escuela técnica superior
de ingeniería informática

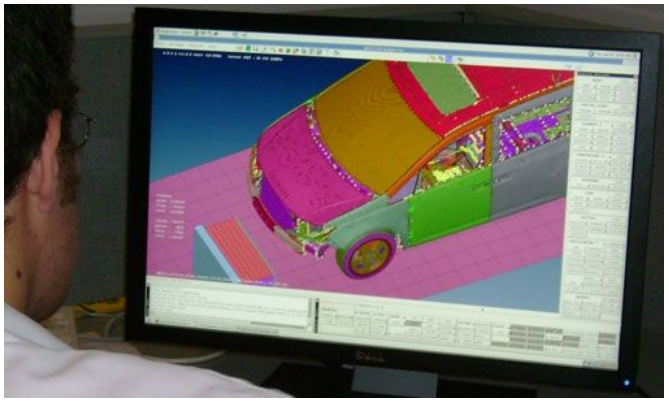
Pruebas en Django

Departamento de
Lenguajes y Sistemas Informáticos

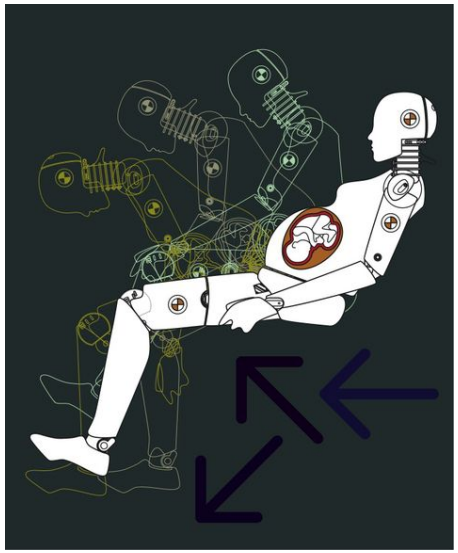
UNIVERSIDAD D SEVILLA

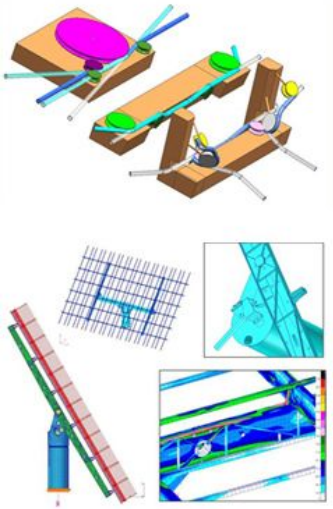
Ejemplo de otro dominio

Probar: 1. tr. Hacer examen y experimento de las cualidades de alguien o algo.



Ejecución de la prueba





Diseño de la prueba

Diseño de pruebas

¿**Cómo** probar que un sistema software está haciendo algo mal?
¿**Dónde** está el problema?

¿Interesa en Investigación?
¿Y en el trabajo?

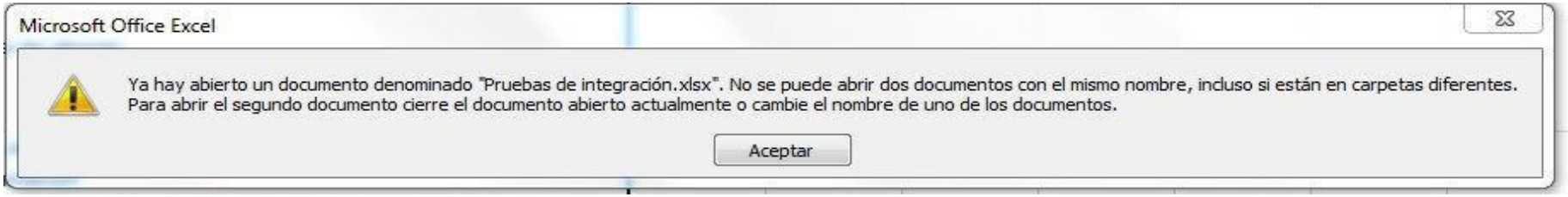
Google académico

"software testing"

