Curs MAP : Testarea programelor

Florentin Bota

Data: 12.01.2021

Ora: 10:00

UBB

Testarea programelor

Program (informatic): o grupare de instrucțiuni, metode sau module ce pot fi executate de către un calculator.

Testarea:

- reprezintă un proces de investigație al unui program
- semnalează prezența defectelor software
- observă comportamentul programului în cadrul mai multor execuții
- oferă beneficiarilor informații referitoare la calitatea programului testat.

De ce e importantă testarea

2018-2019:

- două avioane Boeing 737 Max s-au prăbușit (Indonezia și Etiopia)
- Cauză probabilă: **bug software**, senzor defect și decizii legate de calitatea codului și a avionului (elementele de fail-safe erau opționale, pentru a reduce costul)
- 346 de victime
- Costuri pentru Boeing: 9.2 miliarde de dolari

De ce e importantă testarea

2020:

- Cyberpunk 2077 este lansat de către CD Project Red în decembrie 2020, după o serie de amânări, la presiunea investitorilor și a utilizatorilor
- Jocul este unul așteptat de peste 10 milioane de fani
- Din păcate acesta prezintă o serie de probleme și bug-uri (cădere de pe hartă, corupere fișier de save etc.), în special pe consolele mai vechi, ceea ce duce la retragerea jocului de către Sony de pe Playstation Store și la o serie de refunds
- Problemele au ca rezultat prăbușirea acțiunilor dezvoltatorului și o scădere a valorii acestora cu 1.8 miliarde de dolari în primele zile de la lansare

Program testat

- **Program testat (software under test, SUT):**
- **≈ funcţie matematică;**

P : D ⟶ R, unde

- **D – mulţimea datelor de intrare;**
- **R – mulţimea datelor de ieşire.**

Caz de testare

Caz de testare: Set de date de intrare, condiții de execuție și rezultate așteptate (proiectate cu un anumit scop), prin intermediul căruia un tester poate determina dacă un program sub testare satisface cerințele de funcționare sau rulează corect.

- o interogare adresată de tester programului testat

Notație: (i , r), i ϵ D, r ϵ R;

•pentru intrarea i se așteaptă să se obțină rezultatul r.

Caracteristicile unui caz de testare

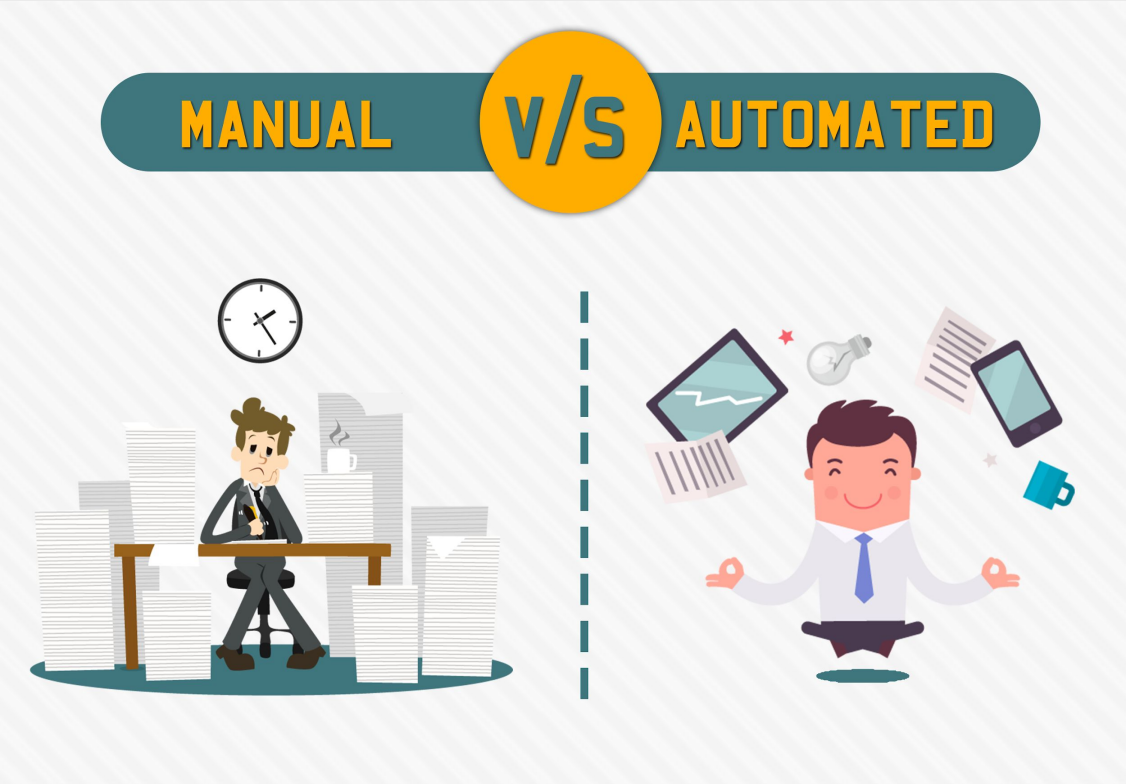
- probabilitate mare de a identifica bug-uri;
- nu este redundant;
- relevant în cadrul categoriei din care face parte;
- nu este prea simplu;
- nu este prea complex.
- atomic (order independent)
- repetabil
- reutilizabil

