ArgentinaPrograma
YoPrograMo

Pruebas

por Leonardo Blautzik, Federico Gasior y Lucas Videla

Julio / Diciembre 2021

Ministerio de
Desarrollo Productivo
Argentina

Necesidad

- ¿Cómo sabemos que el software **hace lo que tiene que hacer**?

Necesidad

- ¿Cómo sabemos que el software **hace lo que tiene que hacer**?
- ¿Cómo sabemos que el software **no hace lo que no tiene que hacer**?

Concepto de prueba

La prueba es la **comparación** de un resultado **obtenido** con un resultado **esperado**.

Concepto de prueba

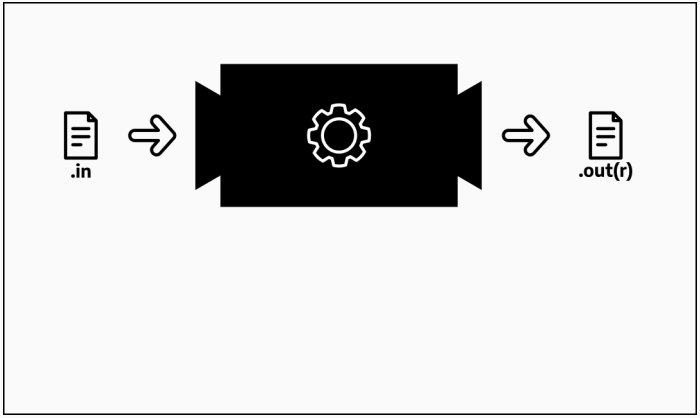

Figure 1: El software como una máquina

Concepto de prueba

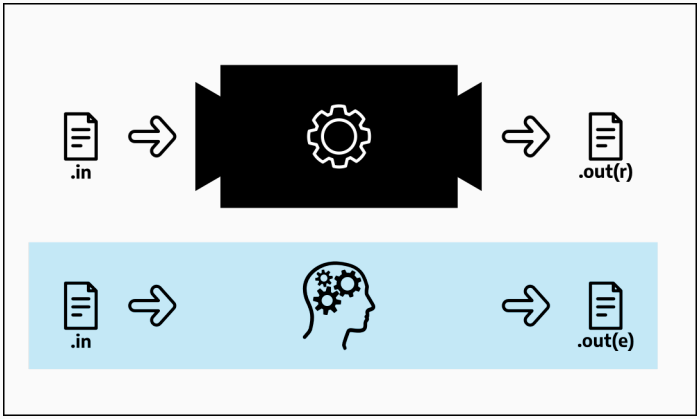

Figure 2: Preparación de la prueba

Concepto de prueba

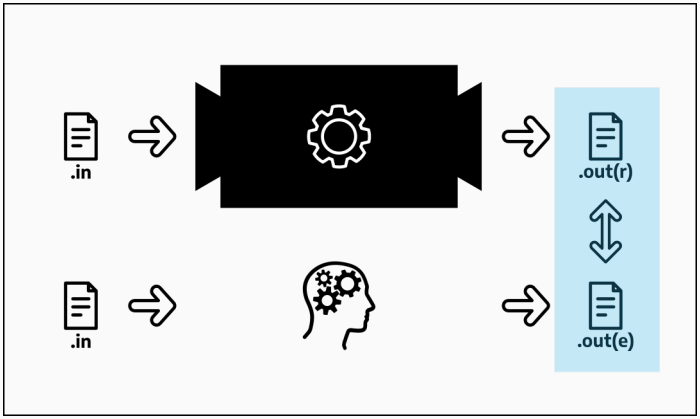

Figure 3: Verificación

¿Para qué sirven las pruebas?

- Sería **deseable** que las pruebas garanticen la ausencia de errores

¿Para qué sirven las pruebas?

- Sería **deseable** que las pruebas garanticen la ausencia de errores
- Lamentablemente, sólo puede garantizarse la ausencia de los errores **expresamente comprobados**

Entonces... ¿por qué probamos?

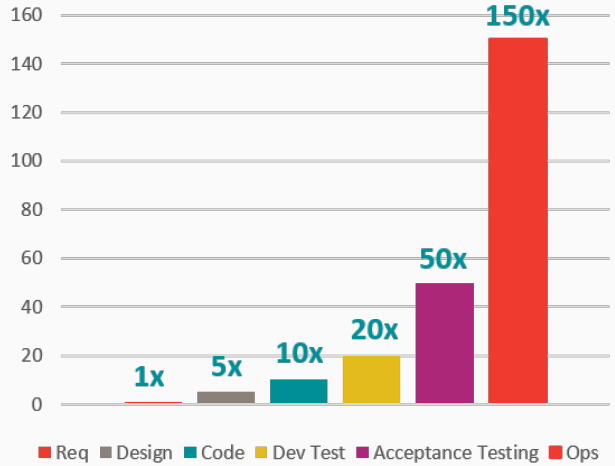

Figure 4: Costos de arreglar errores en distintas etapas del desarrollo de software

Calidad de software

- ¿Qué es la calidad?
- ¿Cómo se mide?
- ¿Quién la percibe?

