# 



PRUEBAS



INDICE

[Pruebas 1](#_Toc455588392)

[Introducción a la Prueba 3](#_Toc455588393)

[La Prueba 3](#_Toc455588394)

[Propósito de la prueba 3](#_Toc455588395)

[Tipos de tests 4](#_Toc455588396)

[Niveles de Prueba 5](#_Toc455588397)

[Prueba de unidad 5](#_Toc455588398)

[Pruebas de Integración 6](#_Toc455588399)

[Prueba de Sistema 7](#_Toc455588400)

[Estrategias de Prueba 8](#_Toc455588401)

[El Proceso de Prueba 9](#_Toc455588402)

[Planificación de la prueba. 9](#_Toc455588403)

[Bibliografía 12](#_Toc455588404)

# Introducción a la Prueba

La fase de evaluación pretende verificar que lo diseñado esté de acuerdo con la especificación requerida. En sí, el objetivo de las actividades de evaluación es solamente asegurar una calidad certificada para el producto.

Así las actividades de testeo normalmente se dividen en verificación y validación. La verificación chequea que el resultado coincide con la especificación. Sin embargo esto no garantiza la satisfacción del cliente. La validación chequea si el resultado realmente es el que se desea.

La validación se obtiene principalmente por medio de un análisis de requerimientos completo, incluyendo, la participación activa de los clientes. Una herramienta firme para la validación son los Caso de uso.

## La Prueba

La prueba es independiente del método de desarrollo utilizado. El enfoque Orientado a Objetos da nuevas posibilidades y también nuevos problemas. La actividad de prueba puede normalmente dividirse en:

Verificación: ¿Estamos construyendo el sistema CORRECTAMENTE?

Validación: ¿Estamos construyendo el sistema CORRECTO?

En este momento discutiremos la *verificación* porque la validación se resuelve con la determinación de requerimientos, por el uso de prototipos, etc.

## Propósito de la prueba

Se definirán en primer lugar algunos conceptos:

* Falla: Cuando un programa funciona mal.
* Falta: Existe en el código del programa. Puede provocar una falla.
* Error: Acción humano que resulta en software que contiene una falla.

La primera lección a aprender es que no se puede probar que el sistema no tenga falta, sin que esté libre de fallas.

El propósito de la prueba es encontrar fallas.

La prueba es un proceso destructivo, tener que indagar sobre lo que hicimos para detectar lo que hicimos mal. Es conocido que la corrección de una falla provoca fallas adicionales, en consecuencia, si una falla aparece debemos probar todo el software.

# Tipos de tests

El siguiente es un resumen de varios tipos de tests. Ninguno es independiente de los otros, cuando se realiza la prueba de un sistema, se usan en combinación.

**Tests de Operación**

Es el más común. El sistema es probado en operación normal.Mide la confiabilidad del sistema y se pueden obtener mediciones estadísticas.

Ejecutamos el sistema al *máximo,* todos los parámetros enfocan a **Tests de Escala** valores máximos, todos los equipos conectados, usados por **Completa** muchos usuarios ejecutando caso de usos simultáneamente.

**Tests de Perfomance o de Capacidad**

Cumple la función de determinar cómo se comporta el sistema

**Tests de Sobrecarga**. No se puede esperar que supere esta prueba, pero sí que no se venga abajo, que no ocurra una catástrofe. Cuántas veces se cayó el sistema es una medida interesante.

**Tests Negativos**

El sistema es sistemática e intencionalmente usado en formaincorrecta. Este maltrato debe ser planeado para probar casos especiales.

**Tests Ergonómicos** Son muy importantes si el sistema será usado por gente inexperta.Se prueban cosas como:

* Consistencia de la interfaz.
* Consistencia entre las interfaces de los distintos casos. Si los menús son lógicos y legibles.

Si se entienden los mensajes de falla.