Tipuri de testare

În funcție de mulțimea cazurilor de testare:

- **Exhaustivă** (toate cazurile de testare posibile, toate scenariile etc)
  - Nu este realistă, deoarece de multe ori D nu este finit, sau conține foarte multe elemente
- **Selectivă** (se selectează o parte din D, conform anumitor criterii)
- **Depanare** (bug fixing, debugging)
  - Se urmărește localizarea și eliminarea unui bug identificat in sistem

Tipuri de testare - Manual vs Automat



Source

Tipuri de testare

- Criteriul **cutiei negre** (testare **Black-box**) – testare funcţională:
  - Partiţionarea în clase de echivalenţă;
  - Analiza valorilor limită;
  - Tabele de decizie, Cazuri de utilizare, Scenarii de utilizare, etc.;
- Criteriul **cutiei transparente** (testare **White-box**) **– testare structurală:**
  - Acoperirea fluxului de control (e.g., instrucţiuni, ramificaţii, decizii, condiţii, bucle, drumuri);
  - Acoperirea fluxului de date; (bazat pe urmărirea variabilelor)
- Criteriul **cutiei gri** (testare **Grey-box**) **– testare mixtă:**
  - folosirea simultană a avantajelor abordărilor black-box şi white-box pentru proiectarea cazurilor de testare;
- Criteriul **statistic:**
  - generarea aleatoare de date de test pe baza unor modele;
  - experienţa testerului.

Tipuri de teste - pe scurt

- **Unit tests**
  - Low level, de obicei la nivel de metode/funcții din cadrul unei clase sau modul software
- **Integration tests**
  - Verifică funcționarea corectă a mai multor module sau servicii din cadrul aplicației (interacțiuni cu baza de date, microservicii)
- **Functional tests**
  - Se testează din punctul de vedere al cerințelor business ale aplicației și se pune accent pe acțiuni -> rezultat așteptat

