CURS 04.
NIVELURI DE TESTARE

Verificarea și validarea sistemelor soft
[21 Martie 2023]

Lector dr. Camelia Chisăliță-Creţu
Universitatea Babeş-Bolyai

Conţinut

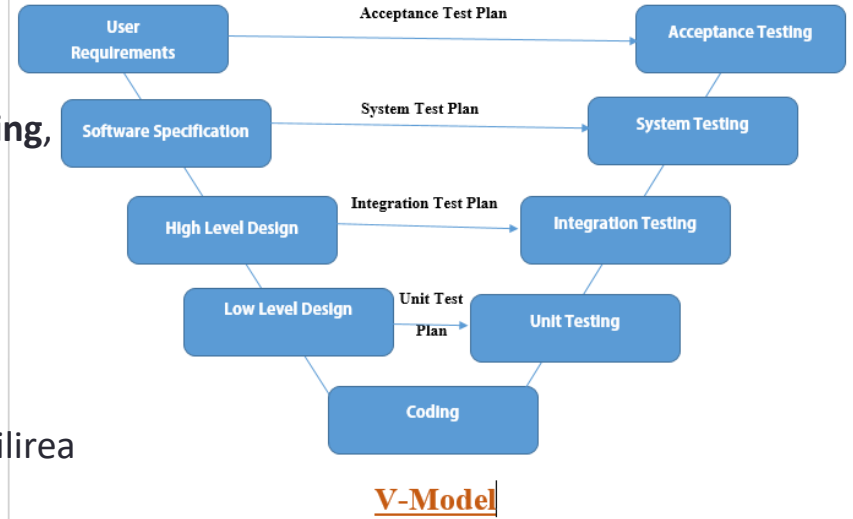
- **Niveluri de testare**
  - Definiţie. Clasificare
- **Testare unitară**
  - Definiţie. Motivație. Caracteristici
  - Proiectarea cazurilor de testare
  - Tipuri de bug-uri identificate
  - Reguli generale de aplicare
- **Testare de integrare**
  - Definiţie. Motivație. Clasificare
  - Integrare non-incrementală. Integrare incrementală. Integrare mixtă
  - Compararea strategiilor de integrare
  - Testarea interfeţei modulelor. Definiţie. Clasificare
  - Tipuri de bug-uri identificate
  - Exemplu
- **Testare de sistem**
  - Definiţie. Caracteristici
  - Testare funcţională
  - Testare non-funcţională
- **Testare de acceptare**
  - Definiţie. Caracteristici. Etape de realizare
  - Clasificare
  - Alpha Testing. Beta Testing
  - Alpha Testing vs Beta Testing
  - Alte tipuri de testare de acceptare
  - Dificultăţi de testare
- **Nivel de testare vs. Tip de testare**
  - Tip de testare. Nivel de testare. Definiţie
  - Obiective de testare. Exemple
  - Retestare. Definiţie
  - Testare de regresie. Definiţie
  - Retestare vs Testare de regresie
- **Pentru examen...**
- **Bibliografie**

NIVELURI DE TESTARE

Definiţie. Clasificare

Nivel de testare. Definiţie. Clasificare

- **nivel de testare** (*engl.* **testing level**):
  - o serie de activităţi de testare asociate unei etape din procesul de dezvoltare a produsului soft;
- clasificare:
  - **testare unitară / testare de modul** (*engl.* **unit testing, module testing**);
    - etapa: implementare/codificare;
  - **testare de integrare** (*engl.* **integration testing**);
    - etapa: proiectare;
  - **testare de sistem** (*engl.* **system testing**);
    - etapa: specificarea cerinţelor sistemului = stabilirea obiectivelor de realizat;
  - **testare de acceptare** (*engl.* **acceptance testing**);
    - etapa: descrierea cerinţelor utilizatorului.



