20

**Ingeniería de Software**

**Controles de Software**

**Páez, Daniel M**

Uno de las partes importantes en las que consiste el proceso de desarrollo de software es la verificación, validación y pruebas del mismo. Verificación y validación son términos similares pero con diferencias importantes en relación con los requisitos del software.

El proceso que se lleva a cabo con ambos términos en conjunto con la planificación de las pruebas permite la presentación e identificación temprana de errores, característica esencial previniendo posibles gastos imprevistos en cuanto a costo por detección tardía de fallos u errores.

Tabla de contenido

[Verificación y Validación del Software. 3](#_Toc41667213)

[Pruebas de Desarrollo. 6](#_Toc41667214)

[Desarrollo Dirigido por Pruebas. 6](#_Toc41667215)

[Pruebas de Versión. 6](#_Toc41667216)

[Pruebas de Usuario. 6](#_Toc41667217)

[Conclusiones. 6](#_Toc41667218)

[Bibliografía. 6](#_Toc41667219)

# Verificación y Validación del Software.

Comprende el conjunto de procesos de comprobación y análisis que aseguran que el software que se desarrolla está acorde a su especificación y cumple las necesidades de los clientes.

* **Verificación ¿Estamos construyendo el producto correctamente?:** proceso que comprueba que el software está de acuerdo a como se especificó. Se comprueba que satisface los requerimientos funcionales y no funcionales.
* **Validación ¿Estamos construyendo el producto correcto?:** proceso de asegurar que el sistema software satisface las expectativas del cliente.

Este proceso, llamado V&V, comprueba durante y después del proceso de implementación del software, para asegurar que satisfaga su especificación y su entrega funcional esperada por aquellos que pagaron por él. Comienza con revisiones de los requerimientos, para continuar con revisiones en el diseño e inspeccionar el código, hasta llegar a probar el producto.

El objetivo del proceso V&V es establecer la seguridad de que el sistema software está hecho para un propósito, por lo cual el sistema debe ser lo suficientemente bueno para su uso. Para esto, el nivel de confianza depende de:

1. Función del Software: depende de lo crítico que sea el software para la organización.
2. Expectativas del Usuario: muchos usuarios tienen pocas expectativas sobre su software y no se sorprenden cuando este falla durante su uso. Aceptan estos fallos del sistema cuando los beneficios de su uso son mayores que sus desventajas.
3. Entorno del Mercado: cuando se comercializa el sistema, los vendedores del sistema deben tener en cuenta los programas competidores, el precio que están dispuestos a pagar los clientes por el sistema y la agenda para la entrega.

En V&V, existen dos aproximaciones complementarias para el análisis y comprobación de sistemas:

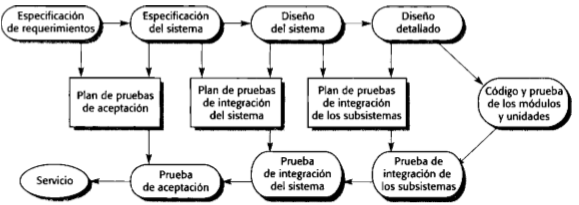
* **Las inspecciones de software** analizan y comprueban las representaciones del sistema tales como el documento de requerimientos, diagramas de diseño y código fuente del programa. Las inspecciones se usan en todas las etapas del proceso. Estas inspecciones de software y los análisis automáticos son técnicas de V&V estáticas, ya que no se necesita ejecutar el software en una computadora.
* **Las pruebas del software** implican la ejecución de una implementación del software con datos de prueba. Examinando las salidas y su entorno operacional para comprobar que funciona tal y como se requiere. Las pruebas son una técnica dinámica de V&V.



A través de las inspecciones sabemos cómo se va avanzando, mientras que con las pruebas, permitirá saber si el software funciona.

El proceso de V&V es un proceso caro (por ejemplo en sistemas de tiempo real con restricciones no funcionales complejas, más de la mitad del presupuesto para el desarrollo de estos se invierte en V&V). Para evitar estos elevados costos y obtener el máximo provecho en las inspecciones y pruebas y controlar los costes del proceso de verificación y validación se debe empezar la planificación de este en etapas tempranas del proceso del desarrollo.

El siguiente modelo es denominado a veces modelo V (instanciación del modelo genérico de cascada), muestra que los planes de pruebas deberían derivarse a partir de la especificación y diseño del sistema, dividiendo el proceso de V&V en varias etapas, por lo que cada etapa esta conducida por las pruebas que tienen que definirse para comprobar la conformidad del programa con su diseño y especificación.