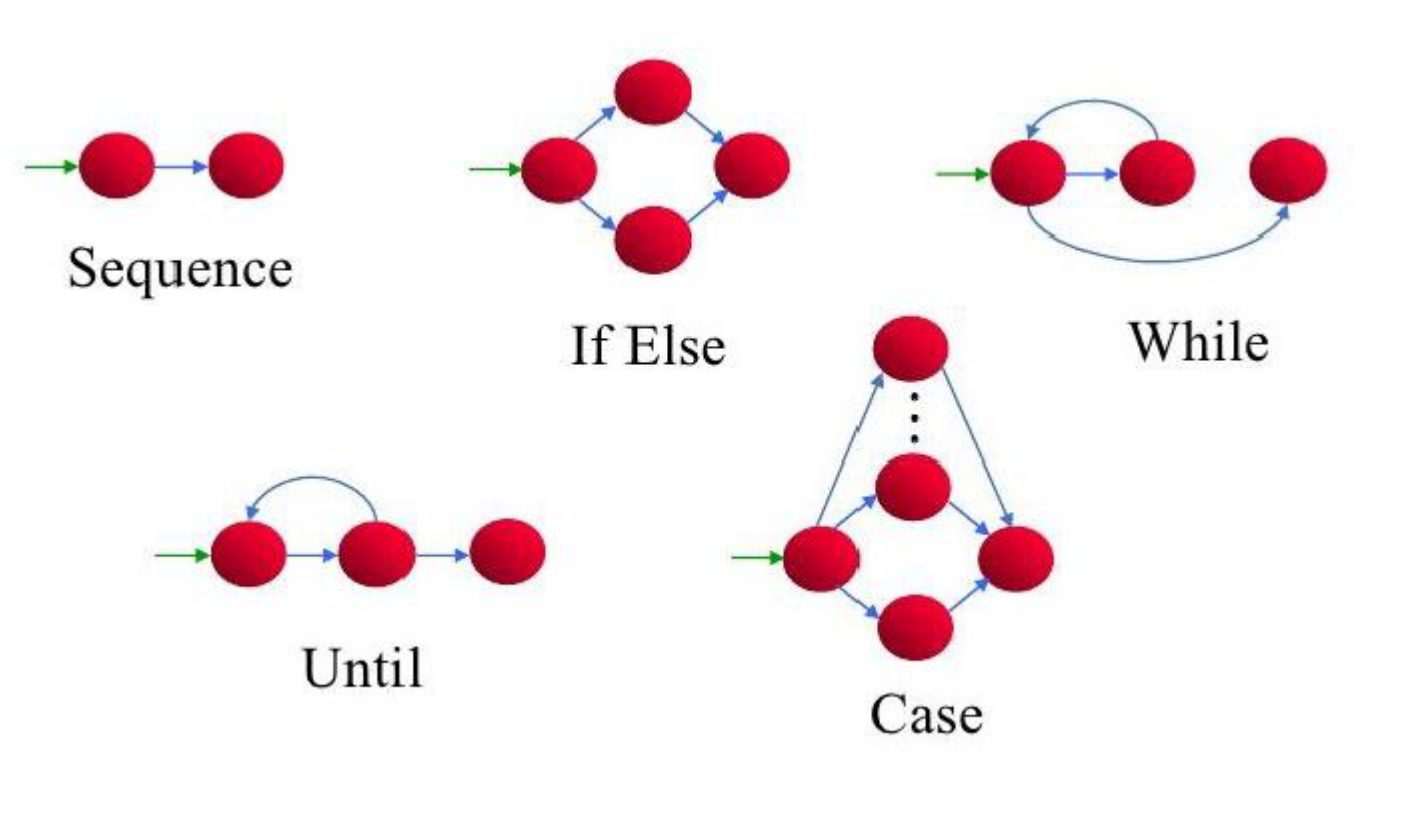
Tipuri de teste - pe scurt

- **End-to-end tests**
  - Se testează scenarii complexe de utilizare, flow-uri complete de acțiuni, ce pot avea mai multe ramuri de finalizare.
  - Necesită un efort mare de dezvoltare și mentenanță
- **Acceptance testing**
  - Teste formale, cu accent pe feedback-ul direct al userilor/clienților, stakeholders etc
  - Se testează scenarii complexe și se verifică dacă acestea corespund cerințelor business ale aplicației
- **Performance testing**
  - Se testează comportamentul sistemului sub sarcină (timp de răspuns, stabilitate, etc)
- **Smoke testing**
  - Testare rapidă a funcționalităților de bază, utilizat de obicei pentru a verifica direcții noi de dezvoltare a aplicației, platforme noi de deploy etc.

