Unit Testing – Assertions

ALIN ZAMFIROIU

Ce este testarea?

- Procesul de căutare a erorilor și al defectelor?

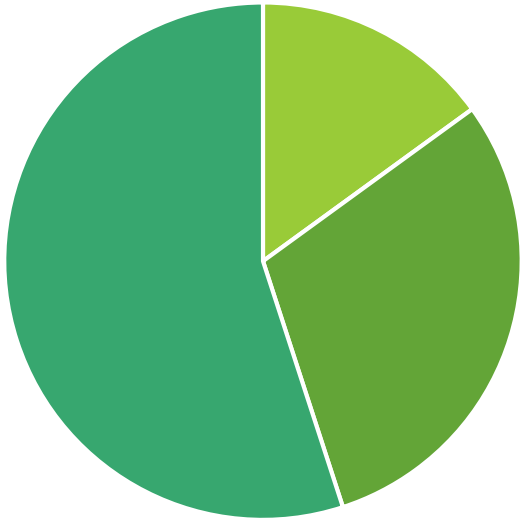
- Este utilizata pentru a semnala prezența defectelor, dar nu garantează absența acestora. - **Dijkstra**

Termeni specifici

- Eşec
- Defect
- Excepție
- Problemă
- Eroare
- Incident
- Anomalie
- Inconsistență
- Aparență
- Neajuns
- Bug

Cauzele erorilor software

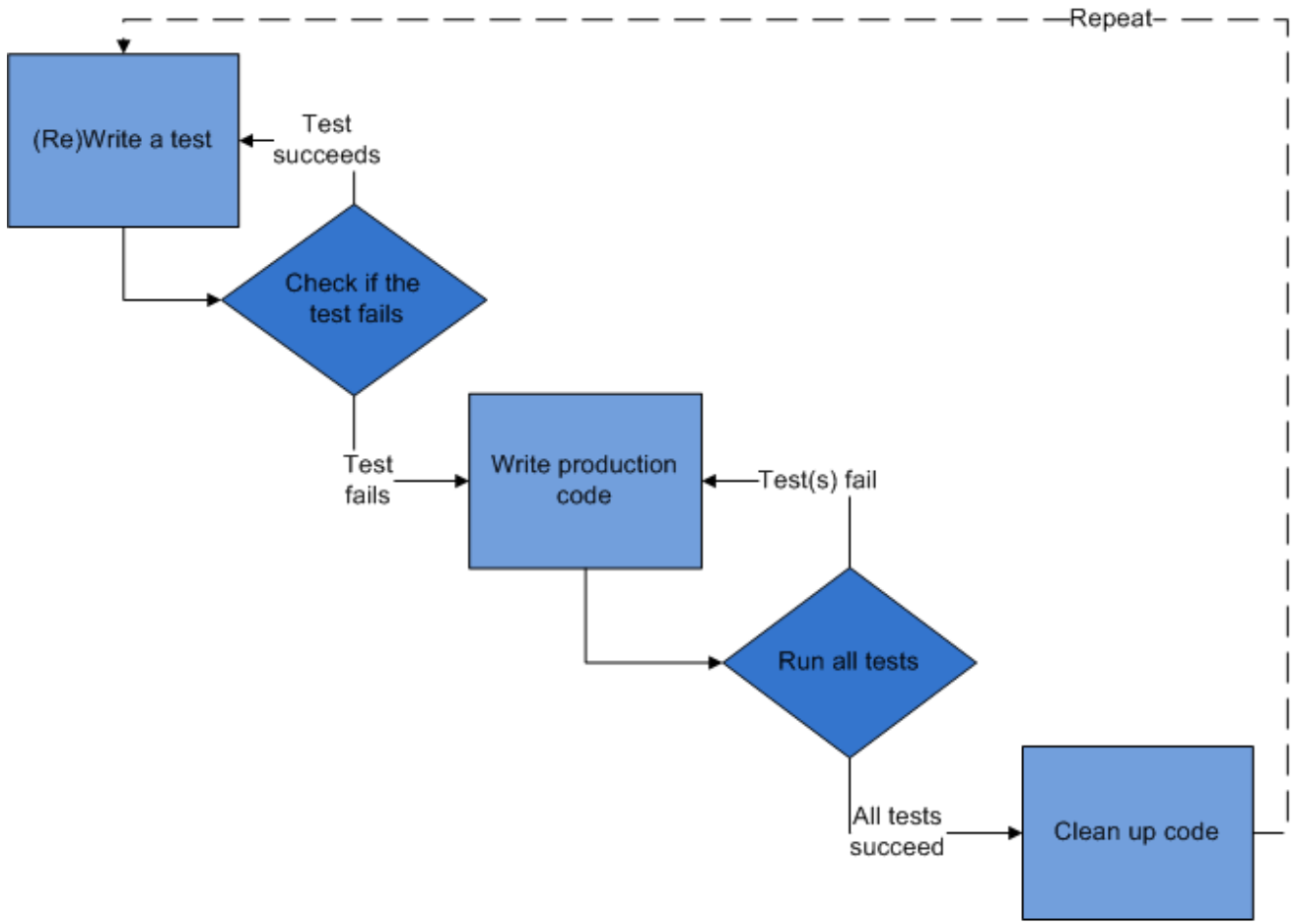
- Erori de programare – 15%
- Erori de proiectare – 30%
- Erori de specificații – 55%