Los siguientes son los principales componentes de un plan de pruebas para un sistema grande y complejo:

* **Proceso de Pruebas**: descripción de las principales fases del proceso de pruebas.
* **Trazabilidad de Requerimientos**: son los usuarios los más interesados en que el sistema satisfaga sus requerimientos y las pruebas deben planificarse para que todos los requerimientos se prueben de manera individual.
* **Elementos Probados**: se deben especificar los elementos del proceso del software que tienen que ser probados.
* **Calendario de Pruebas**: un calendario de todas las pruebas y las asignaciones de recursos para este calendario se enlaza, obviamente, con la agenda general del desarrollo del proyecto.
* **Procedimientos de Registro de las Pruebas**: no es suficiente ejecutar simplemente las pruebas; los resultados de las pruebas deben ser registrados sistemáticamente. Debe ser posible auditar el proceso de pruebas para comprobar que se ha llevado a cabo correctamente.
* **Requerimientos de Hardware y Software**: se debería determinar las herramientas software requerido y la utilización estimada del hardware.
* **Restricciones:** se deberían anticiparse las restricciones que afectaran a los procesos de pruebas como la escasez de personal.

Durante las **Inspecciones de Software:**

* **Se examina la fuente de representación, tratando de encontrar anomalías y defectos.**
* **No se necesitan ejecutar el sistema, se pueden usar antes de la ejecución.**
* **Es aplicable a cualquier representación del sistema (Requisitos, Diseño, Configuración de Datos, Datos de Ensayo, etc.).**

El éxito de las inspecciones depende de:

* Muchos defectos pueden descubrirse en una sola inspección.
* La reutilización de dominio y conocimientos de programación estiman la probabilidad de encontrar errores comunes a través de los encuestados.
* Inspecciones y Pruebas son complementarias y no técnicas opuestas de verificación.

Un método de Verificación son los **Métodos Formales,** los cuales son basados en representaciones matemáticas de software, como especificación formal. Estos métodos se ocupan del análisis matemático de la especificación, transformándola a una representación más detallada semánticamente equivalente, o verificar formalmente que una representación del sistema es semánticamente equivalente a otra representación.

Estos métodos requieren un análisis más detallado de la especificación del sistema y del programa, lo cual su uso consume tiempo y resulta muy caro, como consecuencia el uso de estos métodos formales se restringen principalmente a los procesos de desarrollo de software críticos y seguros.

Estos métodos se pueden usar en diferentes etapas de V&V:

1. Se puede desarrollar una especificación formal del sistema y analizarse matemáticamente para buscar inconsistencias. Es efectiva esta técnica para descubrir errores y omisiones de especificación.
2. Se puede verificar formalmente, usando argumentos matemáticos, para que el cogido de un sistema software es consistente con su especificación. Se requiere una especificación formal y es efectiva para descubrir errores del diseño y programación.

Las desventajas de los métodos formales:

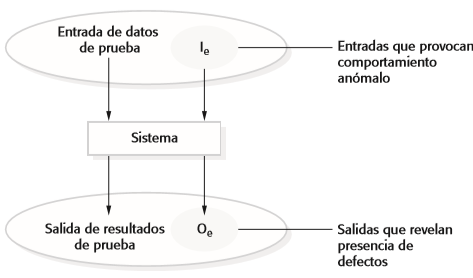
* La especificación puede no reflejar los requerimientos reales de los usuarios del sistema.
* La demostración podría contener errores.
* La demostración puede asumir un patrón de uso que es incorrecto.

# Pruebas de Software.

Las pruebas intentan demostrar que un programa hace lo que se intenta que haga, así como descubrir defectos en el programa antes de usarlo. Al probar el software, se ejecuta un programa con datos artificiales. Hay que verificar los resultados de la prueba que se opera para buscar errores, anomalías o información de atributos no funcionales del programa.

Las pruebas tienen como objetivo:

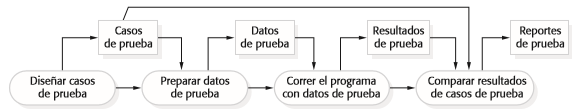
* Demostrar al desarrollador y al cliente que el software cumple con los requerimientos.
* Encontrar situaciones donde el comportamiento del software sea incorrecto, indeseable o no esté de acuerdo con su especificación. Tales situaciones son consecuencia de defectos del software.