Testare White-box

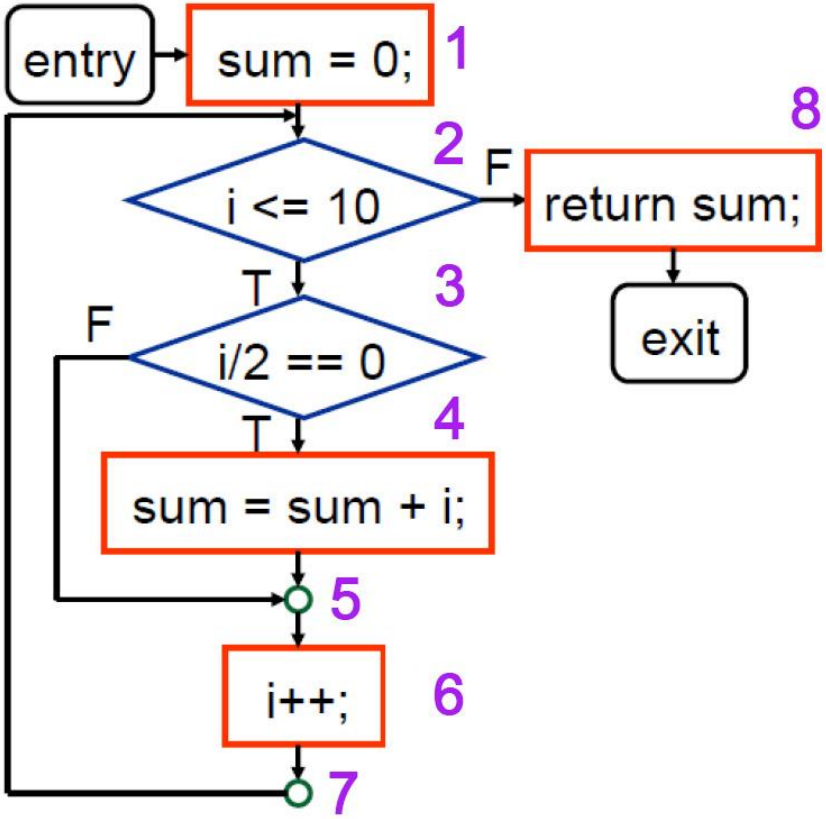
- Testare bazată pe **fluxul de control** (ordinea în care sunt apelate instr.)
  - utilizează structurile de control pentru proiectarea cazurilor de testare;
  - scop: acoperirea prin cazuri de testare la un nivel satisfăcător a structurilor de control din programul testat;
- **Componente**:
  - graful fluxului de control (**Control Flow Graph - CFG**);
    - reprezentare grafică detaliată a unei unităţi de program;
    - permite vizualizarea tuturor drumurilor din unitatea de program;
  - complexitatea ciclomatică.