**Test de Aceptación** Este test es ejecutado por la organización que solicita el sistema.El sistema es probado en un entorno real usualmente llamado *A/fa.* Cuando no hay un usuario que solicita el producto seusan las pruebas *Beta,* que son encargadas a clientes selectos antes de liberar la versión

# Niveles de Prueba

Los niveles de prueba se mencionan ahora en forma general, para luego profundizarlos.

* Prueba de Unidad: se prueban las clases, bloques, paquetes.
* Prueba de Integración: el objetivo es probar que las unidades trabajan correctamente juntas.
* Prueba de Sistema: se prueba el sistema completo. Requiere la colaboración de un usuario final y de casos de prueba típicos.

## Prueba de unidad

La prueba de unidad involucra: clases, bloques, paquetes de servicio. En sistemas tradicionales: procedimientos, subrutinas.

Las pruebas de unidad de sistemas orientados a objeto son más complejas; conceptos como la herencia, el polimorfismo, etc., hacen más compleja la prueba.

Los requerimientos para depuradores son mayores para sistemas Orientados a Objeto. Normalmente entornos como Smalltalk, C++, Simula, contienen soportes para o inspeccionar la estructura de los objetos durante la ejecución.

La prueba de unidad consiste de:

***Prueba de Especificación o Caja Negra***

Verifican el comportamiento de la interfaz de la unidad. *Lo que hace* sin importar *cómo.* Es importante ver no solamente si se produce una salida, sino verificar que lamisma sea correcta. También podemos probar la relación entrada/salida en diferente estados de la unidad, pero eso se hace con los casos de prueba basados en estados. Como las unidades sólo se comunican con interfaces definidas, las pruebas de especificación son bastante directas. Hemos definido cuáles operaciones soporta la unidad y qué comportamiento debería mostrar para cada operación.

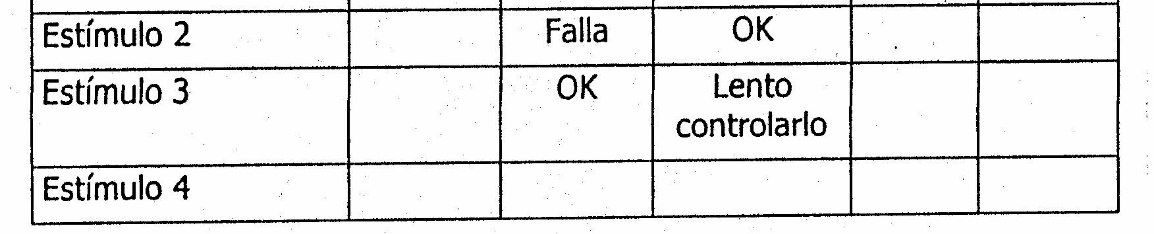
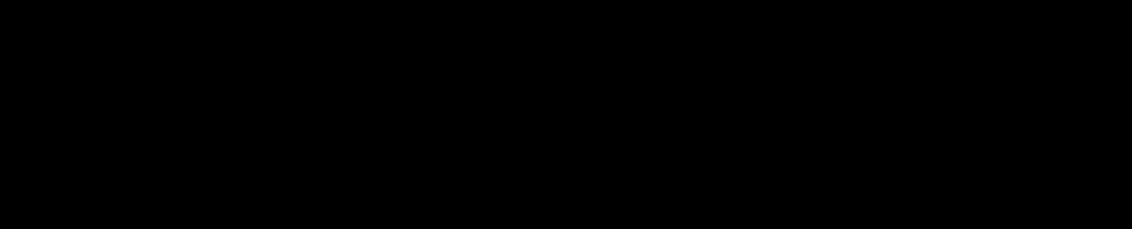
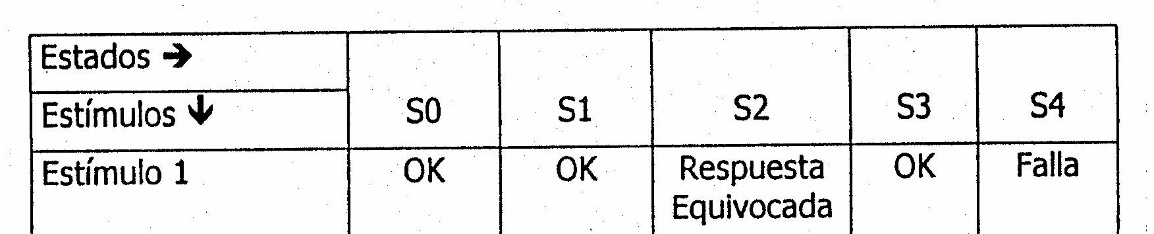
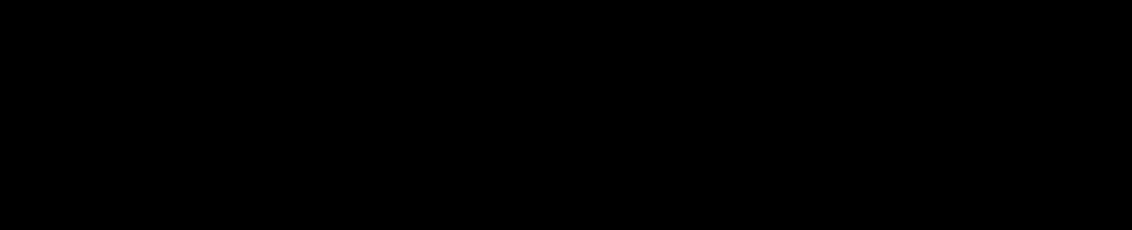
***Prueba Estructural o de Caja Blanca***

