

# Software Testing

## Unit Testing e Continuous Integration

Luca Morandini

# Software Testing

- Introduzione al testing del software
- Unit Testing
  - Unit test con Java (*JUnit*)
- Continuous Integration
  - Automazione di unit test (*GitHub Actions*)

# Software Testing

- **Introduzione al testing del software**
- Unit Testing
  - Unit test con Java (*JUnit*)
- Continuous Integration
  - Automazione di unit test (*GitHub Actions*)

**“Il testing del software può essere usato per dimostrare la presenza di bug, ma mai per dimostrare la loro assenza”**

**Edsger W. Dijkstra**

Se non è possibile garantire l'assenza di errori all'interno di un programma...

**Che senso ha effettuare la verifica del software?**

# Software Testing

## Motivazione

- Importante **ridurre al minimo il rischio** di introdurre bug quando vengono sviluppate nuove funzionalità
- Necessaria un'attenta e **continua** verifica
  - Durante l'intero processo di sviluppo
  - Non solo a prodotto finito

## Motivazione:

cercare di identificare il prima possibile eventuali errori

# Software Testing

## Obiettivo

- Il testing è un **riscontro parziale** della qualità perché il software viene provato solo per alcuni dati in input
- Importante identificare dati di test che **massimizzino la probabilità** di scoprire errori durante l'esecuzione

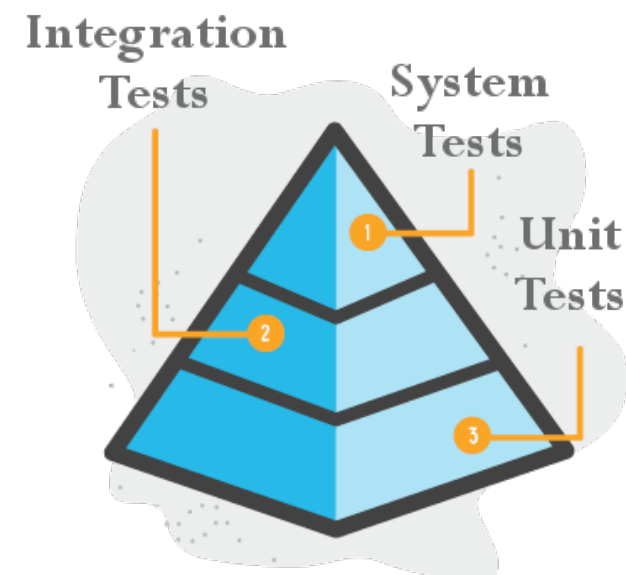
## Obiettivo:

trovare controesempi per dimostrare che  
il codice contiene errori

# Software Testing

## Livelli di test

- Esistono diversi livelli di test che vengono svolti in **diverse fasi** del processo di sviluppo ed hanno lo scopo di verificare vari aspetti di un software:
  - **Unit testing:** ogni modulo viene verificato e testato individualmente in modo indipendente dalle altre parti del programma
  - **Integration testing:** i moduli vengono integrati e si effettuano delle verifiche sulla loro interazione
  - **System testing:** il sistema completo viene validato per controllare che funzioni correttamente





# Software Testing

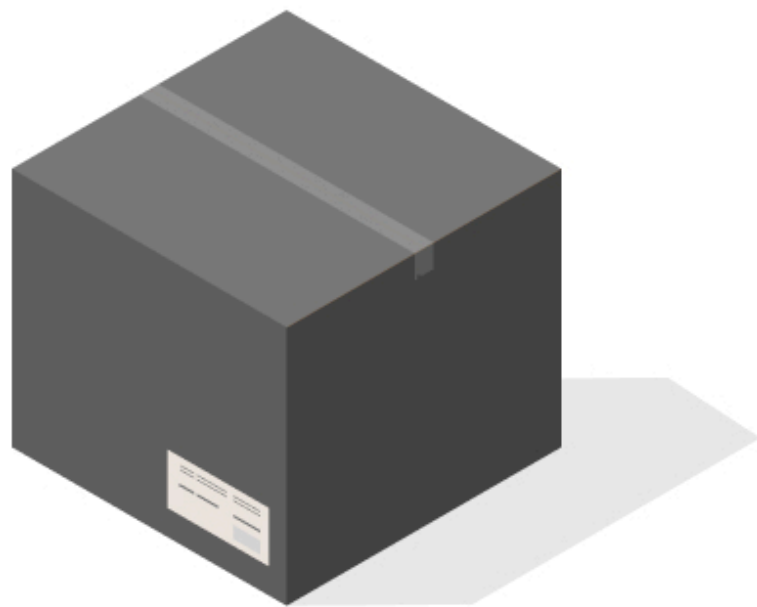
- Introduzione al testing del software
- **Unit Testing**
  - Unit test con Java (*JUnit*)
- Continuous Integration
  - Automazione di unit test (*GitHub Actions*)

