

Resumen del Primer Parcial

Validación y verificación: Proceso de comprobar que un sistema de software cumple con sus especificaciones y su propósito previsto (también llamado control de **calidad de software**)

Objetivos principales

- Descubrir defectos en el sistema
- Asegurar que el software respeta su especificación
- Determinar si satisface las necesidades de sus usuarios

La calidad no puede inyectarse al final, esta depende de tareas realizadas durante todo el proceso.

Detectar errores en forma temprana ahorra esfuerzos, tiempo, recursos.

Calidad en Software

Uno de los objetivos principales en el desarrollo de software es obtener productos de alta calidad

Generalmente, se mide en atributos de calidad...	
Confiabilidad	Usabilidad
Corrección	Robustez
Facilidad de Mantenimiento	Seguridad (en datos, acceso, ...)
Reusabilidad	Funcionalidad
Verificabilidad + Claridad	Interoperabilidad
Etc..	



Verificación

- **Dinámica:** trata con ejecutar y observar el comportamiento de un producto
- **Estática:** trata con el análisis de una representación estática del sistema para descubrir problemas

Verificación estática y dinámica

Técnicas de Verificación Estática

- Inspecciones, Revisiones
- Análisis de reglas sintácticas sobre código
- Análisis Data Flow sobre código
- Model checking
- Prueba de Teoremas
- Entre otras...

Técnicas de Verificación Dinámica

- **Testing**
- Run-Time Monitoring. (pérdida de memoria, performance)
- Run-Time Verification
- Entre otras...

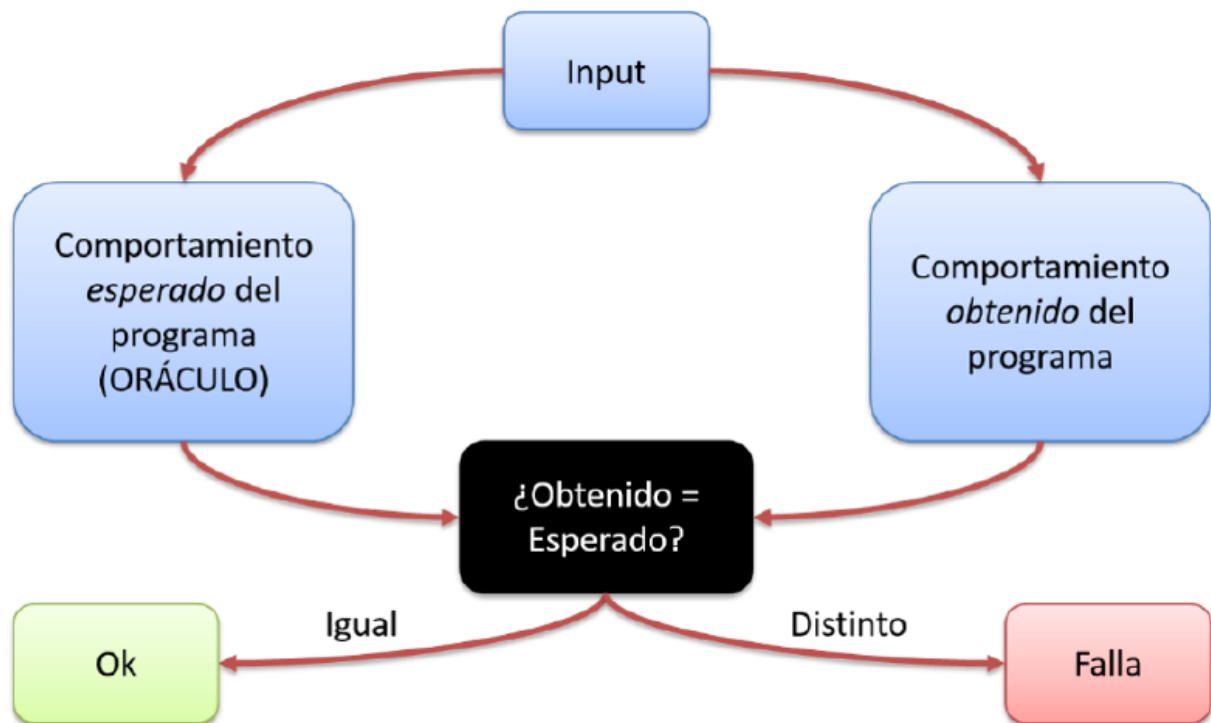
Testing

- Verifica que un producto satisface los requerimientos (especificación)
- Identifica diferencias entre el comportamiento real y el comportamiento esperado

Niveles de Testing

- Test de Sistema
Comprende todo el sistema. Por lo general constituye el test de aceptación.
- Test de Integración
Test orientado a verificar que las partes de un sistema que funcionan bien aisladamente, también lo hacen en conjunto
Testeamos la interacción, la comunicación entre partes
- Test de Unidad
Se realiza sobre una unidad de código pequeña, claramente definida. (Este es nuestro caso, el de HUnit)

¿Cómo se hace testing?



- **Falla**
Diferencia entre los resultados esperados y reales
Manifestación del defecto
- **Defecto**
Desperfecto en algún componente del sistema (en el texto del programa, una especificación, un diseño, etc), que origina una o más fallas.
Lleva a cero, una o más fallas
- **Error (Bug)**
Equivocación humana
Lleva a uno o más defectos, que están presentes en un producto de software

Test Input, Test Case y Test Suite

- **Test Input**
Es una asignación concreta de valores a los parámetros de entrada para ejecutar el programa bajo test.
- **Test Case**
Caso de Test (o caso de prueba). Es un programa que ejecuta el programa bajo test usando un dato de test, y chequea (automáticamente) si se cumple la condición de aceptación sobre la salida del programa bajo test.
- **Test Suite**
Es un conjunto de casos de Test (o de conjunto de casos de prueba).

¿Qué caso de test elegir?

1. No hay un algoritmo que proponga casos tales que encuentren todos los errores en cualquier programa.
2. Ninguna técnica puede ser efectiva para detectar todos los errores en un programa arbitrario
3. En ese contexto, veremos dos tipos de criterios para seleccionar datos de test:
 - **Test de Caja Negra:** Los casos de test se generan analizando la especificación sin considerar la implementación. Los datos de test se derivan a partir de la descripción del programa sin conocer su implementación.
 - **Test de Caja Blanca:** Los casos de test se generan analizando la implementación para determinar los casos de test. Los datos de test se derivan a partir de la estructura interna del programa.

Limitaciones del testing:

Una de las mayores dificultades es encontrar un conjunto de tests adecuado:

Suficientemente grande para abarcar el dominio y maximizar la probabilidad de encontrar errores.

Suficientemente pequeño para poder ejecutar el proceso con cada elemento del conjunto y minimizar el costo del testing.