Al igual que las pruebas de software, el proceso de verificación y validación implicaría inspecciones y revisiones de software. Estas últimas analizan y comprueban los requerimientos del sistema, los modelos de diseño, el código fuente del programa, e incluso las pruebas propuestas para el sistema. Éstas son las llamadas técnicas V&V “estáticas” donde no es necesario ejecutar el software para verificarlo.

Las inspecciones se enfocan principalmente en el código fuente de un sistema, aun cuando cualquier representación legible del software, como sus requerimientos o modelo de diseño, logre inspeccionarse. Cuando un sistema se inspecciona, se utiliza el conocimiento del sistema, su dominio de aplicación y el lenguaje de programación o modelado para descubrir errores. Las inspecciones no sustituyen las pruebas del software, ya que no son eficaces para descubrir defectos que surjan por interacciones inesperadas entre diferentes partes de un programa, problemas de temporización o dificultades con el rendimiento del sistema.

En un modelo de pruebas tradicional los casos de prueba son especificaciones de las entradas a la prueba y la salida esperada del sistema (los resultados de la prueba), además de que contienen la información sobre lo que se pone a prueba. Los datos de prueba son las entradas que se diseñaron para probar un sistema. En ocasiones pueden generarse automáticamente datos de prueba; no obstante, es imposible la generación automática de casos de prueba. Sin embargo, es posible automatizar la ejecución de pruebas. Los resultados previstos se comparan automáticamente con los resultados establecidos, de manera que no haya necesidad de que un individuo busque errores y anomalías al correr las pruebas.



## Pruebas de Desarrollo.

Incluyen todas las actividades de prueba que realiza el equipo que elabora el sistema. El examinador del software suele ser el programador que diseñó dicho software, aunque éste no es siempre el caso.

Para sistemas críticos, puede usarse un proceso más formal, con un grupo de prueba independiente dentro del equipo de desarrollo. Son responsables del desarrollo de pruebas y del mantenimiento de registros detallados de los resultados de las pruebas.

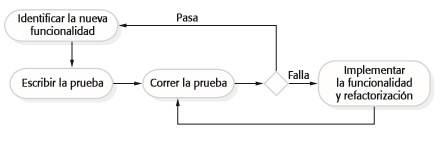
En el desarrollo, las pruebas se realizan en tres niveles de granulación:

* **Pruebas de Unidad:** se ponen a prueba unidades de programas o clases de objetos individuales. Las pruebas de unidad deben enfocarse en comprobar la funcionalidad de objetos o métodos.
* **Pruebas del Componente:** muchas unidades individuales se integran para crear componentes compuestos. Las pruebas de componentes deben enfocarse en probar interfaces del componente.
* **Pruebas del Sistema:** algunos o todos los componentes en un sistema se integran y el sistema se prueba como un todo. Las pruebas del sistema deben enfocarse en poner a prueba las interacciones de los componentes.

Las pruebas de desarrollo son, ante todo, un proceso de prueba de defecto, en las cuales la meta consiste en descubrir bugs en el software. Por lo tanto, a menudo están entrelazadas con la depuración: el proceso de localizar problemas con el código y cambiar el programa para corregirlos.

# Desarrollo Dirigido por Pruebas.

El desarrollo dirigido por pruebas (TDD, Test Driven Development) es un enfoque al diseño de programas donde se entrelazan el desarrollo de pruebas y el del código. El código se desarrolla incrementalmente, junto con una prueba para ese incremento. No se avanza hacia el siguiente incremento sino hasta que el código diseñado pasa la prueba. El desarrollo dirigido por pruebas se introdujo como parte de los métodos ágiles como la programación extrema.



Los pasos del proceso de TDD son:

1. Se identifica el incremento de funcional requerido. Este debe ser pequeño y aplicable en pocas líneas de código.
2. Se escribe una prueba para esa funcionalidad, implementándose como una prueba automatizada. Esto significa que la prueba puede ejecutarse y reportarse, sin importar si aprueba o falla.
3. Se corre la prueba, junto con todas las otras pruebas que se implementaron. Al principio no se aplica la funcionalidad, de modo que la primera prueba fallara.
4. Luego se implementa la funcionalidad y se opera nuevamente la prueba. En este paso se debe incluir la refactorización del código existente, para perfeccionarlo y adicionar nuevo código al que ya existe.
5. Una vez puestas en funcionamiento con éxito todas las pruebas, se avanza a la implementación de la siguiente funcionalidad.

Un entorno automatizado de pruebas, como el entorno JUnit que soporta pruebas del programa Java es esencial para TDD. Conforme el código se desarrolla en incrementos muy pequeños, uno tiene la posibilidad de correr cada prueba, cada vez que se adiciona funcionalidad o se refactoriza el programa. De esta forma, en pocos segundos se ejecutan cientos de pruebas independientes.

