Unit Testing: Right-BICEP

ALIN ZAMFIROIU

Recapitulare

- Testing
- Unit testing
- Junit
- Test
- TestCase
- TestSuite
- Assertion
- Skeleton

What to test?

Right-BICEP

- Acum știm cum să facem teste, mai trebuie să știm ce trebuie să testăm.
- Persoanele cu experiență știu direct ce să testeze, din propria experiență.
- Persoanele cu mai puțină experiență au nevoie de direcții de urmat sau de principii care să îi îndrume în realizarea acestor teste.
- Cel mai cunoscut principiu de testare este Right-BICEP

Right-BICEP

- **RIGHT** – dacă rezultatele furnizate de către metodă sunt corecte;
- **B** – trebuie verificate toate limitele (**B**oundery) și dacă în cazul acestor limite rezultatele furnizate de metoda testată sunt de asemenea corecte;
- **I** – trebuie verificate relațiile inverse (**I**nverse);

Right-BICEP

- **C** – trebuie verificată corectitudinea printr-o verificare încrucișată (**C**ross-**C**heck), folosind metode de calcul asemănătoare, testate și validate de către o comunitate mare de programatori;

- **E** – trebuie simulată și forțată obținerea erorilor (**E**rrors) pentru verificarea comportamentului metodei în cazul anumitor erori;

- **P** – trebuie verificată păstrarea performanței (**P**erformance) între limitele acceptanței pentru produsul software final.

Unit testing - Right

- De fiecare dată când testăm o metodă, primul lucru care ar trebui verificat este că această metodă oferă rezultatele corecte.
- De aceea prima direcție este de a verifica corectitudinea rezultatelor.
- Această verificare se face în conformitate cu specificațiile proiectului dezvoltat.

Unit testing - Boundary

- Problemele apar de obicei la "*margini*", deci trebuie să fim atenți să testăm metodele pentru limitele intervalelor.

- Pentru fiecare metodă, trebuie să determinăm intervalul în care pot fi valorile parametrilor de intrare, precum și intervalul de rezultate furnizat de metodă.

Unit testing - Boundary

- Odată ce aceste limite au fost determinate, se efectuează teste exact pentru aceste valori.
- Testele Boundery nu presupun testarea valorilor din afara acestor valori, ci verificarea corectitudinii acestor valori - valori limită.
- Există de obicei limite inferioare și limite superioare. Testele se fac pentru ambele situații.

Unit testing - Boundary

- Pentru a identifica mai ușor limitele extreme, putem să utilizăm principiul CORECT :
  - C – Conformance;
  - O – Ordering;
  - R – Range;
  - R – References;
  - E – Existence;
  - C – Cardinality;
  - T – Time.

Unit testing - Inverse relationship

- Anumite metode pot fi testate prin aplicarea regulii inverse: pornind de la rezultat, trebuie să se ajungă la aceeași intrare de la care a început inițial.
- Nu se aplică pentru toate metodele. De obicei se aplică metodelor matematice.
- De asemenea, pentru bazele de date se poate verifica dacă a fost efectuată o inserare prin operația inversă: select.

Unit testing - Cross-Check

- Pentru fiecare metodă, putem incerca să o testăm utilizând altă metodă.
- De obicei, există mai multe modalități de a rezolva o problemă. Astfel, se poate utiliza o altă metodă pentru rezolvarea problemei pentru verificarea / testarea metodei nou implementate.

Unit testing - Cross-Check

- Această situație este posibilă atunci când metoda implementată a fost concepută pentru a crește productivitatea sau dacă metoda veche consuma prea multe resurse.

- Testarea metodei noi se face prin metoda veche, chiar dacă aeasta consumă mai multe resurse.

Unit testing - Error conditions

- Probabil cel mai urât scenariu pentru o aplicație este să crape. De aceea, atunci când testăm fiecare metodă unitar, trebuie să testăm și situațiile în care aplicația ar putea să crape.

- Dacă am studiat limitele extreme pentru valorile de intrare sau valori rezultate, testarea pentru furnizarea erorilor ar trebui să utilizeze valori în afara acestor intervale.

Unit testing - Error conditions

- Testarea de forțare a erorilor se face pentru toate metodele. Toate metodele au cel puțin o situație în care vor oferi erori. Testarea se face pentru aceste situații și se verifică dacă metoda tratează acel caz și aruncă sau oferă o excepție.

