

# Test automatici con PHPUnit

Corso Backend System Integrator  
Modulo **Programmazione PHP**

Docente: Dott. Enrico Zimuel

in collaborazione con:



REGIONE  
PIEMONTE

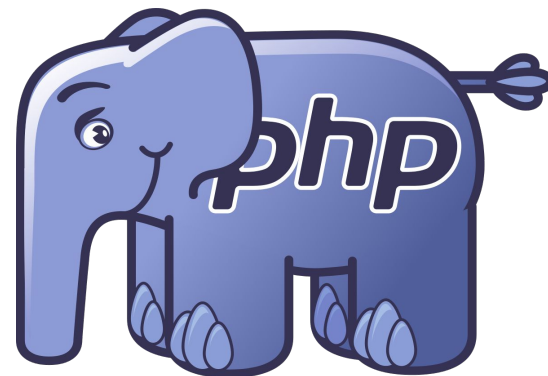
per una crescita intelligente,  
sostenibile ed inclusiva

[www.regione.piemonte.it/europa2020](http://www.regione.piemonte.it/europa2020)

INIZIATIVA CO-FINANZIATA CON FSE

# Programma

- Test automatici
- PHPUnit
- Configurazione
- setUp() e tearDown()
- Data provider
- Mock di oggetti
- Code coverage



Relay 2145  
Relay 337

# Harvard Mark II

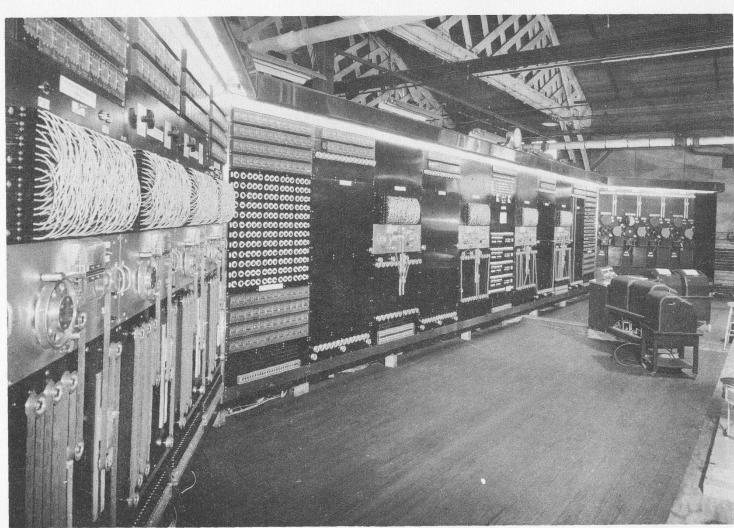


Plate I Main Control Board and Wings

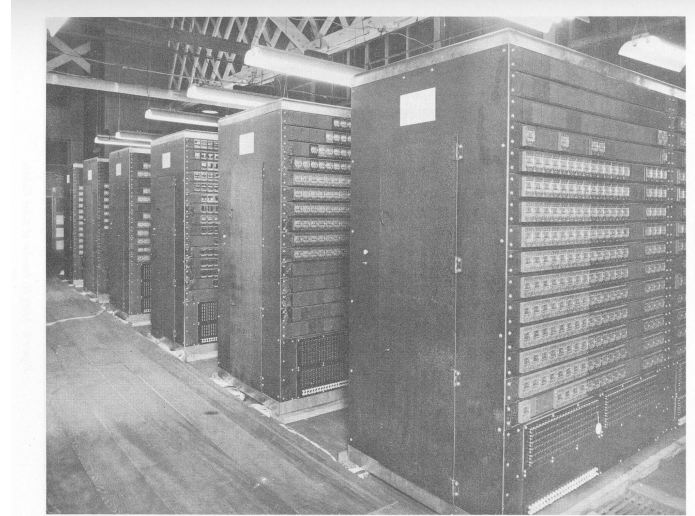


Figure 5 Mark II: Relay Cabinets

Harvard Mark II (1947) peso di 23 t per uno spazio occupato di 370 m<sup>2</sup>.  
Velocità di 0.125 sec (8 Hz) per una somma e 0.750 sec per un prodotto.