TESTARE UNITARĂ

Definiţie. Motivație. Caracteristici
Proiectarea cazurilor de testare
Tipuri de bug-uri identificate
Reguli generale de aplicare

Testare unitară. Definiție. Motivație. Etape

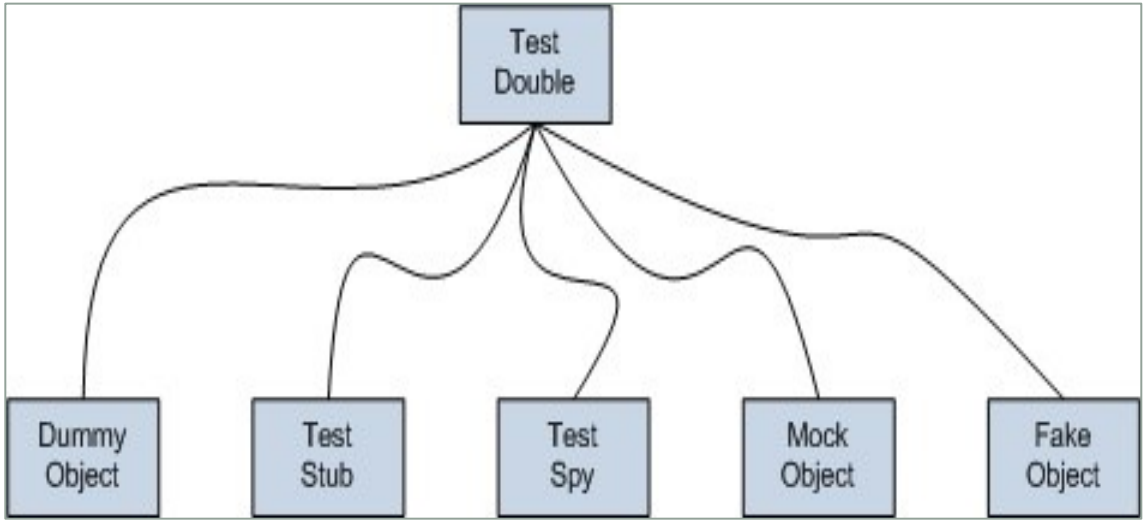
- **testare unitară/ testare de modul** (*engl.* **unit testing, module testing**) :
  - testarea individuală a unor unități separate dintr-un sistem software (funcție, procedură, clasă, metodă);

- **motivație:**
  - gestionarea eficientă a modulelor sistemului – mai întâi se testează modulele;
  - proces de depanare eficient – aplicat la nivel de modul;
  - permite paralelizarea procesului de testare – testare simultană pentru mai multe module.

- **etape:**
  - contextul de testare;
  - proiectarea cazurilor de testare;
  - execuția cazurilor de testare și evalurea rezultatelor testării.

Testare unitară. Tipuri de obiecte (1)

- la nivelul testării unitare se folosesc diferiţi termeni pentru a indica obiecte, stări sau caracteristici care apar la proiectarea cazurilor de testare;
- literatura de specialitate indică abordări diferite în descrierea obiectelor folosite, denumite generic **Test Doubles**, sugerând funcţionarea obiectelor utilizate în realitate [MeszarosFowler2006];

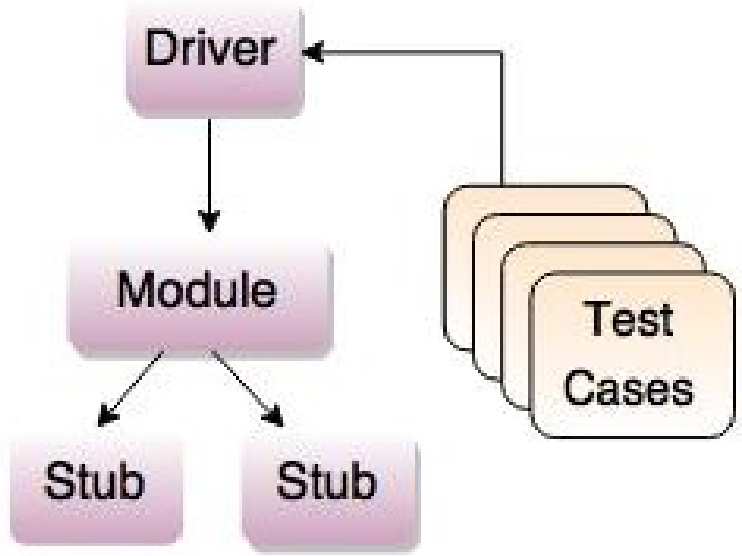


Testare unitară. Tipuri de obiecte (2)