Unit testing - Performance

- Pentru diferite metode, este posibil să se testeze cât de bine funcționează metoda respectivă.

- Pe lângă testarea corectitudinii rezultatelor metodelor, este foarte important să se verifice perfomanța procesării.

Unit testing - Performance

- Verificarea performanței se face atât din punctul de vedere al resurselor consumate, cât și din punctul de vedere al timpului necesar pentru obținerea rezultatelor.

- Testarea performanței este efectuată atunci când input-ul sau rezultatul unei metode este reprezentat de o listă sau de un număr foarte mare de elemente, iar aceste valori pot crește foarte mult.

- Pentru JUnit4 pentru a testa timpul în care rulează o anumită metodă, este folosită următoarea adnotare : *@Test(timeout=100)*

CORRECT

- **C –** Conformitatea formatului (**C**onformance);
- **O –** Ordinea (**O**rder);
- **R –** Intervalul (**R**ange);
- **R –** Referințe externe (**R**eferences);
- **E –** Existența obiectelor sau a rezultatelor (**E**xistence);
- **C –** Cardinalitatea rezultatelor (**C**ardinality)
- **T –** Timpul (**T**ime).

CORRECT

- Fiecare sub-principiu are o întrebare care ar trebui să fie în mintea testerului.

- Acest principiu este folosit și pentru a stabilii condițiile limită pentru testele de Boundary din Right-BICEP.

Conformance

- Este, de asemenea, cunoscut sub numele de:
  - Type testing
  - Compliance testing
  - Conformity assessment

Conformance

- Se aplică în numeroase domenii în care ceva ar trebui să îndeplinească anumite standarde specifice.
- De obicei, pentru orice intrare și pentru orice ieșire, trebuie să se verifice conformitatea cu un format sau cu un standard.

Conformance

- Testele pot fi efectuate pentru a verifica ce se întâmplă dacă datele de intrare nu sunt conforme cu formatul sau pentru a vedea dacă rezultatul obținut este conform cu formatul specific proiectului.

Ordering

- Testele de ordine sunt specifice listelor, dar nu numai.
- În cazul listelor, trebuie să verificăm dacă ordinea articolelor este cea dorită.
- De asemenea, putem testa comportamentul metodei dacă primește anumiți parametri într-o altă ordine sau o listă de elemente într-o ordine diferită de cea așteptată.

Range

- Pentru valorile de **intrare** și de **ieșire**, sunt setate anumite intervale. Aceste intervale trebuie verificate.

- Pentru anumite metode sunt stabilite mai multe intervale. Acest lucru va fi testat pentru toate aceste intervale.

Range

- Toate funcțiile care au un index trebuie să fie testate pentru interval, deoarece acel index are un domeniu bine stabilit.
- De obicei, este necesar să verificați :
  - Valorile inițiale și finale pentru index au **aceeași valoare**;
  - **Primul** element este mai mare sau mai mic decât **ultimul** element;
  - Ce se întâmplă dacă indicele este **negativ**;
  - Ce se întâmplă dacă indicele este mai mare decât **limita superioară**;
  - Numărul de articole nu este același cu cel pe care îl doriți - **dimensiunea**;
  - etc.

Reference

- Anumite metode depind de lucrurile externe sau de obiectele externe acestor metode. Aceste elemente trebuie verificate și controlate.

- Exemple:
  - O aplicație web necesită **conectarea** utilizatorului;
  - O **extragere** din stivă funcționează dacă există elemente în stivă;
  - etc.

Reference

- Aceste elemente sunt numite **precondiții** sau **condiții preliminare**.
- Condiții preliminare pentru ca metoda să funcționeze în mod normal.
- Aceste teste sunt efectuate folosind dubluri de test (stub, fake, dummy, mock).