	1990	2000	2009
En el título	53	115	319
En alguna parte	290	1140	3860

	2018
En el título:	335
En alguna parte:	7110

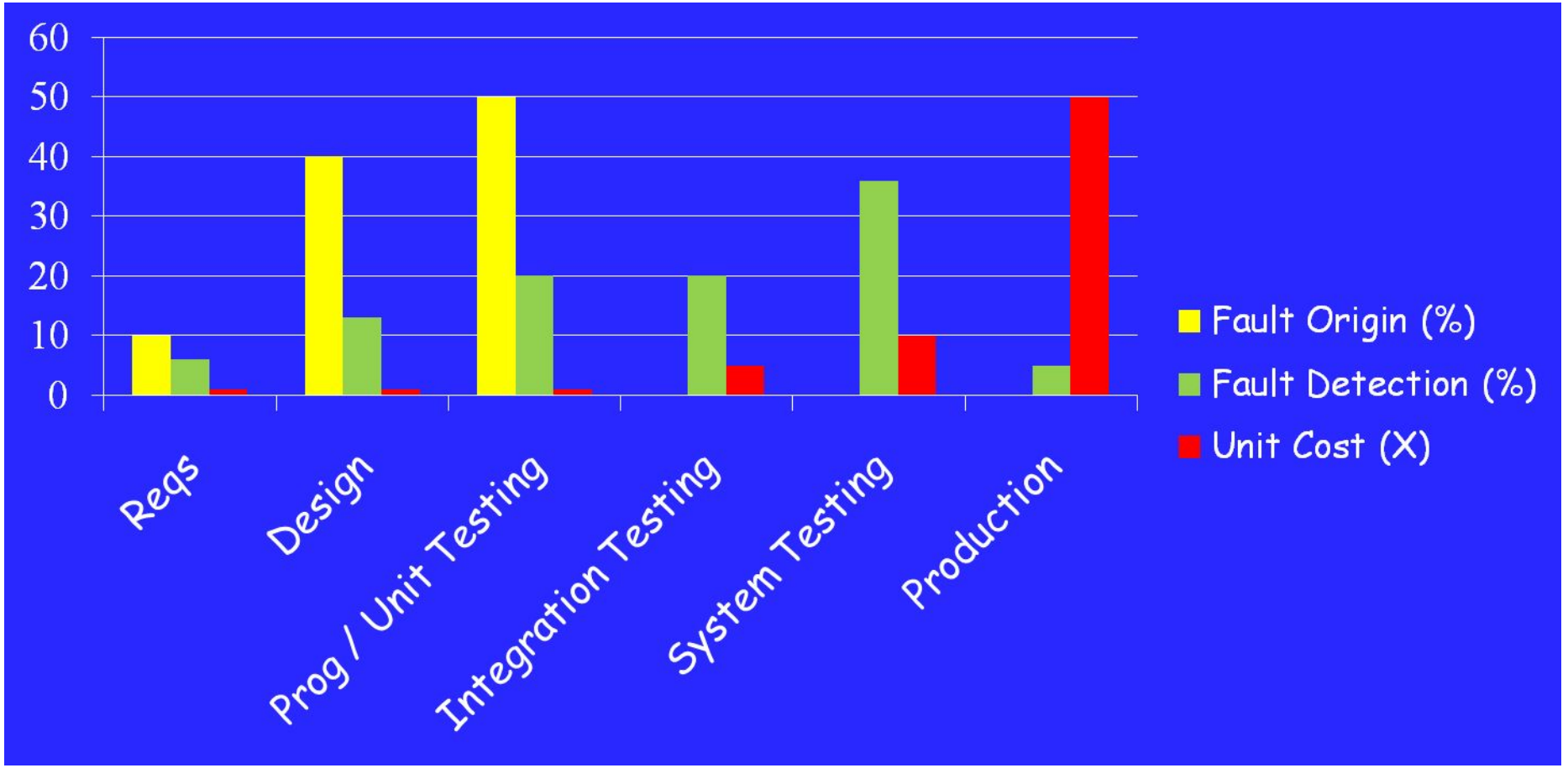
Ejemplos de fallos



"There are two ways to write error-free programs; only the third one works."