Se verifica si la estructura interna es la correcta. Todos los caminos posible planteados en el código deben ser contemplados y ejecutados (los llamados caminos de decisión a decisión). Dado que los casos de prueba estructurales y los basados en estado pueden modificar la estructura del código, es preferible hacer la prueba estructural al último. Es casi imposible recorres absolutamente todos los caminos posibles, considerando los parámetros y los valores de las variables. Los depuradores son de gran ayuda.

***Prueba Basada en estados***

Prueba la interacción entre las operaciones de una clase, monitoreando los cambios que tienen lugar en los atributos de los objetos, probar sólo una operación aislada no es suficiente para probar una unidad. Se deben probar también los atributos del objeto, puesto que si persisten entre invocaciones de diferentes operaciones, y posiblemente a través de interacciones entre las operaciones. Es importante basarse en los diagramas de transición de estados, así, al menos cada estado es visitado por lo menos una vez y cada transición es atravesada al menos una vez.

La matriz de estados es una buena herramienta para este tipo de pruebas. La combinación de estados y estímulos puede probarse con esta matriz.



Una de las ventajas de este tipo de matrices en que focaliza la atención del diseñador en la combinación estímulo /estado que puede ser descuidada durante el diseño. Es posible incluir todas las combinaciones de atributos del objeto (todos los posibles valores de variables) y todas las variantes de estímulos (distintos parámetros). En general algunas combinaciones específicas de atributos pueden ser más interesantes que otras.

Algunas operaciones, como las de lectura, que no afectan el estado, no deben ser consideradas.

Debe verificarse que todos los posibles estados pueden alcanzarse con alguna combinación de operaciones, de otra manera puede haber una falla en el diseño de la clase.

## Pruebas de Integración

Una vez que las unidades han sido certificadas en las pruebas de unidad, estas unidades deberían integrarse en unidades más grandes y finalmente al sistema. El propósito de las pruebas de integración es determinar si las distintas unidades que han sido desarrolladas trabajan apropiadamente, juntas.

Aquí se incluyen pruebas de paquetes de servicio, de caso de usos, subsistemas y el sistema completo. Consecuentemente no hay una sola prueba de integración en un desarrollo, por el contrario, se realizan varias a distintos niveles.

Estas pruebas son necesarias porque:

* Al combinar las unidades pueden aparecer nuevas fallas.
* La combinación aumenta exponencialmente el número de caminos posibles.