# Unit Testing

- Lo scopo è validare che ogni unità di codice funzioni come previsto
- Unit testing ha un grande impatto sulla qualità del codice:
  - Appena un'unità viene completata, si creano alcuni unit test (*test suite*) per verificare il comportamento del codice con **input corretti ed incorretti**
  - Aiuta a **rilevare difetti in anticipo** che potrebbero diventare difficili da correggere in successive fasi di test
  - **Semplifica il debugging** perché se un test fallisce solo le ultime modifiche devono essere controllate

**Come definire i casi di test?**

# Black box vs White box



## Black box

Testing funzionale

*"Non conosciamo niente"*

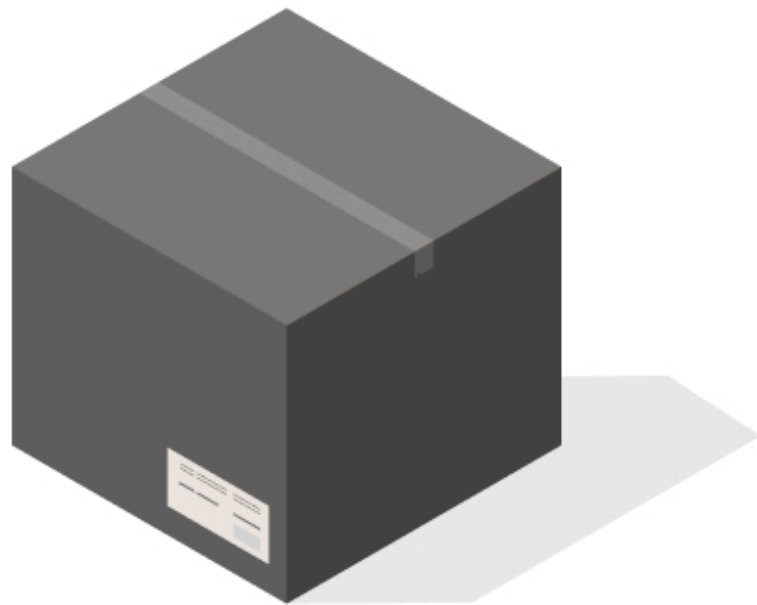


## White box

Testing strutturale

*"Conosciamo tutto"*

# Black box testing

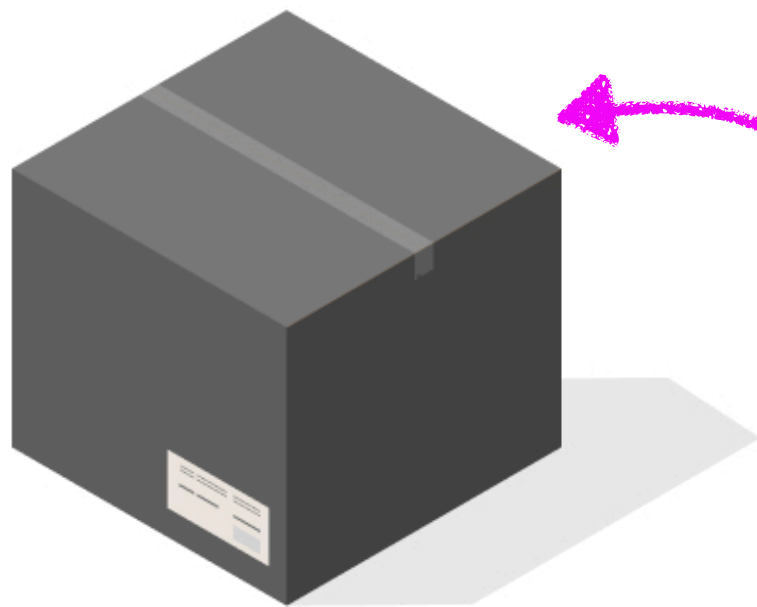


## Black box

*"Non conosciamo niente"*

- **Obiettivo:** verificare che un componente svolga la funzione richiesta
- **Casi di test:** definiti in base a quello che il componente deve fare (*la specifica*)
- Non è necessario conoscere l'implementazione
- **Boundary values:** se l'input può stare in un intervallo, testare gli estremi dell'intervallo:
  - *stringa*: vuota o di 1 carattere
  - *indici di array*: valori estremi

