**"El primer error que comete la gente es pensar que el equipo de pruebas es responsable de asegurar la calidad".**

# 4. PRINCIPIOS DE PRUEBAS. ENFOQUE ESTRATÉGICO.

Tom Gilb (creador de EVO) plantea lo siguiente para implementar una estrategia exitosa de prueba del software:

***Especificar los requisitos del producto de manera cuantificable mucho antes de que comiencen las pruebas.*** Aunque el objetivo principal de la prueba es encontrar errores, una buena estrategia de prueba también evalúa otras características de la calidad, tales como la portabilidad, facilidad de mantenimiento y facilidad de uso. Todo esto debería especificarse de manera que sea medible para que los resultados de la prueba no sean ambiguos.

***Establecer los objetivos de la prueba de manera explícita.*** Se deberían establecer en términos medibles los objetivos específicos de la prueba. Por ejemplo, la efectividad de la prueba, la cobertura de la prueba, tiempo medio de fallo, el coste para encontrar y arreglar errores, densidad de fallos remanente o frecuencia de ocurrencia, y horas de trabajo por prueba de regresión deberían establecerse dentro de la planificación de la prueba.

***Comprender qué usuarios van a manejar el software y desarrollar un perfil para cada categoría de usuario.*** Usar casos que describan el escenario de interacción para cada clase de usuario pudiendo reducir el esfuerzo general de prueba concentrando la prueba en el empleo real del producto.

***Usar revisiones técnicas formales efectivas como filtro antes de la prueba.*** Las revisiones técnicas formales pueden ser tan efectivas como las pruebas en el descubrimiento de errores. Por este motivo, las revisiones pueden reducir la cantidad de esfuerzo de prueba necesaria para producir software de alta calidad.

***Llevar a cabo revisiones técnicas formales para evaluar la estrategia de prueba y los propios casos de prueba.*** Las revisiones técnicas formales pueden descubrir inconsistencias, omisiones y errores claros en el enfoque de la prueba. Esto ahorra tiempo y también mejora la calidad del producto.

***Desarrollar un enfoque de mejora continua al proceso de prueba.*** Debería medirse la estrategia de prueba. Las métricas agrupadas durante la prueba deberían usarse como parte de un enfoque estadístico de control del proceso para la prueba del software.

# 5. PRINCIPIOS DE PRUEBAS. ENFOQUE OPERATIVO.

## 5.1. Conceptos Básicos de Testing

- Un test exitoso es aquel que encuentra muchos defectos, no lo opuesto - por lo tanto, desarrollo exitoso puede llevar a testing no exitoso (no encuentra defectos).
- El propósito del testing es encontrar defectos - es un proceso destructivo, se debe mostrar que algo es incorrecto - no es recomendable probar los propios desarrollos.
- Cada vez que se corrigen defectos detectados, se pueden introducir nuevos defectos al sistema.

Testing es el proceso de ejecutar un programa con la intención de encontrar errores/no conformidades/fallos. Es un proceso con enfoque destructivo. (Encontrarlos y eliminarlos).

**Testing NO es:**

- Demostración de que no hay errores.
- Demostración de que el software desempeña correctamente sus funciones.
- Establecimiento de confianza que un programa hace lo que debe hacer.

## 5.2. Principios de testing

- Una parte necesaria de un caso de prueba es la definición de la salida o resultado esperado.
- Un programador debería evitar probar su propio código, o al menos, ser el único que lo prueba.
- Una unidad de programación no debería probar sus propios programas.
- Los resultados de cada prueba deben ser acuciosamente inspeccionados.
- Los casos de prueba deben diseñarse para condiciones de entrada inválidas e inesperadas, no solo para aquellas válidas y esperadas.

Examinar un programa para ver si hace lo que debe hacer es solo la mitad de la tarea; la otra mitad es ver que no hace lo que no debe hacer. (No es un trabalenguas).

- Evitar casos de prueba espontáneos y que no dejan registro - es solo pérdida de tiempo.
- No planificar un esfuerzo de prueba bajo el supuesto tácito que no se encontrará defectos.
- La probabilidad de existencia de más defectos en una sección de un programa es proporcional al número de defectos ya detectados en dicha sección.
- Testing es una tarea creativa e intelectualmente desafiante.

## 5.3. Testing vs Debugging.

El propósito del testing es mostrar que un programa tiene defectos, mientras el propósito del debugging es encontrar el error o mala concepción que llevó a la falla, y diseñar e implementar los cambios que corrijan el error.

- El debugging usualmente sigue al testing.
- El testing lo puede hacer alguien externo, el debugging no.

## 5.4. Aspectos económicos.

- ¿Es posible probar un programa para encontrar todos los defectos? ---- No, ni siquiera para los programas más triviales.
- En general, es impráctico y a menudo imposible encontrar todos los defectos en un programa.
  - Habría que probar todas las combinaciones de entrada, correctas e incorrectas ..... número (prácticamente) infinito de casos de prueba
  - Habría que probar todos los caminos posibles en un programa, que pueden contener ciclos, el número de casos de prueba no es infinito, pero puede considerarse como tal.

## 5.5. Estándar para testing

ISO/IEC 29119 *(http://www.softwaretestingstandard.org)*

Parte 1 – Conceptos y definiciones

Parte 2 – Modelo de procesos de pruebas

Parte 3 – Documentación de pruebas

Parte 4 – Técnicas de prueba

Parte 5 – Pruebas dirigidas por palabras clave (busca estandarizar automatización)

## 5.6. Resumiendo los Principios de testing

- Todas las pruebas deben ser trazables a requerimientos del usuario.
- Las pruebas deben ser planificadas mucho antes de comenzarlas.
- La ley de Pareto aplica al testing de software.
- Testing debería comenzar en pequeña escala y progresar.
- Testing exhaustivo no es posible.
- Para mayor efectividad, testing debería ser desarrollado por una tercera parte independiente.

***---------- FIN DEL DOCUMENTO***