- tipuri de obiecte utilizate de tool-uri:
  - **Dummy** – obiecte care sunt transmise ca parametri dar care **nu sunt folosite de metodele apelate**;
    - e.g., diverşi parametri precizaţi doar pentru a respecta signatura metodei apelate;
  - **Fake** – obiecte cu **implementări funcţionale**/utilizabile, dar **simpliste**, care nu sunt adecvate pentru a fi incluse în livrabilul către client;
    - e.g., o colecţie de date in-memory;
  - **Stubs** – obiecte sau metode ale unor obiecte care **furnizează rezultate prestabilite** atunci când sunt apelate în cadrul unui test; nu au altă utilitate în afara contextului testării unde au fost definite;
  - **Spies** – obiecte ***stub*** care pot păstra/reţine informaţii referitoare la modul în care au fost folosite;
    - e.g., un serviciu pentru e-mail care reţine numărul de mesaje transmise;
  - **Mocks** – obiecte pentru care s-a stabilit **un anumit comportament** (behavior expectations) şi sunt utilizate pentru a observa interacţiunea cu obiectul supus testării.

Testare unitară. Context de testare

- tipuri de module:
  - **driver** (*engl.* **driver**):
    - modul apelant al modulului testat, care furnizează datele de intrare modulului testat;
  - **stub** (*engl.* **dummy subprogram**):
    - modul apelat în cadrul modulului testat, înlocuiește modulul apelat în contextul real;
      - arată că modulul testat apelează un modul subordonat;
      - returnează o valoare prestabilită în modulul testat care să îi permită să își continue execuția;
- pentru fiecare modul testat trebuie să existe un driver dedicat și mai multe module stub, dacă este necesar;
- modulele driver și stub sunt create suplimentar (*engl.* **overhead**) și **nu** sunt livrate împreună cu produsul final;



Testare unitară. Proiectarea cazurilor de testare

- **caracteristici:**
  - aplică tehnici de testare black-box, grey-box și white-box;
  - folosește documentele care conţin specificația modulelor;
- informații necesare proiectării unui caz de testare pentru un modul:
  - specificația modulului;
  - codul sursă pentru modul;
- **tipuri de bug-uri identificate:**
  - **testarea black-box:**
    - nerespectarea condițiilor impuse în specificaţii;
    - înțelegerea greșită a specificațiilor;
  - **testarea white-box:**
    - înțelegerea greșită a precedenței operatorilor;
    - inițializare incorectă;
    - lipsa acurateței/preciziei;
    - operații aplicate eronat.

Testare unitară. Reguli generale de aplicare (1)

1. număr de paşi de executat;
2. execuţie: planificare şi durată;
3. consecvenţă;
4. atomicitate;
5. responsabilitate unică;
6. izolarea testelor;
7. izolarea de mediul de execuţie;
8. izolarea claselor;
9. automatizare completă;
10. *self-descriptive;*
11. fără condiţii logice;
12. fără bucle;
13. fără tratarea excepţiilor;
14. utilizarea instrucţiunilor assert;
15. utilizarea de mesaje sugestive;
16. fără testare în codul sursă livrat;
17. separarea pe module şi niveluri ale arhitecturii;
18. gruparea testelor în funcţie de tipul de testare.

Testare unitară. Reguli generale de aplicare (2)

- **număr de paşi de executat:**
  - 3-5 paşi:
    1. **set up;**
    2. date de intrare;
    3. apelarea metodei testate;
    4. verificarea rezultatului (`assert`);
    5. **tear down;**

Testare unitară. Reguli generale de aplicare (3)

- **timp de execuţie:**
  - frecvenţa de execuţie:
    - testare după implementare (*engl.* **test after development**): de câteva ori pe zi;
    - dezvoltare dirijată de testare (*engl.* **test driven development**): de câteva ori pe oră;
    - execuţie după salvare, IDE (*engl.* **IDE runs tests after save**): la fiecare câteva minute;
  - mod de execuţie (singular sau suită de teste):
    - timp de execuţie pentru 10 teste = timp de execuţie 1 test x 10;
    - **un test care se execută greu, încetineşte întreaga suită de teste;**
  - valori medii:
    - un test < 200 milisecunde;
    - o suită cu număr de redus de teste < 10 secunde;
    - o suită cu număr consistent de teste < 10 minute.