Ce este Unit Testing?

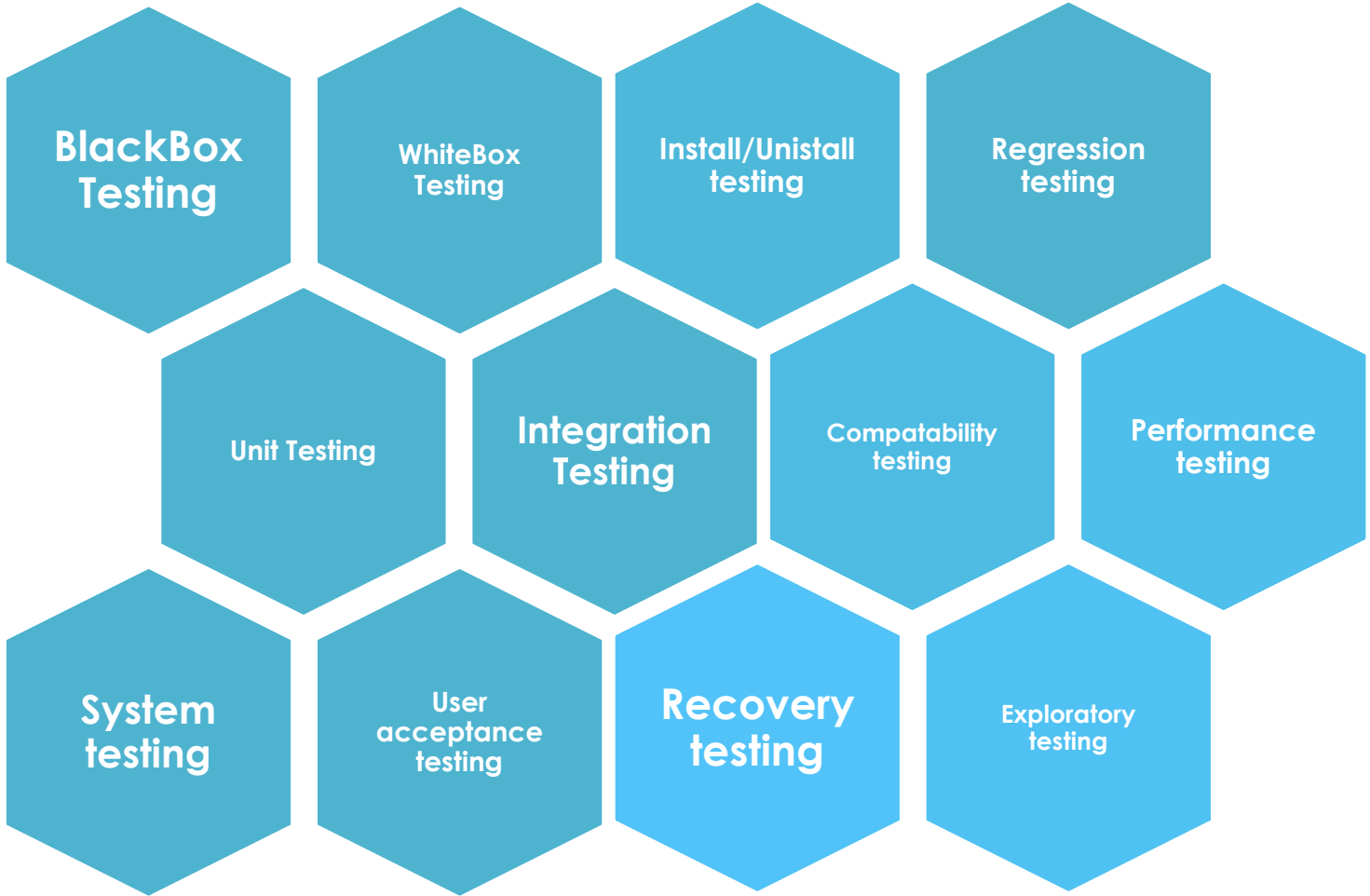
- O cale de testare a codului, de către programatori încă din etapa de dezvoltare a produsului software.
- **UNIT TEST** – secvență de cod folosită pentru testarea unei unități bine definite din codul aplicației software. De obicei unitatea este reprezentată de o metodă.
- Testarea unității se realizează într-un context bine definit în specificațiile de testare.