CFG - instrucțiuni



CFG - Exemplu

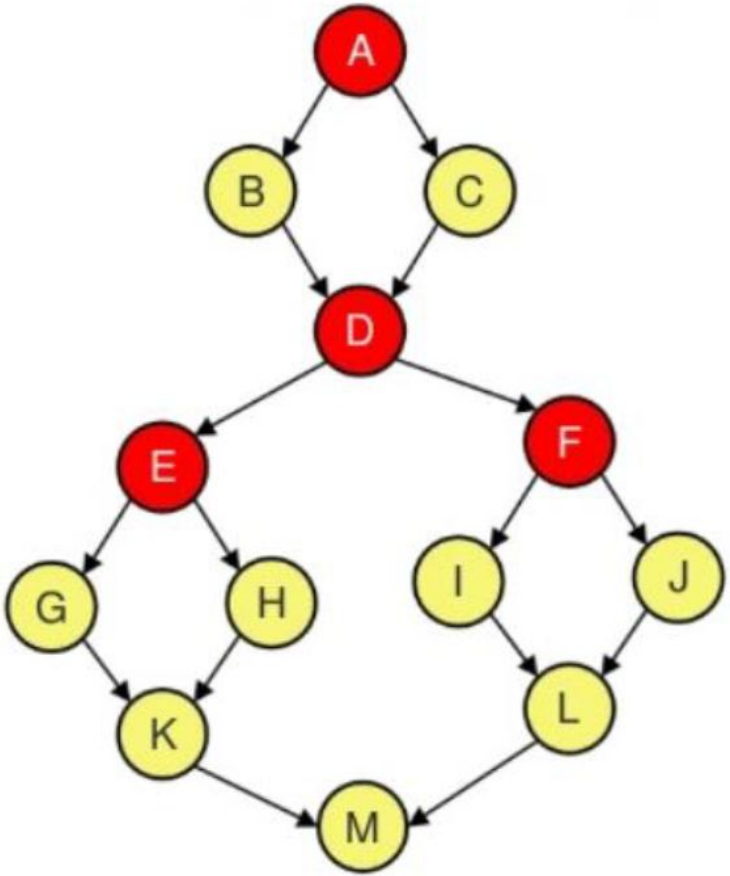
```
*  int evenSum(int i) {
1      int sum = 0;
2      while (i <= 10) {
3          if (i/2 == 0) {
4              sum = sum + i;
5          }
6          i++;
7      }
8      return sum;
*  }
```



CFG - Complexitatea ciclomatică

- **Complexitatea ciclomatică** (McCabe's cyclomatic complexity, **CC**):
  - metrică software aplicată pentru măsurarea cantitativă a complexităţii logice a unui program;
  - Permite determinarea numărului de drumuri independente din mulţimea de bază a unui CFG;
- **Modalităţi de calcul a CC la nivelul CFG:**
  - CC = numărul de regiuni din CFG;
  - CC = E − N + 2, unde E - #arce, N - #vârfuri ;
  - CC = P + 1, unde P - #vârfuri condiţie.
- **Regiune:**
  - zonă a CFG mărginită parţial sau în totalitate de arce şi vârfuri;