En el desarrollo dirigido por pruebas ayuda a los programadores a aclarar sus ideas acerca de lo que realmente debe hacer un segmento de código. Para escribir una prueba, es preciso entender lo que se quiere, pues esta comprensión facilita la escritura del código requerido.

Además de la mejor comprensión del problema, TDD tiene otros beneficios:

* **Cobertura de código:** cualquier segmento de código que se escriba debe tener al menos una prueba asociada. El código que se prueba a medida que se escribe, de manera que los defectos se descubren con oportunidad en el proceso de desarrollo.
* **Pruebas de Regresión:** Un conjunto de pruebas se desarrolla incrementalmente conforme se desarrolla un programa. Siempre es posible correr pruebas de regresión para demostrar que los cambios al programa no introdujeron nuevos errores.
* **Depuración simplificada:** al fallar una prueba debe ser evidente descubrir dónde surge el problema. Es preciso comprobar y modificar el código recién escrito. No se requieren herramientas de depuración para localizar el problema.
* **Documentación del sistema:** Las pruebas en sí actúan como una forma de documentación que describen lo que debe hacer el código. Leer las pruebas suele facilitar la comprensión del código.

El desarrollo dirigido por pruebas resulta ser un enfoque exitoso para proyectos de dimensión pequeña y mediana. Por lo general, los programadores que adoptan dicho enfoque están contentos con él y descubren que es una forma más productiva de desarrollar software.

# Pruebas de Versión.

Las pruebas de versión son el proceso de poner a prueba una versión particular de un sistema que se pretende usar fuera del equipo de desarrollo. Por lo general, la versión del sistema es para clientes y usuarios. No obstante, en un proyecto complejo, la versión podría ser para otros equipos que desarrollan sistemas relacionados.

Existen dos distinciones importantes entre las pruebas de versión y las pruebas del sistema durante el proceso de desarrollo:

1. Un equipo independiente que no intervino en el desarrollo del sistema debe ser el responsable de las pruebas de versión.
2. Las pruebas del sistema por parte del equipo de desarrollo debe enfocarse en el descubrimiento de bugs en el sistema (prueba de defectos). El objetivo de las pruebas de versión es comprobar que el sistema cumpla con los requerimientos y sea suficientemente bueno para uso externo (pruebas de validación).

La principal meta del proceso de pruebas de versión es convencer al proveedor del sistema de que éste es suficientemente apto para su uso. Si es así, puede liberarse como un producto o entregarse al cliente. Por lo tanto, las pruebas de versión deben mostrar que el sistema entrega su funcionalidad, rendimiento y confiabilidad especificados, y que no falla durante el uso normal.

Las pruebas de versión, por lo regular, son un proceso de prueba de caja negra, donde las pruebas se derivan a partir de la especificación del sistema, donde se estudia el comportamiento a determinadas entradas y las salidas correspondientes al procesamiento de ellas.

Algunas de las pruebas que se realizan en cada versión de software son:

* **Pruebas Basadas en Requerimientos:** Un principio general de buena práctica en la ingeniería de requerimientos es que éstos deben ser comprobables: *los requerimientos tienen que escribirse de forma que pueda diseñarse una prueba para dicho requerimiento*. Luego, un examinador comprueba que el requerimiento se cumpla. En consecuencia, las pruebas basadas en requerimientos son un enfoque sistemático al diseño de casos de prueba, donde se considera cada requerimiento y se deriva un conjunto de pruebas para éste. Las pruebas basadas en requerimientos son pruebas de validación más que de defecto: se intenta demostrar que el sistema implementó adecuadamente sus requerimientos.
* **Pruebas de Escenario:** son un enfoque a las pruebas de versión donde se crean escenarios típicos de uso y se les utiliza en el desarrollo de casos de prueba para el sistema. Un escenario es una historia que describe una forma en que puede usarse el sistema. Estos escenarios deben ser realistas y los usuarios reales tienen que relacionarse con ellos.
* **Pruebas de Rendimiento:** Una vez integrado completamente el sistema, es posible probar propiedades emergentes, como el rendimiento y la confiabilidad. Las pruebas de rendimiento deben diseñarse para garantizar que el sistema procese su carga pretendida. Generalmente, esto implica efectuar una serie de pruebas donde se aumenta la carga, hasta que el rendimiento del sistema se vuelve inaceptable. Para probar si los requerimientos de rendimiento se logran, quizá se deba construir un perfil operativo. Un perfil operativo es un conjunto de pruebas que reflejan la mezcla real de trabajo que manejará el sistema.