Test Driven Development

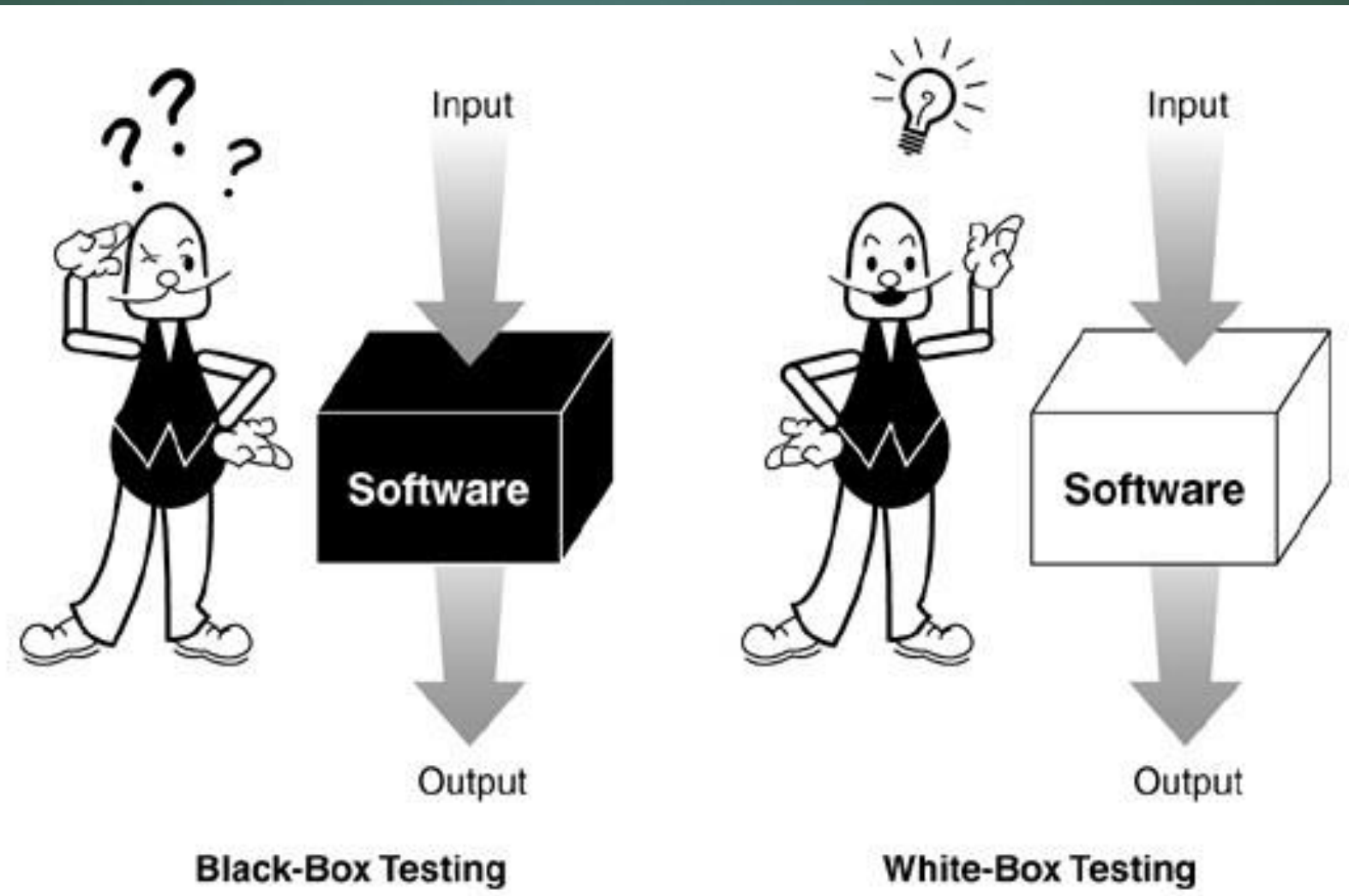


- Dezvoltarea pe baza testelor.
- Este exact modul de gândire al oamenilor pentru realizarea metodelor.