Testare unitară. Reguli generale de aplicare (4)

- **consecvenţă:**
  - execuţia repetată a aceluiaşi test ar trebui să returneze în mod repetat acelaşi rezultat, dacă nu au avut loc modificări asupra codului sursă;

- cod sursă problematic:
  - `Date current Date = new Date();`
  - `Int value = random.nextInt(100);`

- soluţii:
  - utilizarea obiectelor dummy, mock, stub, fake;
  - injectarea dependenţelor.

Testare unitară. Reguli generale de aplicare (5)

- **atomicitate:**
  - rezultate posibile ale execuţiei unui test:
    - **passed**;
    - **failed**;
  - nu există teste care „au trecut” doar parţial;
  - **dacă un punct de execuţie este failed ==> întregul test este failed.**

Testare unitară. Reguli generale de aplicare (6)

- **acțiune/ reponsabilitate unică** (*engl.* **single responsibility**):
  - un caz de testare investighează un singur scenariu de execuţie;

- se testează comportamentul metodei:
  - **o metodă, mai multe utilizări (comportamente)**

    ==> mai multe teste şi cel puţin un test pentru fiecare comportament;
    - mai multe instrucţiuni `assert` în acelaşi test – doar dacă verifică acelaşi comportament;
  - **o utilizare (comportament) descrisă prin folosirea mai multor metode**

    ==> un singur test;
    - E.g.: o metodă care apelează metode private/ protected/ publice, simple, e.g., getters, setters, constructori simpli.

Testare unitară. Reguli generale de aplicare (7)

- **acțiune/ reponsabilitate unică** (*engl.* **single responsibility**):
  - **o metodă, mai multe utilizări (comportamente) ==> mai multe teste;**

```
testMethod(){
    ...
    assertTrue(behaviour1);
    assertTrue(behaviour2);
    assertTrue(behaviour3);
}
```

```
testMethodCheckBehaviour1(){
    ...
    assertTrue(behaviour1);
}
testMethodCheckBehaviour2(){
    ...
    assertTrue(behaviour2);
}
testMethodCheckBehaviour3(){
    ...
    assertTrue(behaviour3);
}
```

Testare unitară. Reguli generale de aplicare (8)

- **acțiune/ reponsabilitate unică** (*engl.* **single responsibility**):
  - **o utilizare (comportament) descrisă prin folosirea mai multor metode ==> un singur test;**
    - comportament 1 = condition1 + condition2 + condition3 ;
    - comportament 2 = condition4 + condition5;

```
testMethodCheckBehaviours(){
...
assertTrue(condition1);
assertTrue(condition2);
assertTrue(condition3);
...
assertTrue(condition4);
assertTrue(condition5);
}
```

```
testMethodCheckBehaviour1(){
    ...
    assertTrue(condition1);
    assertTrue(condition2);
    assertTrue(condition3);
}
testMethodCheckBehaviour2(){
    ...
    assertTrue(condition4);
    assertTrue(condition5);
}
```

Testare unitară. Reguli generale de aplicare (9)