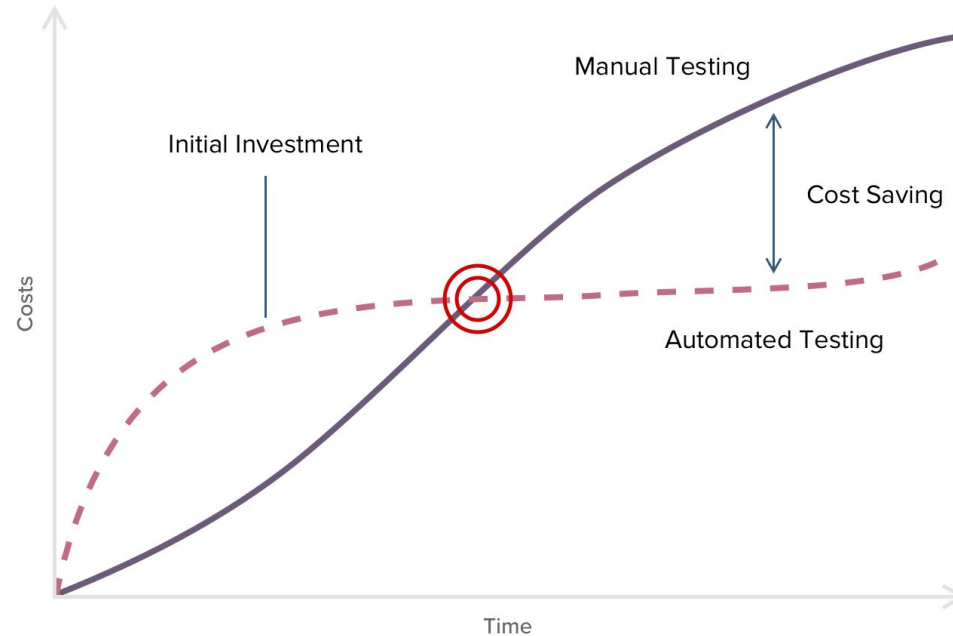
# Test automatici

- I test automatici sono una collezione di asserzioni, ossia una serie di istruzioni che testano la corretta esecuzione di una porzione di codice, utilizzando dei dati in ingresso prestabiliti e verificando che i risultati siano quelli attesi

# Importanza dei test

- Riducono il numero di bug
- Riducono i tempi di sviluppo (a lungo termine)
- Aiutano a chiarire le specifiche del progetto
- Migliorano la confidenza nel codice, e.s. durante il refactoring
- Garantiscono notti tranquille agli sviluppatori

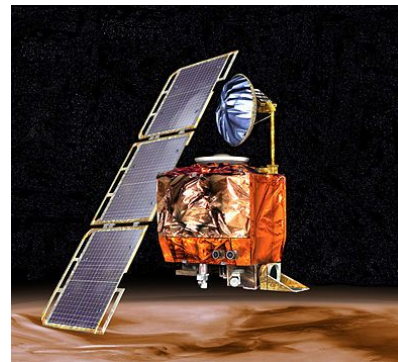
# Manual vs. automated testing





# Esempi famosi di software failure

- [NASA's Mars Climate Orbiter](#), \$128 milioni andati in fumo per un errore di conversione metrico
- [Pentium FDIV bug](#), un bug del processore Pentium nella divisione in virgola mobile. Danni per \$475 milioni
- [Ariane 5 Flight 501](#), \$370 milioni andati in fumo per l'esplosione del satellite pochi secondi dopo il decollo a causa di un overflow di 32-bit in 16-bit