—Alan J. Perlis

Etapas dónde se producen errores



Software Engineering Institute; Carnegie Mellon University; Handbook CMU/SEI-96-HB-002

Pruebas funcionales vs pruebas no funcionales

- Probar que con el carrito vacío no se puede hacer un pedido
- Probar que si se introduce una tarjeta de crédito errónea no se permite realizar un pedido.
- Probar que la interfaz gráfica es conforme a una norma de accesibilidad
- Probar que se pueden conectar 100 usuarios concurrentemente

Proceso general de prueba

Implementación de pruebas en django

Un Ejemplo:

```
from django.test import TestCase

class SimpleTest(TestCase):
    def test_basic_addition(self):
        """
        Tests that 1 + 1 always equals 2.
        """
        self.assertEqual(1 + 1, 2)
```

Framework de testing unitario

(**unnittest** → Inspirado en *JUnit*)

Conceptos

- **test fixture** - Preparación necesaria para realizar las pruebas
- **test case** - Caso concreto e individual que se quiere probar
- **test suite** - Conjunto de casos de prueba.
- **test runner** - Componente que ejecuta los tests.

Lugar de implementación y ejecución

- La aplicación crea un fichero **tests.py** por defecto.
- Si necesitamos **más complejidad** →
  Crear nuevos scripts de formato **test*.py**

Una vez escritos, se ejecutan desde la terminal:

```
#Corre todos los tests disponibles
$./manage.py test

#Corre los tests dentro de “voting”
$./manage.py test voting
```

Test de modelos

En Django, los tests referentes a la base de datos no usan la BBDD de **producción**.

(No es necesario declararla en settings.py)

Test de modelos

```
def test_poll(self):
        """
        Test that Poll is correctly created and saved in DB
        """
        ca = Ca.objects.create(id=1312, name="Andalucia")
        census = Census.objects.create(id=1222, title="Jose",
        postalCode=11510, ca=ca)
        poll = Poll.objects.create(id=1319, title="Prueba",
        description="Votación de prueba",
        startDate="2017-01-13",
        endDate="2018-01-10",census=census, participantes=0,
        votos=0)
self.assertEqual(poll.id, 1319)
```

Integración continua de pruebas

¿Cómo se automatiza la ejecución de las pruebas?

.gitignore	Añadido .gitignore
.project	Fix: Settings.py y nombre del proyecto
.pydevproject	Fix: Settings.py y nombre del proyecto
.travis.yml	Cambio para el despliegue automatico
Dockerfile	Cambio para el despliegue automatico
ISSUE_TEMPLATE.md	Creación de la plantilla de Issues
README.md	Create README.md
manage.py	Fix: Settings.py y nombre del proyecto
requirements.txt	Arreglada gramática en requisitos

Integración continua de pruebas

.travis.yml

```
language: python

sudo: required

python:
    - "2.7.10"

install: "pip install -r requirements.txt"

script: py.test ./principal/tests.py

services:
  - mysql
  - docker
```

etc...

Integración continua de pruebas

¿Cómo se automatiza la ejecución de las pruebas?

Integración continua de pruebas

```
480 principal/tests.py .F.
481
482 ================================== FAILURES ===================================
483 _________________________ SimpleTest.test_busca_votaciones __________________________
484
485 self = <principal.tests.SimpleTest testMethod=test_busca_votaciones>
486
487     def test_busca_votaciones(self):
488         b,votaciones = busca_votaciones()
489 >       self.assertEqual(b,True)
490 E       AssertionError: False != True
491
492 principal/tests.py:30: AssertionError
493 ====================== 1 failed, 2 passed in 0.21 seconds ======================
494
495
496 The command "py.test ./principal/tests.py" exited with 1.
497
498 Done. Your build exited with 1.
```