CFG - Complexitatea ciclomatică - exemplu



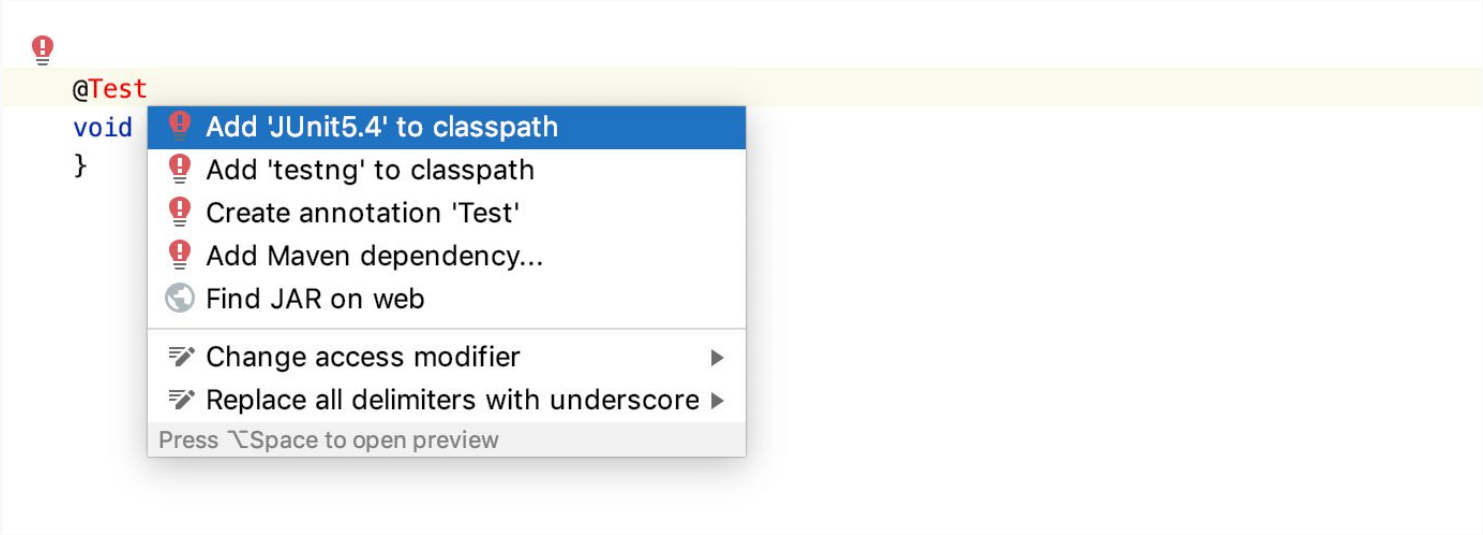
- drumuri independente:
  - drum 1: A-B-D-E-G-K-M;
  - drum 2: A-C-D-E-G-K-M;
  - drum 3: A-B-D-F-I-L-M;.
  - drum 4: A-B-D-E-H-K-M;
  - drum 5: A-C-D-F-J-L-M.
- CC pentru CFG:
  - CC = numărul de regiuni = 5 regiuni = 5;
  - CC = E – N + 2 = 16 arce - 13 vârfuri + 2 = 5;
  - CC = P + 1 = 4 vârfuri condiţie + 1 = 5.

Model Based Testing

- **Model Based Testing**
  - Utilizează un model abstractizat pentru a descrie aspectele funcționale ale programului testat (SUT)
  - Cazurile de testare sunt derivate pe baza acestui model
  - Necesită un efort mai mare de dezvoltare (inițial)
  - Permite testare exhaustivă prin automatizare