# PHPUnit

# PHPUnit

- Libreria PHP per sviluppare test automatici
- Progetto open source nato nel 2004 per opera di [Sebastian Bergman](#)
- Standard de facto per i test automatici in PHP
- Sito ufficiale del progetto: [phpunit.de](http://phpunit.de)

# Configurazione

- PHPUnit può essere configurato tramite un file XML (**phpunit.xml**):

```
<phpunit>
  <testsuites>
    <testsuite name="Test automatici">
      <directory>./test</directory>
    </testsuite>
  </testsuites>
  <filter>
    <whitelist processuncoveredfilesfromwhitelist="true">
      <directory suffix=".php">./src</directory>
    </whitelist>
  </filter>
</phpunit>
```

# Struttura di un test

- I test in PHPUnit sono raggruppati in classi, denominate con il **suffisso** Test (es. FilterTest)
- Ogni classe deve estendere **PHPUnit\Framework\TestCase**
- Un test è una funzione della classe, denominato con il **prefisso** test (es. testEmailsValid)

# Esecuzione di PHPUnit

- PHPUnit è eseguito dal terminale, tramite il comando:

```
vendor/bin/phpunit
```

```
→ app git:(master) x ./vendor/bin/phpunit
```

```
PHPUnit 9.5.1 by Sebastian Bergmann and contributors.
```

```
..... 65 / 85 ( 76%)  
..... 85 / 85 (100%)
```

```
Time: 00:13.550, Memory: 38.00 MB
```

```
OK (85 tests, 5485 assertions)
```

# Esempio: classe Filter

```
namespace App;  
  
class Filter  
{  
    public function isEmail(string $email): bool  
    {  
        // @todo da implementare  
    }  
}
```

# Esempio: test della classe Filter

```
namespace App\Test;
use PHPUnit\Framework\TestCase;
use App\Filter;

class FilterTest extends TestCase
{
    public function testValidEmail() {
        $filter = new Filter();
        $this->assertTrue($filter->isEmail('foo@bar.com'));
    }
    public function testInvalidEmail() {
        $filter = new Filter();
        $this->assertFalse($filter->isEmail('foo'));
    }
}
```



# Assertzioni

- PHPUnit offre numerose **asserzioni**:
  - **assertTrue()**: verifica che l'elemento sia true
  - **assertFalse()**: verifica che l'elemento sia false
  - **assertEmpty()**: verifica che l'elemento sia vuoto
  - **assertEquals()**: verifica che due elementi siano uguali
  - **assertGreaterThan()**: verifica che un elemento sia maggiore di
  - **assertContains()**: verifica che un elemento sia contenuto in un array
  - **assertInstanceOf()**: verifica che un elemento sia istanza di una classe
  - [Qui](#) la lista completa

# Esercizio

Implementare il metodo `App\Filter::isEmail()` e verificare che i test automatici siano tutti **positivi**

**Nota:** i sorgenti dell'esercizio si trovano [qui](#)

# setUp()

**setUp()** è un metodo per inizializzare un test:

```
class FilterTest extends TestCase
{
    public function setUp(): void
    {
        $this->filter = new Filter();
    }
    public function testValidEmail()
    {
        $this->assertTrue($this->filter->isEmail('foo@bar.com'));
    }
}
```

# tearDown()

**tearDown()** è un metodo per resettare lo stato di un test:

```
class FilterTest extends TestCase
{
    public function setUp(): void    {
        file_put_contents('/tmp/foo', 'Test');
    }
    public function tearDown(): void {
        unlink('/tmp/foo');
    }
    public function testFoo()    {
        // utilizzo il file temporaneo
    }
}
```

# Ciclo di vita di un test

**setUp()** e **tearDown()** vengono richiamati prima e dopo ogni test

```
class FooTest extends TestCase
{
    public function setUp() {}
    public function tearDown() {}
    public function testUno() {}
    public function testDue() {}
}
// setUp(), testUno(), tearDown()
// setUp(), testDue(), tearDown()
```