Tipuri de testare



WhiteBox sau BlackBox testing



Testarea BlackBox

- Testarea **BlackBox** sau testarea comportamentală, este o metodă utilizată pentru a testa aplicația software de către persoane care nu cunosc arhitectura internă a aplicației testate.

- Testerul cunoaște doar datele de intrare și datele de ieșire ale aplicației.

Avantajele testării BlackBox

- Testele sunt realizate din punctul de vedere al utilizatorului.
- Testerul nu trebuie să știe programare, limbajul folosit pentru dezvoltare sau structura de cod a aplicației.
- Testele sunt efectuate independent de dezvoltatorii și au o perspectivă obiectivă.

Dezavantajele testării BlackBox

- Cazurile de testare sunt dificil de proiectat, deoarece testerul nu are caietul de sarcini al aplicației;
- Testele vor avea un număr mic de intrări;
- Testele pot fi redundante cu alte teste realizate de dezvoltator.

Testarea WhiteBox

- Este o metodă de testare folosită de către dezvoltatori sau de testeri care cunosc structura internă a aplicației testate.

Testarea WhiteBox

- Testarea WhiteBox mai este cunoscută și sub formele:
  - Clear Box Testing;
  - Open Box Testing;
  - Glass Box Testing;
  - Transparent Box Testing;
  - Code-Based Testing;
  - Structural Testing.

Avantajele testării WhiteBox

- Testarea poate fi începută într-o etapă anterioră punerii în funcțiune. Nu trebuie să se aștepte realizarea interfeței pentru a fi realizată testarea.

- Testarea este mai aprofundată, cu posibilitatea de a acoperi cele mai multe posibilități.

Dezavantajele testării WhiteBox

- Din moment ce testele pot fi foarte complexe, sunt necesare resurse de înaltă calificare, cu o cunoaștere aprofundată a programării și a punerii în aplicare.

- Întreținerea codului de testare poate fi o povară în cazul în care punerea în aplicare se schimbă foarte des.

Motive să folosești teste unitare

- Ușor de scris
- Testele pot fi scrise ad-hoc atunci când ai nevoie de ele
- Deși sunt simple, pe baza lor se pot defini colecții de teste – Test Suites
- Pot fi rulate automat de fiecare dată când e nevoie (write once, use many times)

Motive să folosești teste unitare

- Există multe framework-uri și instrumente ce simplifică procesul de scriere și rulare
- Reduc timpul pierdut pentru debugging și pentru găsirea bug-urilor
- Reduc numărul de bug-uri în codul livrat sau integrat
- Crește rata bug-urilor identificate în faza de scrierea a codului

Motive să nu folosești teste unitare

- De ce trebuie să îmi testez codul ?
- Codul scris de mine este corect !!!
- Nu am timp de teste. Trebuie să implementez funcționalități, nu teste.
- Nu este trecut în specificații că trebuie să facem teste.

Framework-uri folosite pentru testare unitară

Framework	Programming language / scripting
JUnit	Java
PHPUnit	PHP
PyUnit	Python
CPPUnit	C++
VBUnit	Visual Basic
DUnit	Delphi
cfcUnit	ColdFusion
HTMLUnit	Html și JavaScript
Jsunit	JavaScript
dotUnit	.NET
NUnit	C#, ASP.NET
Ruby	Ruby
XMLUnit	XML
ASPUnit	ASP
xUnit	C#

JUnit

- Este un framework ce permite realizarea și rularea de teste pentru diferite metode din cadrul proiectelor dezvoltate.
- Cel mai folosit framework pentru testarea unitară a codului scris în JAVA.
- Reprezintă o adaptare de la xUnit.