Diseño de casos de prueba

Un plan de pruebas exhaustivo es impracticable

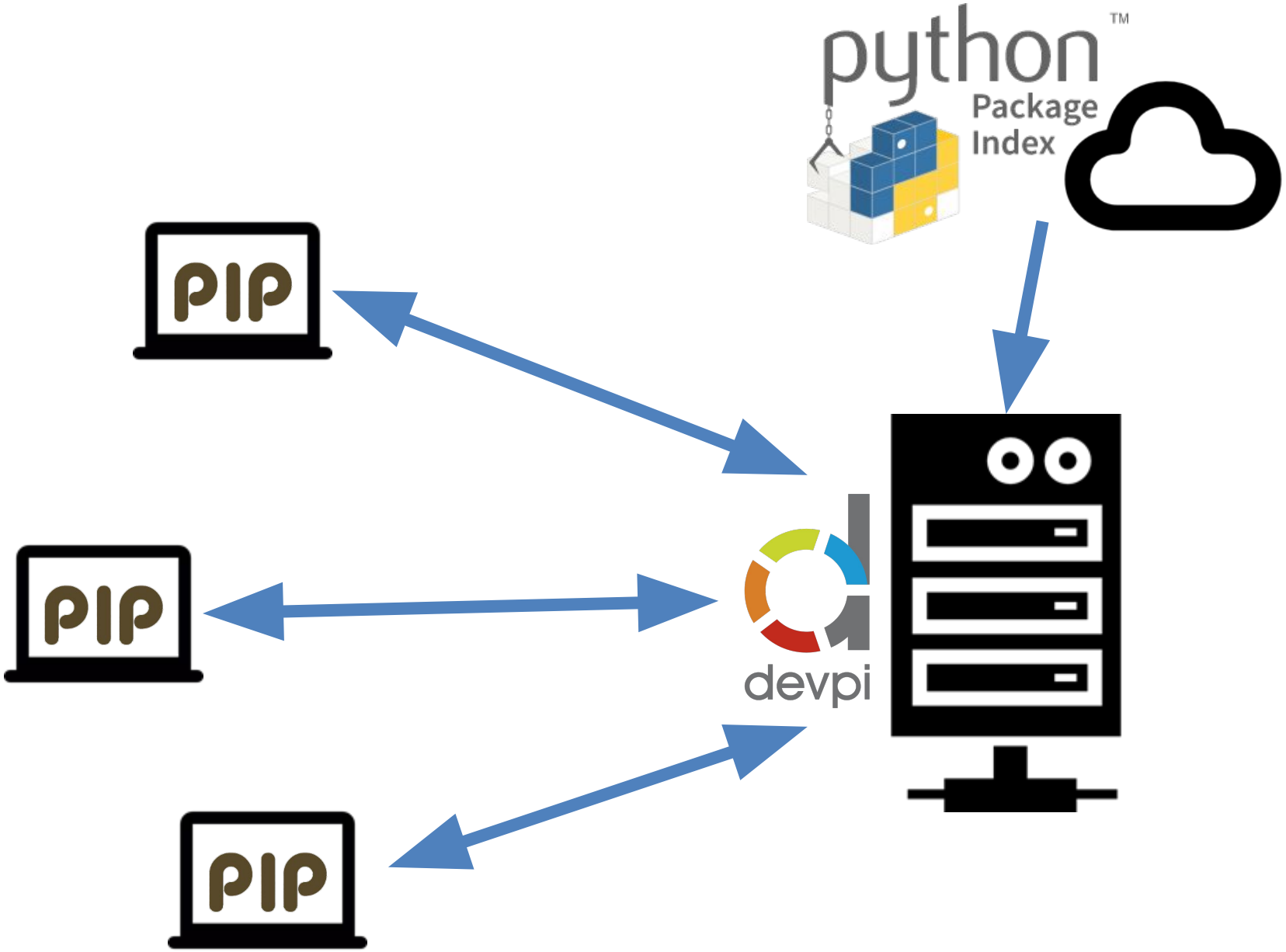
Las posibles entradas de un programa pueden (y suelen) ser infinitas, e.g. compilador

El objetivo de un buen banco de pruebas (*test set, test suite*): pocas entradas, muchos fallos.

¿Cómo seleccionar las entradas? Usando un **criterio de cobertura** (*coverage criteria*)

Criterio de cobertura: **conjunto de reglas** que imponen una serie de requisitos a un banco de pruebas (*test suite/ test set*).

Bonus: Gestión de la construcción en Python



Consejos finales

- La **documentación** es tu amiga.
- El **trabajo en grupo** es esencial.
- Un buen uso de **Git** es **obligatorio en cualquier ámbito.**
- Mantened las **pruebas simples pero útiles.**