# Data Provider

```
class DataTest extends TestCase
{
    /**
     * @dataProvider getDataInput
     */
    public function testAdd($a, $b, $expected) {
        $this->assertEquals($expected, $a + $b);
    }

    public function getDataInput() {
        return [
            [0, 0, 0],
            [0, 1, 1]
        ];
    }
}
```

# Esercizio

Modificare la classe **FilterTest** utilizzando un **data provider** per verificare che i dati "foo@bar.com" e "foo" siano validi o meno



# Mock di oggetti

- Il **mocking** di un oggetto è una tecnica che consente di simulare il comportamento di un oggetto per finalità di testing
- Viene utilizzato per gestire le **dipendenze esplicite** delle classi
- In un test si evitano di utilizzare dipendenze con istanze reali di oggetti

# Esempio

```
class Foo {  
    public function __construct(PDO $pdo)  
    {  
        $this->pdo = $pdo;  
    }  
    public function getAll(): array  
    {  
        $sth = $this->pdo->prepare('SELECT * FROM table');  
        $sth->execute();  
        return $sth->fetchAll(PDO::FETCH_ASSOC);  
    }  
}
```

## Esempio (2)

```
class FooTest extends TestCase
{
    public function setUp()
    {
        $this->pdo = $this->createMock(PDO::class);
        $this->sth = $this->createMock(PDOStatement::class);
        $this->pdo->method('prepare')->willReturn($this->sth);
        $this->sth->method('execute')->willReturn(true);
        $this->foo = new Foo($this->pdo);
    }
    public function testGetAllReturnArray()
    {
        $this->sth->method('fetchAll')->willReturn([[ 'id' => 1 ]]);
        $this->assertEquals([[ 'id' => 1 ]], $this->foo->getAll());
    }
}
```

# Code coverage

- La copertura del codice (**code coverage**) è una misura che indica la percentuale, espressa in righe di codice, delle istruzioni eseguite durante un test
- Esempio una copertura dell'80% indica che i test automatici non eseguono il 20% del codice
- PHPUnit offre la possibilità di calcolare la code coverage tramite il comando:










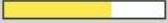


```
vendor/bin/phpunit --coverage-text
```

# Xdebug

Per eseguire la funzionalità di code coverage è necessario installare [Xdebug](#)

Current directory: `/Users/jsambells/We-Create/Projects/Development/wc-432-476`

Legend: Low: 0% to 35% Medium: 35% to 70% High: 70% to 100%

	Coverage								
	Lines			Functions / Methods			Classes		
Total		24.66%	5254 / 21308		15.69%	358 / 2282		20.96%	48 / 229
<u>application</u>		76.74%	66 / 86		20.00%	1 / 5		0.00%	0 / 1
<u>library</u>		24.39%	5169 / 21194		15.50%	352 / 2271		21.15%	48 / 227
<u>tests</u>		67.86%	19 / 28		83.33%	5 / 6		0.00%	0 / 1

Generated by PHPUnit 3.4.11 and Xdebug 2.0.5 using PHP 5.3.2 at Thu Apr 8 6:38:08 EDT 2010.

# Esercizio

Installare o verificare la presenza di **Xdebug** ed eseguire il code coverage dell'esercizio precedente

# Esercizio (da consegnare)

- Scrivere una classe per gestire un sistema di autenticazione (email e password)
- Questa classe deve implementare la seguente interfaccia:

```
interface Authenticate
{
    public function verify(string $email, string $password) : bool
}
```

- Le email e le password devono essere memorizzati in un database **MySQL**
- Le password devono essere memorizzate con la funzione **password\_hash()** del PHP
- Completare l'esercizio scrivendo i test automatici con l'ausilio del **PHPUnit**



# Grazie dell'attenzione!

Per informazioni:

[enrico.zimuel@its-ictpiemonte.it](mailto:enrico.zimuel@its-ictpiemonte.it)