# Black box testing



```
int cumulativeSum(int[] values, int n) {  
    ...  
    ...  
    ...  
}
```

```
int[] values = new int[] {1, 2, 3};  
int n = 2;  
int sum = cumulativeSum(values, n);  
  
// in teoria: sum = 3
```

**Black box**

*"Non conosciamo niente"*

# White box testing

- **Obiettivo:** assicurarsi che un componente sia in grado di gestire tutti i possibili casi
- **Casi di test:** definiti in base alla struttura del componente (*l'implementazione*)
- Permette di avere la certezza di testare **tutte le parti** del codice
  - Selezionare dati che fanno percorrere **ogni sequenza** del codice (*anche parti raramente eseguite*)



**White box**

*"Conosciamo tutto"*

# White box testing

```
int cumulativeSum(int[] values, int n) {  
    int sum = 0;  
  
    for (int i = 0; i < n; i++) {  
        sum += values[i];  
    }  
  
    return sum;  
}
```

```
int[] values = new int[] {1, 2, 3};  
int n = 2;  
int sum = cumulativeSum(values, n);
```

```
// cosa succede se: n > 3?  
// spoiler: 💣 BOOM!
```



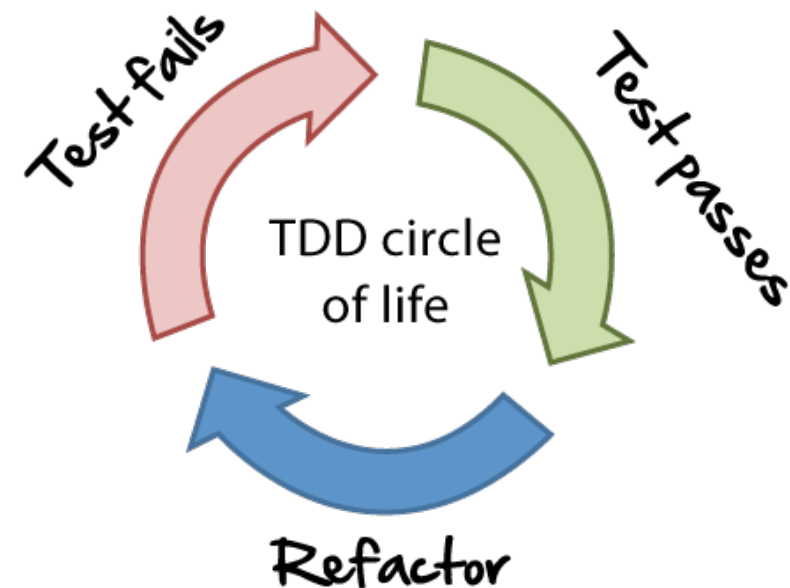
**White box**

*"Conosciamo tutto"*

# Test Driven Development

- Con la tecnica del TDD, lo sviluppo del software è **guidato dai test**
  - Scrittura dei **test prima dell'implementazione**
- I test non devono essere complessi ed ognuno esegue un solo compito
  - Caratteristica principale degli **unit tests**
- I vantaggi sono la **riduzione di bug** e una più **alta qualità** del software

Test → Codice → Design






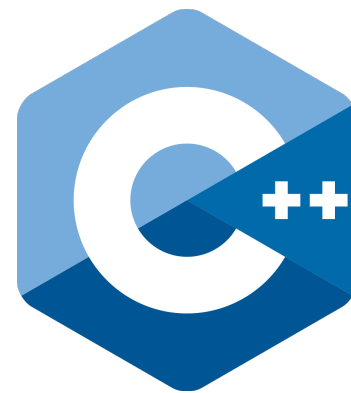
# Software Testing

- Introduzione al testing del software
- Unit Testing
  - **Unit test con Java (*JUnit*)**
- Continuous Integration
  - Automazione di unit test (*GitHub Actions*)

# Framework

## Unit testing

- **Java:** JUnit 
- **Python:** unittest
- **C++:** googletest
- **JavaScript:** Mocha



