# Unidad 6: Estrategias de Prueba de Software

## ¿Qué es probar software?

- **Testing**: Ejecutar un programa con el objetivo de **encontrar errores**

  Ej.: probar que `2 + 2 == 4`

- **Depuración (Debugging)**: Encontrar y corregir errores después de detectar un fallo

  Ej.: si `2 + 2 == 5`, buscar el error en el código

---

## Verificación vs Validación

| Concepto | ¿Qué verifica? | Ejemplo |
|---|---|---|
| **Verificación** | ¿Se construyó bien el software? | ¿El campo email valida bien? |
| **Validación** | ¿Se construyó el **software correcto**? | ¿Cumple lo que pidió el cliente? |

## ¿Cuándo termina la prueba?

- **Nunca se prueba "todo"**
- Se termina cuando **se acaba el tiempo o el presupuesto**
- Siempre se transfiere algo de riesgo al usuario final

---

# Estrategia de prueba efectiva (Pressman)

1. Especificar **requisitos medibles**
2. Definir **objetivos claros de prueba**
3. Desarrollar un **perfil de usuario realista**
4. Planificar pruebas **rápidas y automáticas**
5. Diseñar el software para que **se pueda probar**
6. Usar **revisiones técnicas** antes de probar

7. Mejorar continuamente el proceso de pruebas

# TIPOS DE PRUEBAS

## 1. Prueba Unitaria

- Se testea una **función o método en aislamiento**
- Detecta errores tempranos
- Actúa como **documentación viva**

## 2. Prueba de Integración

- Verifica que **los módulos funcionen bien juntos**
- Detecta errores en **interfaz entre componentes**

## 3. Prueba de Regresión

- Asegura que **los cambios no rompan lo que ya funcionaba**
- Se repiten pruebas anteriores
- Idealmente **automatizadas**

## 4. Prueba de Humo

- Testeo básico diario en proyectos grandes
- Detecta errores graves rápido
- No es exhaustiva, pero sí **crítica**

## 5. En Programación Orientada a Objetos

- Unidad de prueba = **clase**
- Se testean **métodos y estado interno**
- Se debe considerar: herencia, composición, polimorfismo

# DOBLES DE PRUEBA

| Tipo | ¿Para qué sirve? | Ejemplo |
|---|---|---|
| Dummy | Relleno para completar parámetros | Se pasa pero no se usa |
| Stub | Devuelve una respuesta fija | Simula una respuesta sin lógica real |
| Fake | Funciona pero es una implementación simple | Ej.: base de datos en memoria |
| Mock | Verifica si se llamó un método | Comprueba interacciones |
| Spy | Observa internamente qué hizo el objeto real | Conteo de llamadas, parámetros usados |

# Pruebas de Validación

- Pruebas E2E (end-to-end): verifican el comportamiento del sistema como lo ve el **usuario final**
- Requiere un plan de validación
- Resultados esperados:
  - Funcionalidad aprobada
  - Lista de defectos si hay desviaciones

---

# Pruebas Alfa y Beta

| Tipo | ¿Dónde se realiza? | ¿Quién participa? |
|---|---|---|
| **Alfa** | En el sitio del desarrollador | Usuarios reales, pero en ambiente controlado |
| **Beta** | En el sitio del cliente | Usuarios finales reales, sin desarrollador |

# Otras pruebas (Pressman)

- **Recuperación**: se fuerza un fallo y se analiza si el sistema se recupera
- **Seguridad**: se intenta vulnerar el sistema
- **Esfuerzo (stress)**: se sobrecarga con datos o transacciones
- **Performance**: se mide la eficiencia (tiempo, memoria)

- **Despliegue**: instalación en entornos distintos

---

## Depuración: preguntas clave

1. ¿El error se repite en otras partes?
2. ¿La corrección genera nuevos errores?
3. ¿Qué se podría haber hecho para evitar este error?

---

## Resumen Final

| Fase | Objetivo principal |
|---|---|
| Unit Test | Verificar métodos individuales |
| Integración | Verificar que las partes se comuniquen |
| Regresión | Asegurar que no se rompa lo viejo |
| Validación | Asegurar que cumple con el pedido |
| Alfa/Beta | Verificar en uso real (con o sin devs) |