Junit - istoric

- **Kent Beck** a dezvoltat in anii 90 primul instrument de testare automata, **xUnit**, pentru Smalltalk
- Beck si Gamma (Gang of Four) au dezvoltat **JUnit** in timpul unui zbor de la Zurich la Washington, D.C.
- **Junit** a devenit instrumentul standard pentru procesele de dezvoltare de tip **TDD** - Test-Driven Development in Java
- **Junit** este componenta standard in multiple IDE-uri de Java (Eclipse, BlueJ, Jbuilder, DrJava, InteliJ)
- Intrumentele de tip **Xunit** au fost dezvoltate si pentru alte limbaje (Perl, C++, Python, Visual Basic, C#, …)

JUnit

- JUnit funcționează conform a două design patterns: **Composite** și **Command**.

- O clasă TestCase reprezintă un obiect Command iar o clasă TestSuite este compusă din mai multe instanțe TestCase sau TestSuite.

Concepte JUnit

- **Fixture** – set de obiecte utilizate în test
- **Test Case** – clasă ce definește setul de obiecte (fixture) pentru a rula mai multe teste
- **Setup** – o metodă/etapă de definire a setului de obiecte utilizate (fixture), înainte de testare.
- **Teardown** – o metodă/etapă de distrugere a obiectelor (fixture) după terminarea testelor
- **Test Suite** – colecție de cazuri de testare (test cases)
- **Test Runner** – instrument de rulare a testelor (test suite) și de afișare a rezultatelor

JUnit- Assertions

assertEquals(expected, actual)

assertEquals(expected, actual, delta)

assertSame(expected, actual)

assertNotSame(expected, actual)

assertNull(object)

assertNotNull(object)

assertTrue(condition)

assertFalse(condition)

fail(message)

assertEquals(message, expected, actual)

assertEquals(message, expected, actual, delta)

assertSame(message, expected, actual)

assertNotSame(message, expected, actual)

assertNull(message, object)

assertNotNull(message, object)

assertTrue(message, condition)

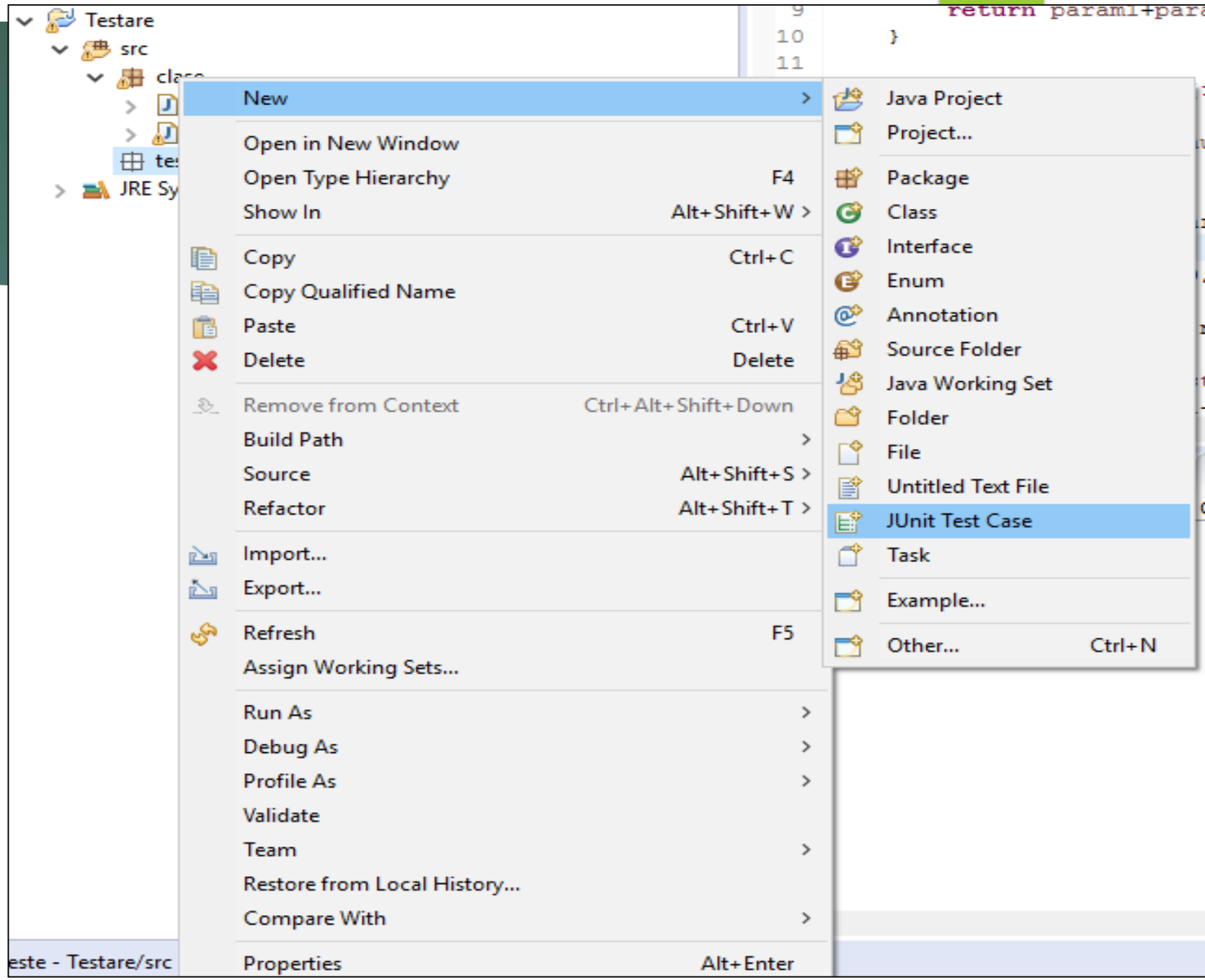
assertFalse(message, condition)