# JUnit 4

## Test Suite

- **Test Suite:** insieme di test che verificano la correttezza di una **classe** Java

```
public class DecimalEngineTest {  
    @BeforeClass  
    public static void setup() {  
        // eseguito una volta prima di tutti i test  
        // codice di inizializzazione per l'intera classe  
    }  
  
    @Before  
    public void init() {  
        // eseguito prima di ogni test  
        // codice comune di inizializzazione per ogni test case  
    }  
  
    @Test  
    public void ingest_InvalidNumber_ShouldDiscardValue() {  
        // metodo di test (test case)  
        // un metodo per ogni caso da verificare  
    }  
}
```

← Convenzione per i nomi delle test suite

# JUnit 4

## Test Case

- **Test Case:** test per verificare un **particolare caso** di un metodo
- In generale vengono definiti più test case per ogni metodo

```
@Test  
public void nomeMetodo_ValoriInput_RisultatoAtteso() { ... }
```

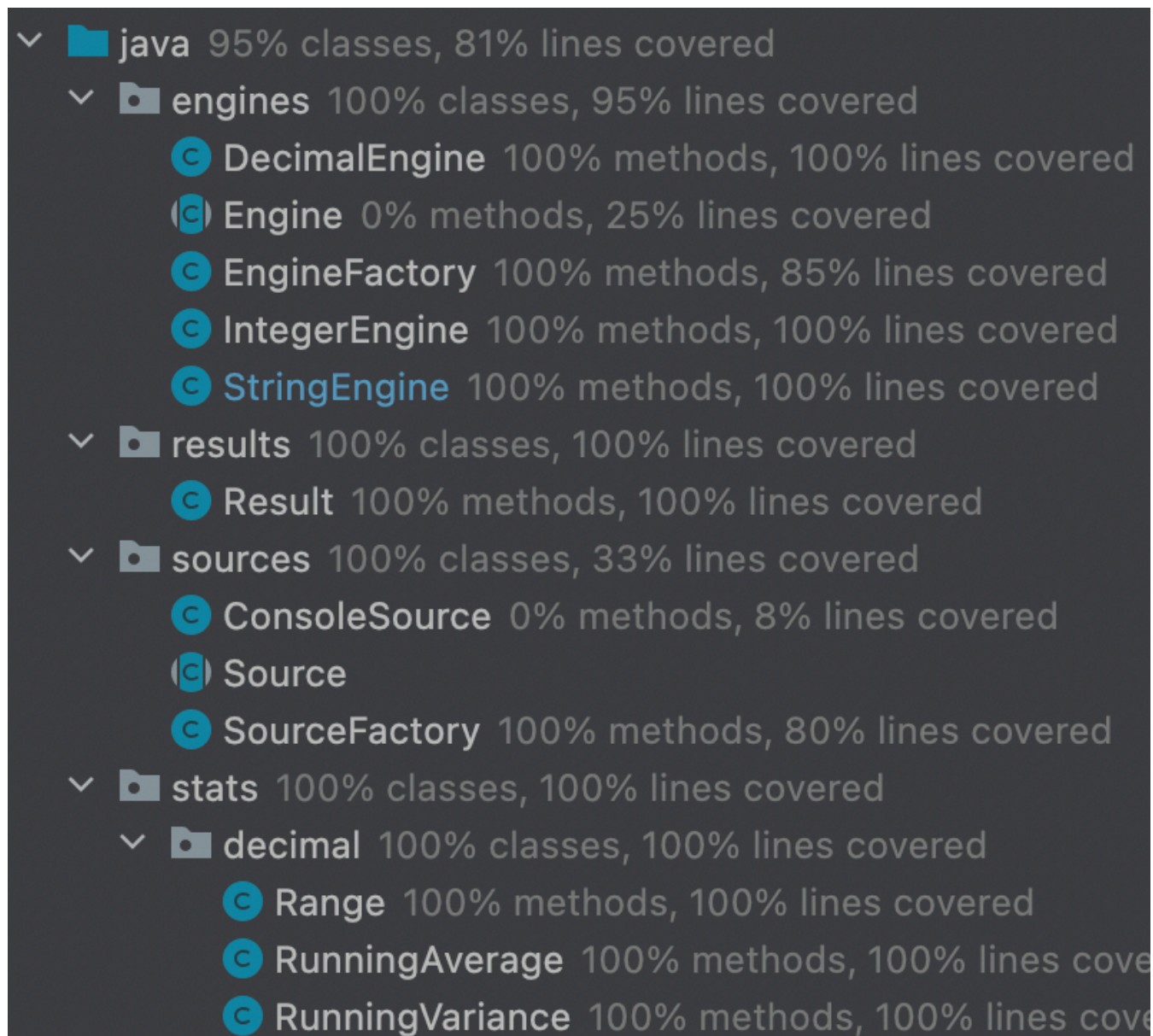
Convenzione per i nomi dei test case

```
@Test  
public void cumulativeSum_ThreeElems_ShouldSumTwoElems() {  
    // preparazione test  
    int[] values = new int[] {1, 2, 3};  
    int n = 2;  
  
    // esecuzione test  
    int sum = Stats.cumulativeSum(values, n);  
  
    // verifica risultato  
    assertEquals(3, sum);  
}
```