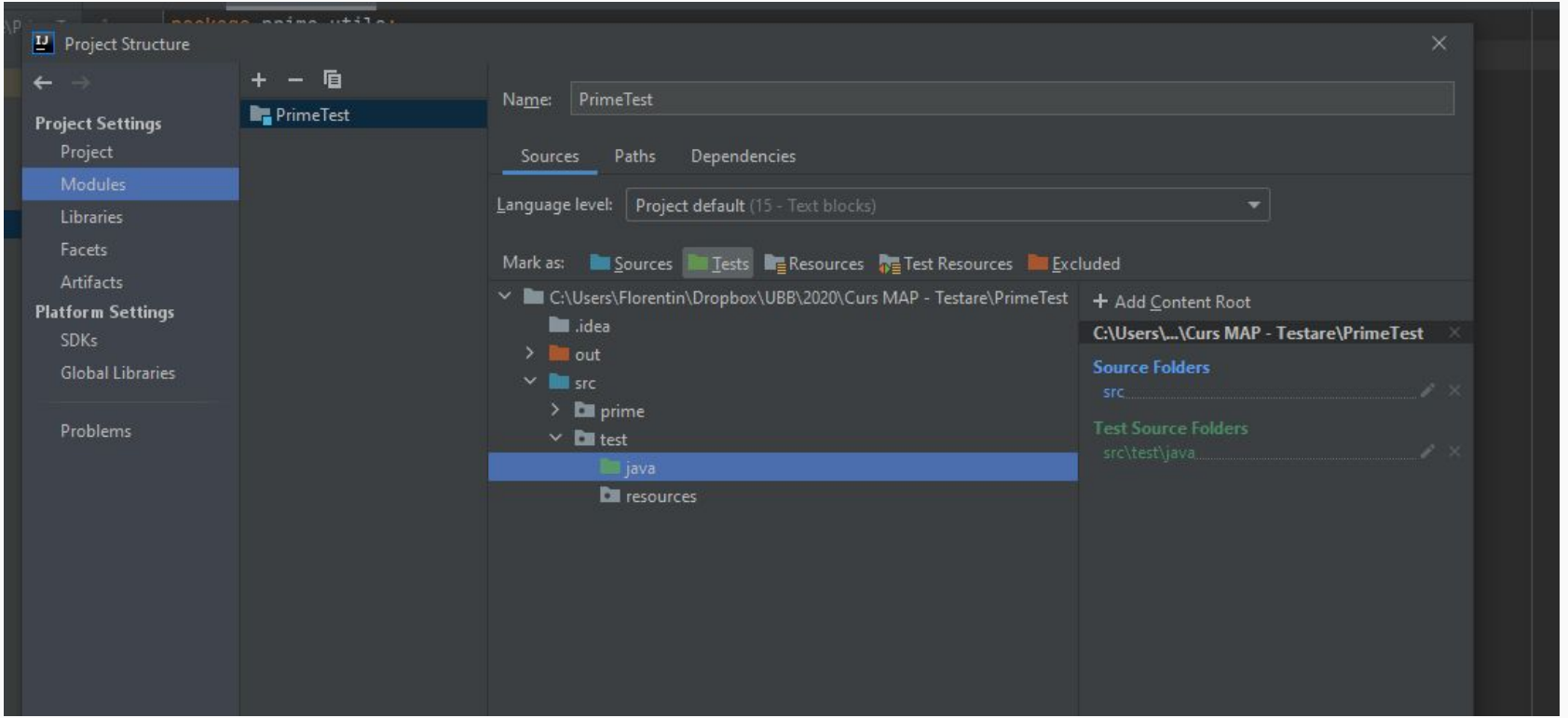
Unit Tests in JAVA

- **JUnit**
  - Framework de testare utilizat in Java ce permite implementarea ușoară a testelor