Existence

- **“does some given thing exist?”**

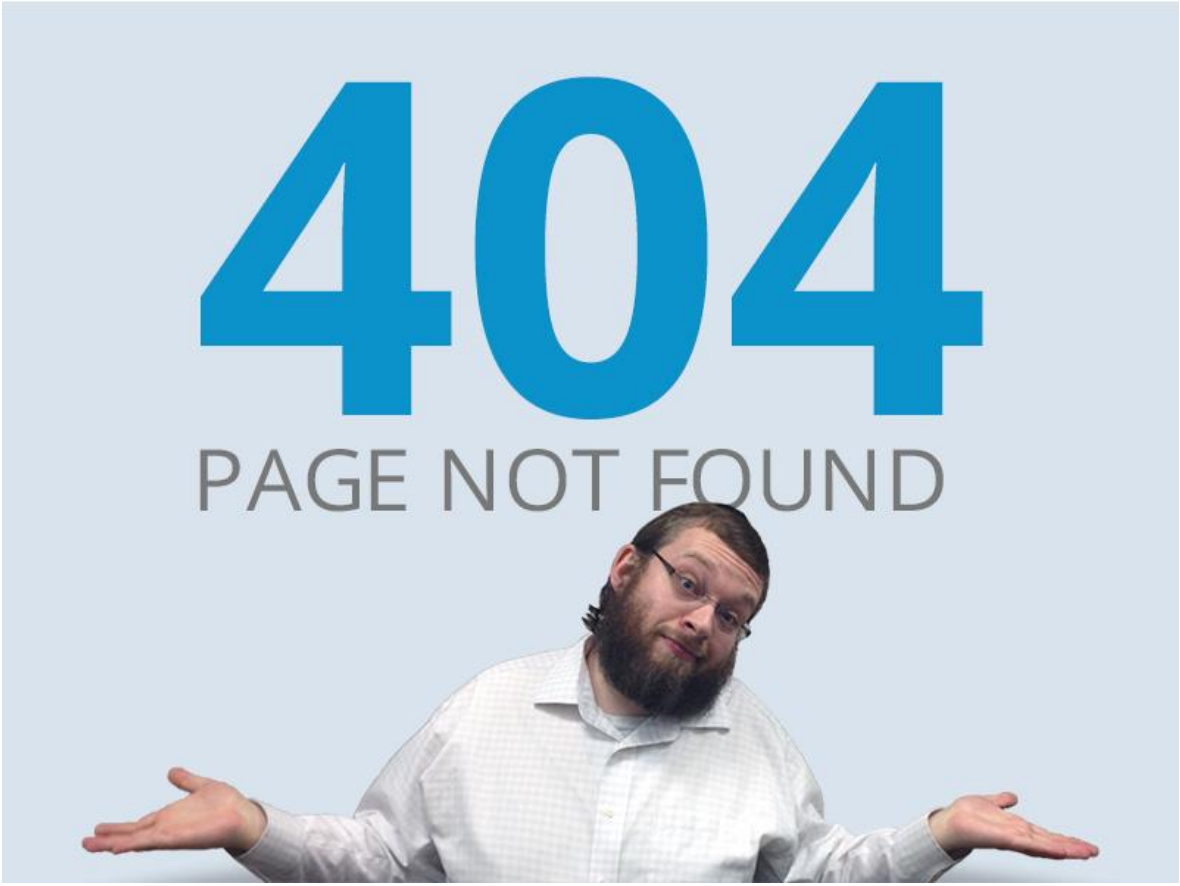
- Trebuie să ne întrebăm ce se întâmplă cu metoda dacă un parametru nu există, dacă este nul sau dacă este 0.

- De asemenea, pentru sistemele software care funcționează cu fișiere sau cu conexiune la internet, este necesar să se verifice existența acestor fișiere sau disponibilitatea conexiunii la internet. În caz contrar, aplicația nu trebuie să dea eroare, ci trebuie să se comporte normal cu avertizarea utilizatorului de problema întâmpinată.

Existence

- **Make sure your method can stand up to nothing.**

- Este asemănătoare cu condiția de eroare din **Right-BICEP**.



Cardinality

- **0-1-*n* Rule**

- Este similar cu testele de existență (**Existence**) și testele privind intervalul (**Range**).

- Trebuie să verificăm dacă metoda/lista/colecția are 0 elemente, 1 element sau elemente n.

- Dacă funcționează pentru 2, 3 sau 4 elemente, se consideră că va funcționa pentru mai multe elemente, însă nu trebuie să uităm de testul de **Boundary** superior.

Time

- Este similar cu testul de performanță din Right-BICEP.
- De asemenea, poate fi testat dacă șablonul de apeluri este respectat. Similar cu design pattern-ul **Template**.
- De exemplu, pentru a apela metoda *logout()*, trebuie mai întâi să apelăm metoda de *conectare()*.