← Verifica che il risultato del metodo sia uguale al valore atteso dal test case

# JUnit 4

## Code coverage



- Una metrica utilizzata per misurare la qualità del testing è il **code coverage**
- Percentuale di righe di codice che sono state eseguite (*coperte*) dalla test suite
- L'obiettivo dello unit testing è ottenere una coverage di oltre **80/90%**

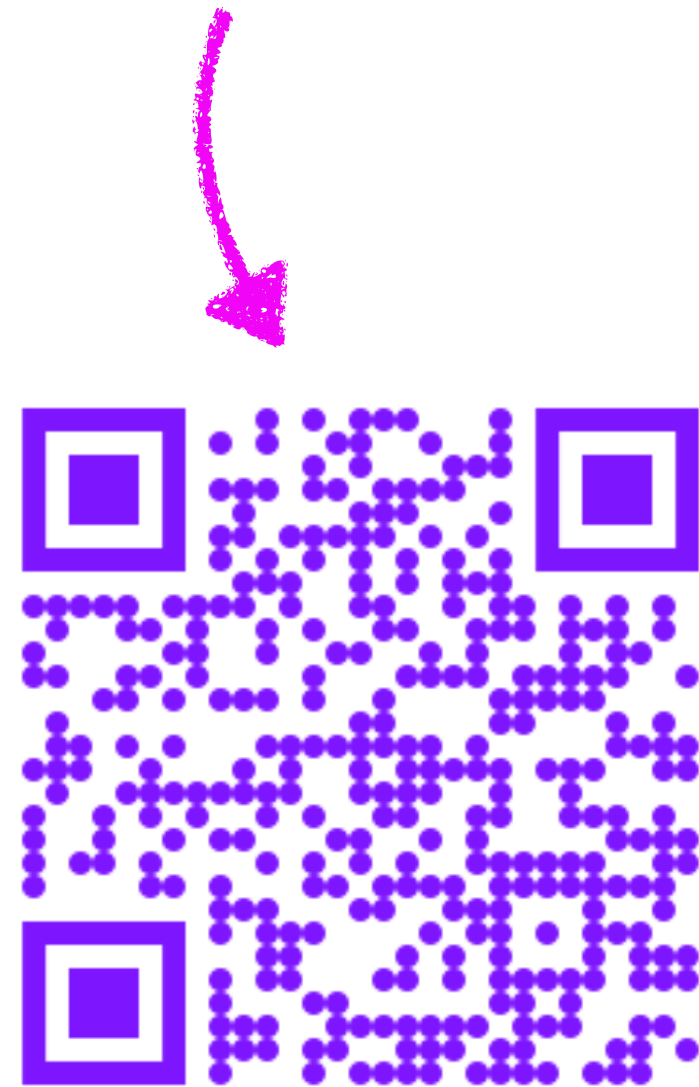
```
> starting simple stats
> select stats engine: [string|decimal|integer]
decimal
> select data source: [console]
console
> loading data
-10.4
6.3
2.8
-1.5
4.2

> exporting stats
results:
  engine: decimal
  stats:
    - name: average
      value: 0.280
    - name: variance
      value: 35.038
    - name: range
      value: 16.700
```

IDE utilizzato: IntelliJ IDEA

# simple-stats

[github.com/lucamora/simple-stats](https://github.com/lucamora/simple-stats)