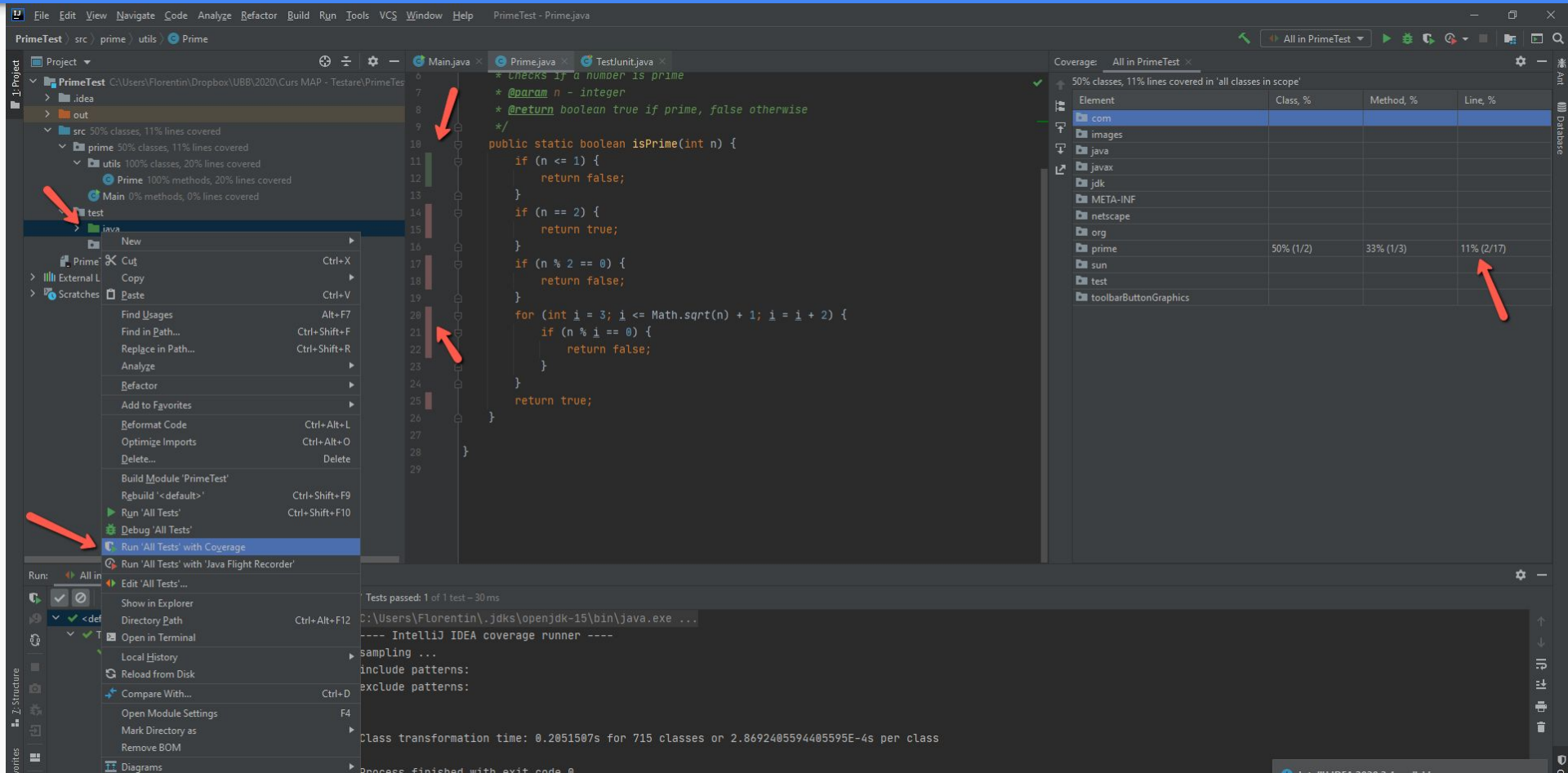


Unit Tests in JAVA

- **Setup Test Sources Root**



Unit Tests in JAVA - Run with coverage



Unit Tests in JAVA - Example

```java
import org.junit.jupiter.api.Test;
import prime.utils.Prime;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;


public class TestJunit {

    @Test
    void isPrime_NegativeNumber_False (){

        // Setup
        int a = -1;

        // Act
        boolean result = Prime.isPrime(a);

        // Assert
        assertFalse(result);
    }
}
```

JUnit

- **Assert**
  - void assertEquals(boolean expected,boolean actual): checks equality
  - void assertTrue(boolean condition): checks that a condition is true.
  - void assertFalse(boolean condition): checks that a condition is false.
  - void assertNull(Object obj): checks that object is null.
  - void assertNotNull(Object obj): checks that object is not null.

JUnit assert exceptions

```
@Test
public void whenExceptionThrown_thenAssertionSucceeds () {

    Exception exception = assertThrows(NumberFormatException.class, () -> {
        Integer.parseInt("1a");
    });

}
```

Bibliografie

[Frentiu2010] M. Frentiu, Verificarea si validarea sistemelor soft, Presa Universitara Clujeana, 2010.

•[Myers2004] Glenford J. Myers, The Art of Software Testing, John Wiley & Sons, Inc., 2004.

•[NT2005] K. Naik and P. Tripathy. Software Testing and Quality Assurance, Wiley Publishing, 2005.

•[Patton2005] R. Patton, Software Testing, Sams Publishing, 2005.

•[Collard2003] J. F. Collard, I. Burnstein. Practical Software Testing. Springer-Verlag New York, Inc., 2003.

•[Beizer1990] Beizer, B., Software Testing Techniques, Van Nostrand Reinhold., New York, 1990.

•VVSS Lectures and Seminars by Dr. Maria Camelia Chisăliţă-Creţu, Lecturer in Computer Science, Babes-Bolyai University

Resurse online:

https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

https://www.jetbrains.com/help/idea/testing.html