fail(message)

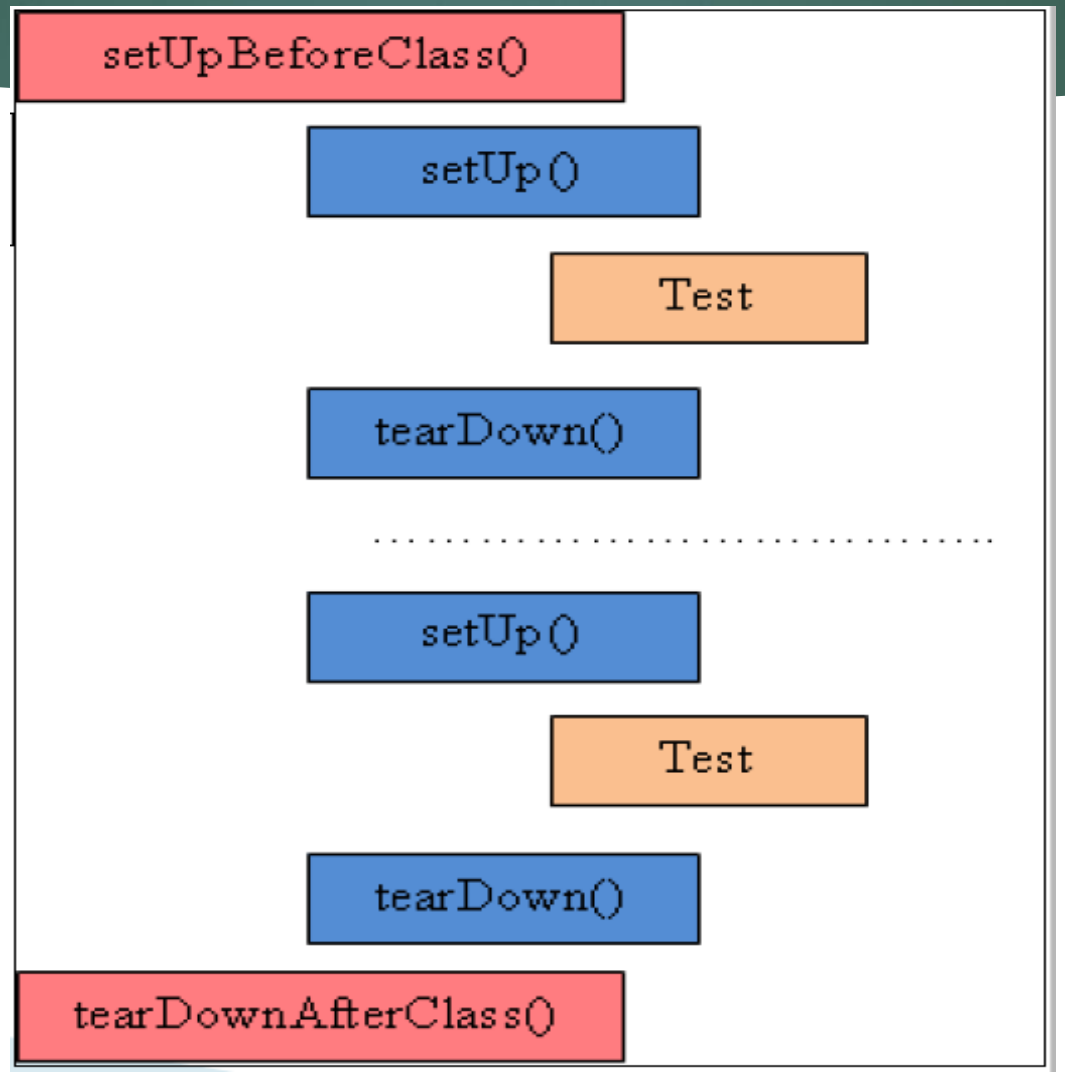
Junit și NUnit

JUnit	NUnit
assertEquals	Assert.AreEqual
assertEquals	Assert.AreNotEqual
assertSame	Assert.AreSame
assertNotSame	Assert.AreNotSame
assertNull	Assert.IsNull
assertNotNull	Assert.IsNotNull
assertTrue	Assert.IsTrue
assertFalse	Assert.IsFalse

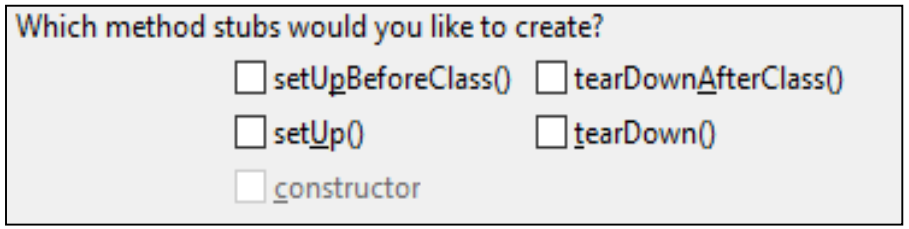
Test Case


Testare
src
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Run As
Debug As
Profile As
Validate
Team
Restore from Local History...
Compare With
Properties Alt+Enter
Java Project
Project...
Package
Class
Interface
Enum
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Task
Example...
Other... Ctrl+N
return param1+para
}

Unit testing - Junit skeleton



Unit testing - Junit skeleton




```
setUpBeforeClass
        setUp
                Test1
        tearDown
        setUp
                Test2
        tearDown
        setUp
                Test3
        tearDown
tearDownBeforeClass
```

JUnit3 și JUnit4 – diferențe

- JUnit 3 are nevoie de o versiune de JDK mai nouă decăt JDK 1.2 în timp ce JUnit 4 are nevoie de o versiune mai nouă decât JDK 5;
- în JUnit 3 clasele de test trebuie să fie derivate din clasa TestCase, iar în JUnit 4 nu este necesară moștenirea clasei TestCase;