Por lo tanto hay fallas que no podrían detectarse de otra forma. Nuevamente aquí, los caso de usos se transforman en la herramienta que conduce la prueba de integración. Se puede comenzar la prueba de caso de usos tan pronto como la prueba de los bloques que los componen hayan sido certificados y aprobados.

Las bases para la especificación de estas pruebas vienen desde los diagramas de interacción, allí se ve claramente la interacción entre usuarios y el sistema y entre los objetos (bloques) del sistema.

Las pruebas de integración se hacen probando cada Caso de uso, uno a la vez desde dos puntos de vista:

* **Uno interno:** basado en los diagramas de interacción.
* **Uno externo:** basado en las descripciones del modelo de requerimientos.

## Prueba de Sistema

Una vez que se han probado todo los caso de usos por separado se probará el sistema completo. Algunos caso de usos son ejecutados en paralelo y el sistema es sometido a diferentes cargas.

Las pruebas de sistema pueden dividirse en los siguientes tests:

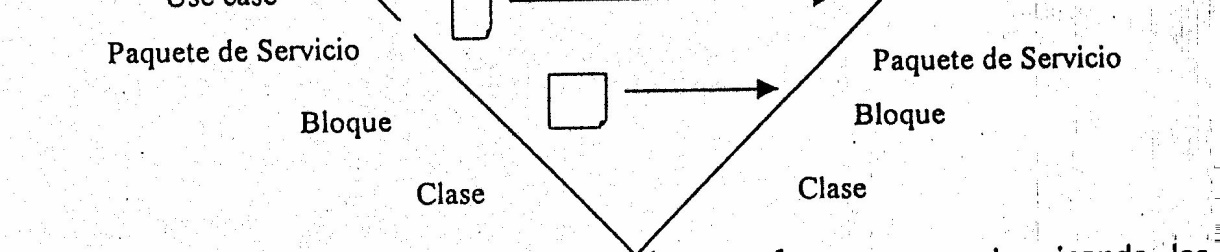
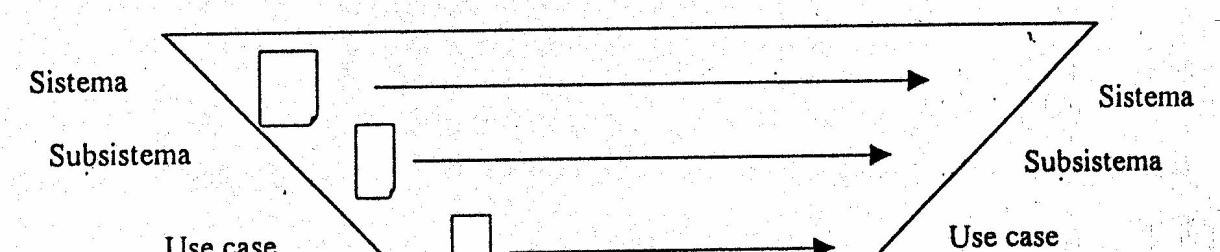
* Tests de operación
* Tests de escala completa
* Tests negativos
* Tests basados en especificación de requerimientos
* Tests de documentación del usuario

Cuando probamos el sistema, los caso de usos debería probarse en paralelo sincronizada y desincronizadamente. Se puede forzar el sistema corriendo varios caso de usos a la vez.

## Estrategias de Prueba

Las estrategias de prueba pueden realizarse de varias maneras, pero lo más común es hacerlas en orden inverso al que se realiza el diseño y la implementación.

La verificación se hace en varios niveles, veamos la siguiente figura:



Sin embargo podemos ir realizando pruebas conforme vamos terminando los diseños, es decir la implantación y la prueba se realizan en forma intercalada e incremental.

Puesto que podemos desarrollar de varias maneras: Top-down, Botton-Up, por caso de uso, podemos hacer estos incrementos usando las mismas estrategias.

Si se asume un enfoque Top- Down en el diseño, esto significa que primero desarrollamos las interfaces entre subsistemas, las que retornan valores que son controlados y posteriormente se reemplaza con el código real, esto permite probar el flujo completo en niveles superiores antes y luego ir a niveles inferiores.