Requisitos funcionales vs. no funcionales

- Un requisito funcional describe las características que necesita el usuario final, y que **el sistema debe ofrecer**.
- Un requisito no funcional es una **restricción de calidad del sistema**. No declaran una necesidad del usuario, aunque pueden resultarle de utilidad.

Requisitos funcionales vs. no funcionales, diferencia

Requisitos Funcionales	Requisitos No Funcionales
Definen al sistema	Define un atributo de calidad que el sistema debe ofrecer
Especifica qué debe hacer el sistema	Especifica cómo debe cumplirlo
Lo especifica el usuario	Lo especifican los roles técnicos
Son obligatorios	No son obligatorios
Verifican funcionalidad	Verifican desempeño
Se utilizan pruebas de unidad, de integración, de sistema	Se utilizan prueba de desempeño, de estrés, de usabilidad, de seguridad, etc
Suelen ser fáciles de definir por su naturaleza concreta	Son difíciles de definir por su naturaleza abstracta

¿Qué es lo que probamos nosotros?

- Ambos >.<

¿Qué es lo que probamos nosotros?

- Ambos >.<
- pero en diferente medida ^.^

Pruebas unitarias (¡ahora sí!)

Una **prueba unitaria** es un fragmento de código que invoca una unidad de trabajo y **verifica un resultado final específico** de esa unidad de trabajo. Si los supuestos sobre el resultado final resultan ser equivocados, la prueba ha fallado.

El alcance de una prueba unitaria puede abarcar **tan poco como un método o tanto como múltiples clases**.

– Osherove, R. [1]

Propiedades de un buen test unitario

La sigla FIRST nos ayuda a recordar que “Lo PRIMERO son las buenos pruebas” (funciona mejor en inglés...)

- [F]ast - Rápidos
- [I]solated - Independientes
- [R]epeatable - Repetibles
- [S]elf-validating - Auto-validados
- [T]imely - Oportunos

Cómo se hace

Una prueba tiene cuatro partes, 4 “A”, en inglés:

- **Arrange - Preparar**
  Se genera el escenario para la ejecución de la prueba.

Cómo se hace

Una prueba tiene cuatro partes, 4 “A”, en inglés:

- **Arrange - Preparar**
  Se genera el escenario para la ejecución de la prueba.
- **Act - Ejecutar**
  Se ejecuta la acción principal.

Cómo se hace

Una prueba tiene cuatro partes, 4 “A”, en inglés:

- **Arrange - Preparar**
  Se genera el escenario para la ejecución de la prueba.
- **Act - Ejecutar**
  Se ejecuta la acción principal.
- **Assert - Verificar**
  Se verifican las consecuencias de la ejecución.

Cómo se hace

Una prueba tiene cuatro partes, 4 “A”, en inglés:

- **Arrange - Preparar**
  Se genera el escenario para la ejecución de la prueba.
- **Act - Ejecutar**
  Se ejecuta la acción principal.
- **Assert - Verificar**
  Se verifican las consecuencias de la ejecución.
- **Annihilate - Restaurar (!)**
  Se restaura el estado inicial, anterior a la prueba.

Pruebas con JUnit

Veremos en forma práctica cómo hacer prueba con JUnit.

¡Vamos al editor de código!

Repaso

Hemos visto cómo utilizar:

- `@Test`, para definir pruebas
- `@Before`, para ejecutar antes de cada prueba
- `@After`, para ejecutar luego de cada prueba
- `@BeforeAll`, para ejecutar antes de todas las pruebas, por única vez
- `@AfterAll`, para ejecutar luego de todas las pruebas, por única vez
- `Assert`, para verificar según algún criterio

Material adicional

- [1] Osherove, R., Feathers, M., & Martin, R. C. (2014). The art of unit testing: With examples in C# (2. ed). Manning.
- [2] Beck, K. (2019). Test Desiderata. Go placidly amid the noise and haste.... Visitado 16 de julio, 2021, desde https://medium.com/@kentbeck_7670/test-desiderata-94150638a4b3
- [3] JUnit Cookbook. (2018). https://junit.org/junit4/cookbook.html

¡Muchas Gracias!

continuará...

Ministerio de
Desarrollo Productivo
Argentina