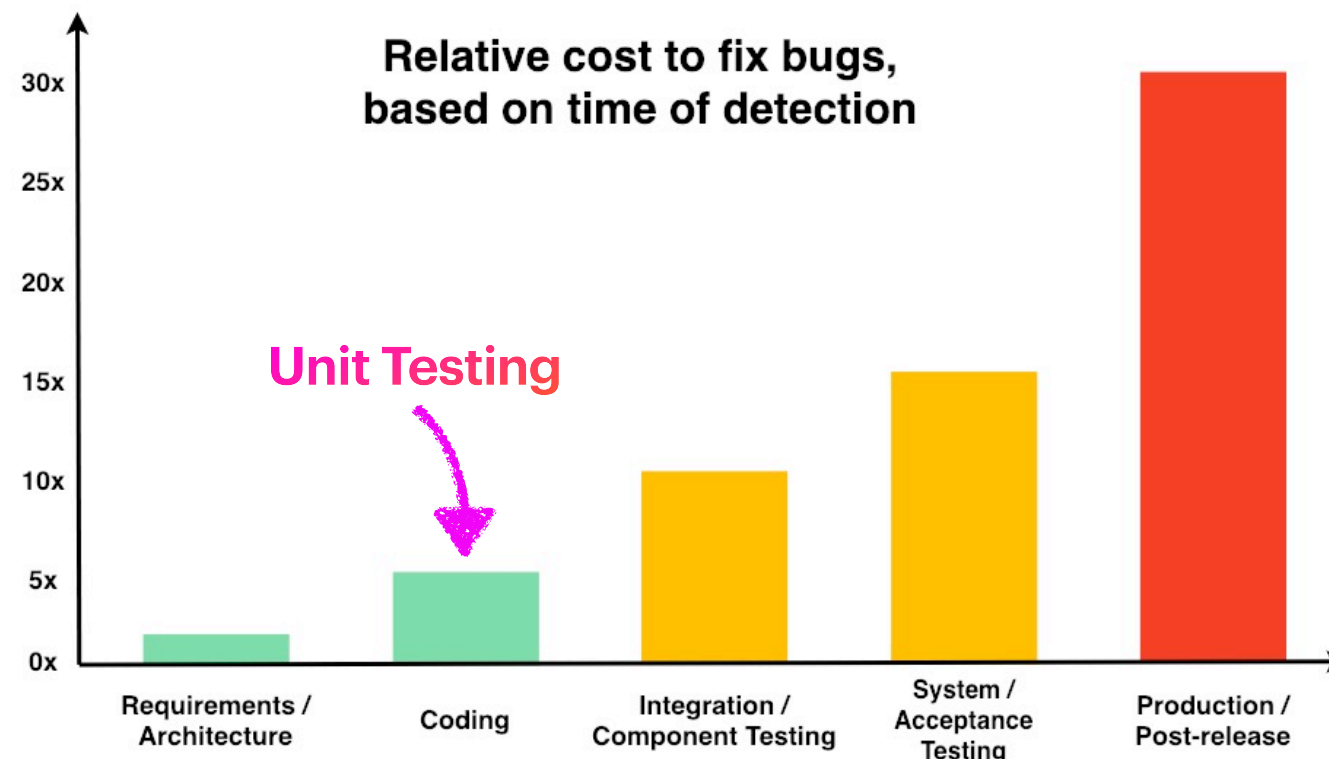


# Software Testing

- Introduzione al testing del software
- Unit Testing
  - Unit test con Java (*JUnit*)
- **Continuous Integration**
  - Automazione di unit test (*GitHub Actions*)

# Continua verifica del software

- Continua verifica del software **aumenta la probabilità** di rilevare difetti
  - Controllo che nuove funzionalità non introducano errori (*regression testing*)
- **Costo per correggere un bug** aumenta esponenzialmente in funzione del tempo in cui viene rilevato
  - Più tardi viene scoperto nel ciclo di sviluppo, più aumentano i costi per sistemarlo