- în JUnit3 numele metodelor de test este construit după formatul *testAAA*; astfel toate metodele de test conțin în numele acestora cuvântul *test* în JUnit4 numele metodele nu este important; însă metodele care sunt rulate ca teste au adnotarea *@Test*;
- în JUnit 4 este folosită adnotarea timeout pentru verificarea finalizării testului într-un interval de timp precizat: *@Test(timeout=1000)*; valoarea pentru timeout este setată în milisecunde;
- pentru ca un test să nu fie rulat în Junit3 trebuie șters, comentat sau modificat numele, astfel încât să nu respecte formatul testAAA; în JUnit 4 testul care nu se dorește a fi rulat primește adnotarea *@Ignore* sau se șterge adnotarea @Test.

Unit testing - Junit skeleton

- *Adnotarile utilizate în JUnit4 pentru metodele automate din skeleton:*
  - *@BeforeClass* pentru metoda *setUpBeforeClass();*
  - *@AfterClass* pentru metoda *tearDownAfterClass();*
  - *@Before* pentru metoda *setUp();*
  - *@After* pentru metoda *tearDown()*.
- În JUnit3 neexistând adnotări numele metodelor *setUp()*, respectiv, *tearDown()* sunt obligatorii.

JUnit5

- Adnotările pentru structura unui test s-au schimbat

JUnit4	JUnit5 - Jupiter
@BeforeClass	@BeforeAll
@AfterClass	@AfterAll
@Before	@BeforeEach
@After	@AfterEach

What next?

- What to test?