CORRECT - Questions

- **C –** Conformitatea formatului (**C**onformance);
- **O –** Ordinea (**O**rder);
- **R –** Intervalul (**R**ange);
- **R –** Referințe externe (**R**eferences);
- **E –** Existența obiectelor sau a rezultatelor (**E**xistence);
- **C –** Cardinalitatea rezultatelor (**C**ardinality)
- **T –** Timpul (**T**ime).

F.I.R.S.T

- ***“ Pentru ca testele unitare să fie utile și eficiente pentru echipa de programare, trebuie să vă amintiți să le faceți FIRST.”***

F.I.R.S.T

- **F**ast
- **I**solated/Independent
- **R**epeatable
- **S**elf-Validating
- **T**imely

Fast

- Testul dezvoltat ar trebui să fie rapid, deoarece dacă avem prea multe teste, nu trebuie să așteptăm prea mult timp când le executăm.

Isolated

- **Single responsibility (SOLID)**

- "Each unit test should have a single reason to fail."

Isolated

- Atunci când un test eșuează, dezvoltatorul nu trebuie să facă debug pentru a identifica ce este greșit și unde este problema.

- Testul ar trebui să fie izolat și să spună exact unde este problema și ce problemă există.

Repeatable

- Rezultatele obținute ar trebui să fie identice indiferent de numărul rulări ale acestor teste.

- Testele ar trebui să se desfășoare în mod repetat, fără alte intervenții.

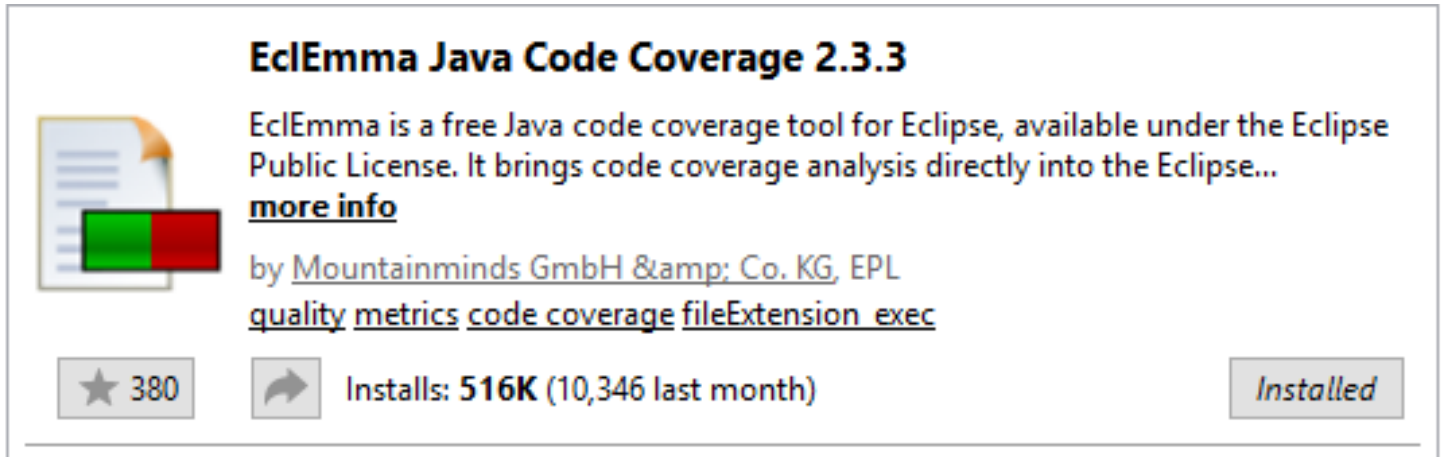
Self-Validating

- Încrederea în testele implementate.
- În cazul în care testele trec, dezvoltatorul ar trebui să aibă mare încredere că codul este corect și fără erori.
- Dacă un test nu reușește să treacă, dezvoltatorul trebuie să aibă încredere în faptul că metoda trebuie îmbunătățită ci nu să considere ca testul este greșit.

Timely

- Când trebuie să punem în aplicare testele pentru metoda noastră?
- Când considerăm că am făcut toate testele?

EclEmma – Code Coverage



EclEmma – Code Coverage

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Seminar11Testare	83.6 %	353	69	422
src	83.6 %	353	69	422
testing	89.8 %	237	27	264
clase	82.3 %	116	25	141
Persoana.java	81.8 %	90	20	110
PersoanaMock.java	62.5 %	5	3	8
Familie.java	91.3 %	21	2	23
program	0.0 %	0	17	17