# Pruebas de Usuario.

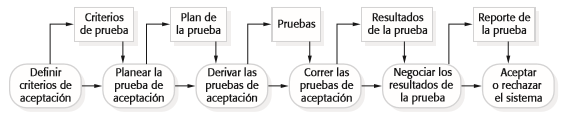
Las pruebas de usuario o del cliente son una etapa en el proceso de pruebas donde los usuarios o clientes proporcionan entrada y asesoría sobre las pruebas del sistema. Las pruebas de usuario son esenciales, aun cuando se hayan realizado pruebas abarcadoras de sistema y de versión. La razón de esto es que la influencia del entorno de trabajo del usuario tiene un gran efecto sobre la fiabilidad, el rendimiento, el uso y la robustez de un sistema.

Hay tres diferentes tipos de pruebas de usuario:

* **Pruebas Alfa** donde los usuarios del software trabajan con el equipo de diseño para probar el software en el sitio del desarrollador.
* **Pruebas Beta** donde una versión del software se pone a disposición de los usuarios, permitiéndoles experimentar y descubrir problemas que encuentran con los desarrolladores del sistema.
* **Pruebas de Aceptación** dondelos clientes prueban un sistema para decidir si está o no listo para ser aceptado por los desarrolladores del sistema y desplegado en el entorno del cliente**.**

En las pruebas de aceptación existen seis etapas, las cuales son:

1. **Definir los criterios de aceptación:** debe de anticiparse en el proceso, antes de firmar el contrato por el sistema. Los criterios de aceptación forman parte del contrato del sistema y tienen que ser lo más convenientes posibles para el cliente y el desarrollador (siendo esto difícil de lograrlo, debido a que es posible que no estén detallados los requerimientos y que haya cambios significativos de estos durante el desarrollo).
2. **Plan de pruebas de aceptación:** incluye decidir sobre los recursos, el tiempo y el presupuesto para las pruebas de aceptación, así como establecer un calendario de pruebas.
3. **Derivar pruebas de aceptación:** establecidos los criterios de aceptación, tienen que diseñarse pruebas para comprobar si un sistema es aceptable o no. Las pruebas de aceptación deben dirigirse a probar tanto las características funcionales como las no funcionales del sistema. Lo ideal sería que dieran cobertura completa a los requerimientos del sistema.
4. **Correr pruebas de aceptación:** las pruebas acordadas se ejecutan sobre el sistema. De manera ideal, esto debería ocurrir en el entorno real donde se usará el sistema, pero esto podría ser perturbador y poco práctico. En consecuencia, quizá deba establecerse un entorno de pruebas de usuario para efectuar dichas pruebas.
5. **Negociar los resultados de las pruebas:** Es poco probable que se pasen todas las pruebas de aceptación definidas y que no haya problemas con el sistema. Si éste es el caso, entonces las pruebas de aceptación están completas y el sistema está listo para entregarse. Con mayor regularidad se descubrirán algunos problemas. En tales casos, el desarrollador y el cliente tienen que negociar para decidir si el sistema es suficientemente adecuado para ponerse en uso.
6. **Rechazo/Aceptación del sistema:** Esta etapa incluye una reunión entre los desarrolladores y el cliente para decidir si el sistema debe aceptarse o no. Si el sistema no es suficientemente bueno para usarse, entonces se requiere mayor desarrollo para corregir los problemas identificados. Una vez completo, se repite la fase de pruebas de aceptación.



# Conclusiones.

Desarrollar un software no es un proceso sencillo que solo conlleva la especificación de requisitos y elegir el modelo de desarrollo que lo llevara a cabo, sino que es una constante negociación con el cliente, determinando que procesos son esenciales, planteando escenarios donde el software podría fallar, verificar y validar (estos términos deben ser tomados como un proceso constante y no solo como centrarse en comprobar que se puede realizar todo lo especificado y solo lo especificado) en todo el proceso de desarrollo, idear un plan de pruebas desde antes de desarrollar el sistema, durante y después de construirlo, e incluso diseñar pruebas en los mismos usuarios para poder llegar al acuerdo o rechazo del sistema. Se comprende que el desarrollo de pruebas es esencial para el proceso de desarrollo de software, las pruebas permitirán descubrir errores nuevos en el sistema, pero esto no significara que el sistema esté libre de ellos.

# Bibliografía.

Ingeniería de Software – Ian Summerville – 7ma y 9na Edición.