Las pruebas también pueden hacerse botton-up o por caso de usos básicos. El enfoque botton-up es preferible en los niveles más bajos, cuando la primera unidad está certificada y los clientes directos pueden ser certificados. Entonces el siguiente nivel de clientes puede ser certificado y así sucesivamente.

Esta técnica minimiza las necesidades de implementar clases piloto sólo para prueba, puesto que las unidades certificadas trabajan como servidores. Esto da una convergencia suave entre pruebas de unidad y de integración. Sin embargo, la detección de fallas en las unidades servidoras podría forzar a comenzar todo el proceso nuevamente.

## Ejecución de las Pruebas

Cuando ejecutamos las pruebas usamos la especificación de pruebas y los reportes de prueba preparados. La estrategia es probar lo que más se pueda en paralelo, aunque sea difícil.

Las pruebas se realizan en forma manual o automatizada, según se haya especificado.

Las especificaciones indican el resultado esperado. Si alguna de las pruebas falla se registra, se interrumpe la ejecución, y el defecto es analizado y corregido si se puede. Luego la subprueba se ejecuta nuevamente.

Al finalizar la prueba se analizan los resultados. Si está aprobado o no. Este análisis resulta en reportes de prueba. Los reportes contienen en forma resumida el resultado individual de cada subtest y uno final, los recursos gastados y si el test está aprobado o no.

Si se descubrieron cuellos de botella también deben registrarse y mostrarse.

Recuerda que puedes ir completando el “Plan de Prueba” que esta en nuestro sitio, y que a través de un tutor, te va orientando en los contenidos a desarrollar.



## Caso de prueba Listar Cliente

|  |
| --- |
| Nombre de caso de prueba: Listar clientes |
| Nombre de UC que deriva: Registrar Clientes |
| Restricciones: Tener previamente cargados los datos de clients en la base de datos |
| Datos de entrada: Apellido/N°/DNI/CUIL/CUIT |
| Pasos: 1 – Ingresar a la interfaz “Listar Clientes”.  2-Seleccionar la búsqueda por un parámetro  3- Ingresar datos identificatorios.  4- Seleccionar cliente, y se muestran en la ventana Registrar clientes, los datos del mismo. |
| Datos de salida: Los resultados obtenidos son correctos. Se complete la interfaz Registrar cliente. |

# Tipos de pruebas realizadas.

Prueba de unidades: Se realiza éste tipo de pruebas para comprobar que cada parte funcione correctamente.

Menú Principal: Se realiza un logeo en el Sistema, y se puede interactuar con las diferentes ventanas del programa, haciendo un test de las interfaces que se ejecuten de manera correcta.

Registrar Clientes, Proveedores, Distribuidores: Se ingresa en el Sistema, luego se debe cargar los datos y se guardan seleccionando la opcion Guardar .Se verificó que los datos que se deben guarder, se hayan registrado de la manera correcto, como por ejemplo DNI , CUIL/CUIT, nombre, apellido, etc.

Pruebas de Sistema: En este tipo de pruebas, se realiza un chequeao de la infraestructura y arquitectura tanto de hardware y software sobre las cuales las pruebas , sean similares a las del entorno de implementación

Pruebas de caja negra: En éste tipo de prueba, se limita unicamente a la salida obtenida del Sistema, y no en el Sistema interior y sus mecanismos.

Pruebas de caja blanca: Analizamos y ademas controlamos que aquellas líneas del código de nuestro programa, estén codificadas de la manera correcta, y además realicen la tarea que debe hacer.

Prueba de aceptación: En ésta etapa, se require al usuario final (que es el que va a utilizer el programa) , que realice diferentes pruebas con el programa para que pruebe las diferentes opciones y logre una aceptación o no del software entregado, para poder realizer mejoras y otros aspectos generales.

# Bibliografía

* Módulo: Sistemas III

Autor: Ing. Savi, Cecilia.

* Módulo: Sistemas IV

Autor: Ing. Savi, Cecilia.