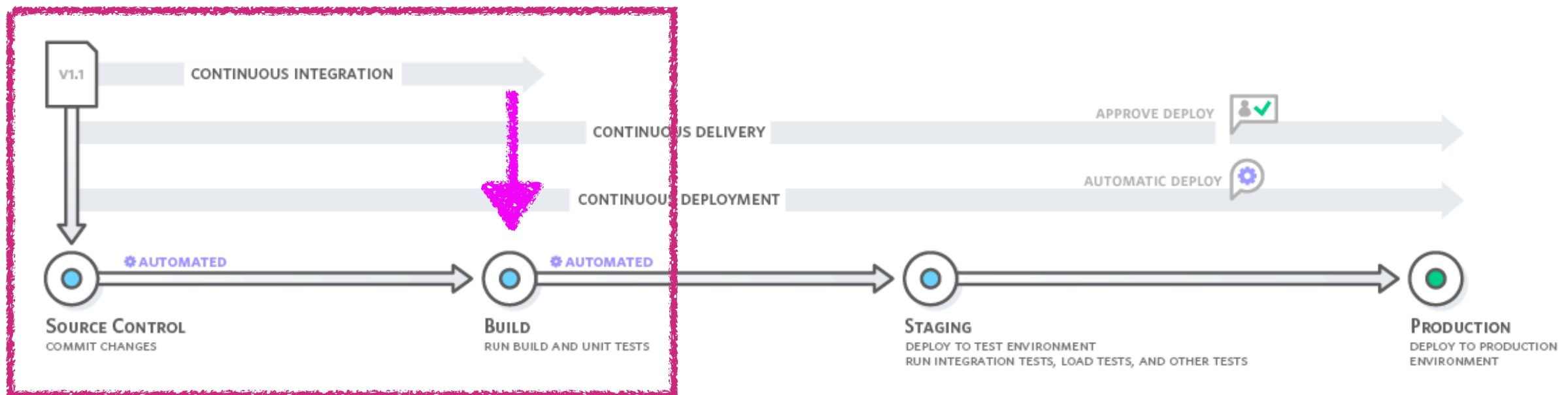




# Continuous Integration

**Continuous Integration (CI)**: sincronizzare periodicamente ad un repository centrale le modifiche realizzate dagli sviluppatori

- Nel repository centrale, quando viene aggiunta una nuova modifica (oppure ad intervalli regolari), il codice sorgente viene automaticamente **compilato** e vengono eseguiti una serie di **unit test**



# Vantaggi della CI



## **Maggiore produttività per gli sviluppatori**

Compilazione e testing vengono svolti in automatico



## **Individuare e risolvere bug con maggiore tempestività**

Aumentando la frequenza del testing, è più facile individuare prima eventuali errori



## **Migliorare la qualità del software**

Il codice viene costantemente controllato



## **Aggiornamenti più rapidi**

Riduce il tempo per validare e pubblicare nuovi aggiornamenti

# Software Testing

- Introduzione al testing del software
- Unit Testing
  - Unit test con Java (*JUnit*)
- Continuous Integration
  - **Automazione di unit test (*GitHub Actions*)**

# GitHub Actions

- Un esempio di piattaforma per CI sono le **GitHub Actions**
  - Creazione di **workflows** che vengono eseguiti su un repository in base a specifici eventi (*push di un commit, creazione di un issue, pubblicazione di una nuova release*)
  - I workflow sono definiti tramite file (*in formato YAML*) salvati all'interno del repository (nella cartella `.github/workflows/`)
  - Consentono di automatizzare la compilazione, l'**esecuzione di test** e la distribuzione di software contenuto in un repository GitHub



# Unit Tests Workflow

```
name: Unit Tests
```

```
on: [push]
```



```
jobs:
```

```
  unit-tests:
```

```
    runs-on: ubuntu-latest
```

```
    steps:
```

```
      - name: Checkout repository
        uses: actions/checkout@v3
```

```
      - name: Set up JDK 17
        uses: actions/setup-java@v3
        with:
```

```
          java-version: '17'
```

```
          distribution: 'temurin'
```

```
      - name: Run the Maven package phase
        run: mvn -B package --file pom.xml
```

- Automazione della compilazione ed esecuzione di **unit test** dopo ogni **push** di un commit
- Controllo di non aver introdotto errori con le ultime modifiche (*regression testing*)



**Fixed JDK version**

Unit Tests #2: Commit c2aa1b2 pushed by lucamora



**Added GitHub Actions workflow**

Unit Tests #1: Commit 85d3a42 pushed by lucamora



**Esempio:** [https://github.com/lucamora/simple-stats/runs/6241108931?check\\_suite\\_focus=true#step:4:529](https://github.com/lucamora/simple-stats/runs/6241108931?check_suite_focus=true#step:4:529)

# Grazie per l'attenzione!

Applicazione di esempio e  
slide della presentazione