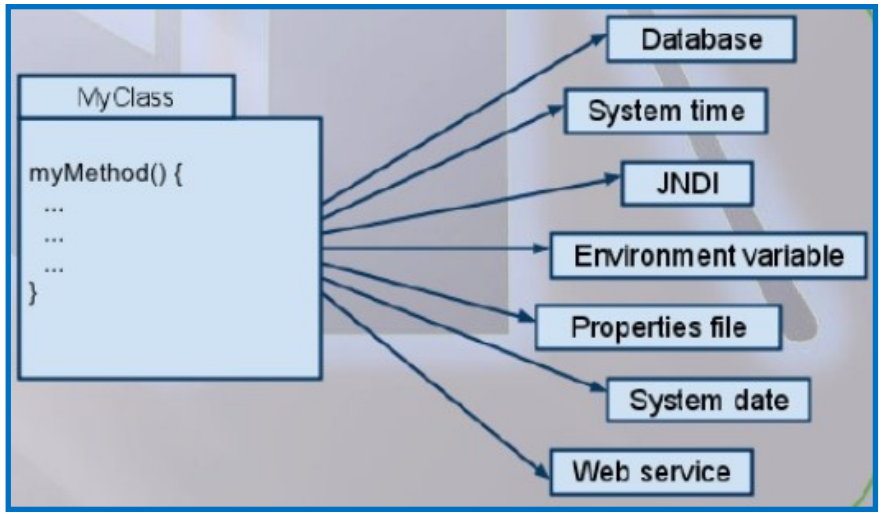
- **izolarea testelor (unele de altele):**
  - testele trebuie să fie independente unele de altele.
    - **la execuţii diferite (la momente de timp diferite, în ordine diferită) ale aceluiaşi test trebuie să se obţină aceleaşi rezultate.**
    - **nu** se partajează starea/ contextul de execuţie între teste;
    - variabile folosite la testare, e.g., JUnit – *variabile partajate sau nu între teste*: `@Before`, `@BeforeClass`.

Testare unitară. Reguli generale de aplicare (10)

- **izolarea testelor de mediul de execuţie (context):**
  - testele trebuie izolate de influenţele mediului de execuţie;
  - E.g.,
    - baze de date;
    - apelarea serviciilor web;
    - Java Naming and Directory Interface (JNDI);
    - variabile de mediu definite local;
    - fişiere de proprietăţi;
    - configurările de dată şi oră ale sistemului.

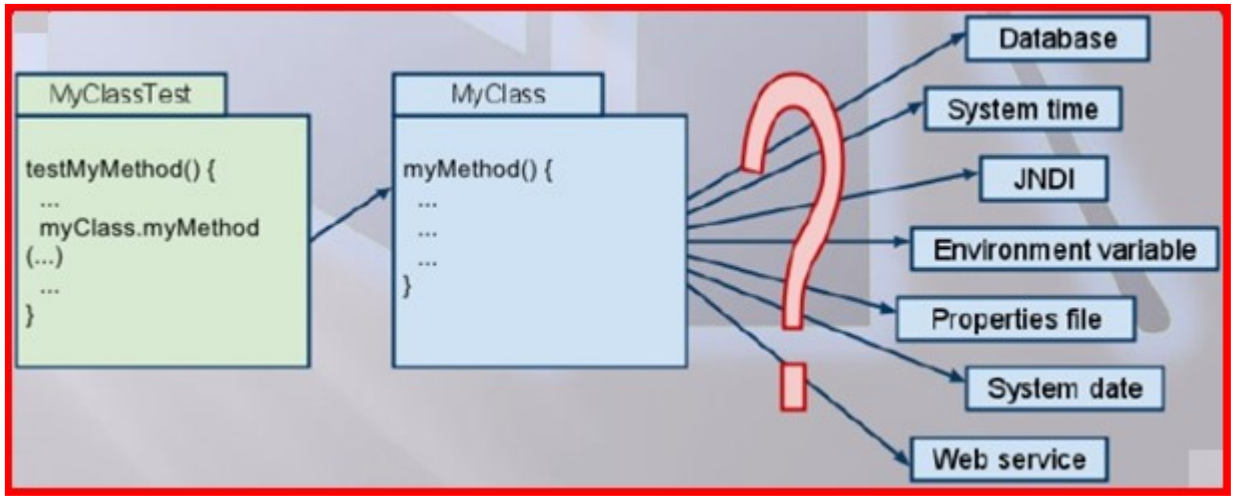
Testare unitară. Reguli generale de aplicare (11)

- **izolarea testelor de mediul de execuţie (context):**
  - codul sursă livrabil (*engl.* **production code**) foloseşte *mediul de execuţie*;



Testare unitară. Reguli generale de aplicare (12)

- **izolarea testelor de mediul de execuţie (context):**
  - la testare se foloseşte un **mediu de testare**, care nu este întotdeauna identic cu **mediul de execuţie** de la dezvoltarea codului sursă sau de la client;

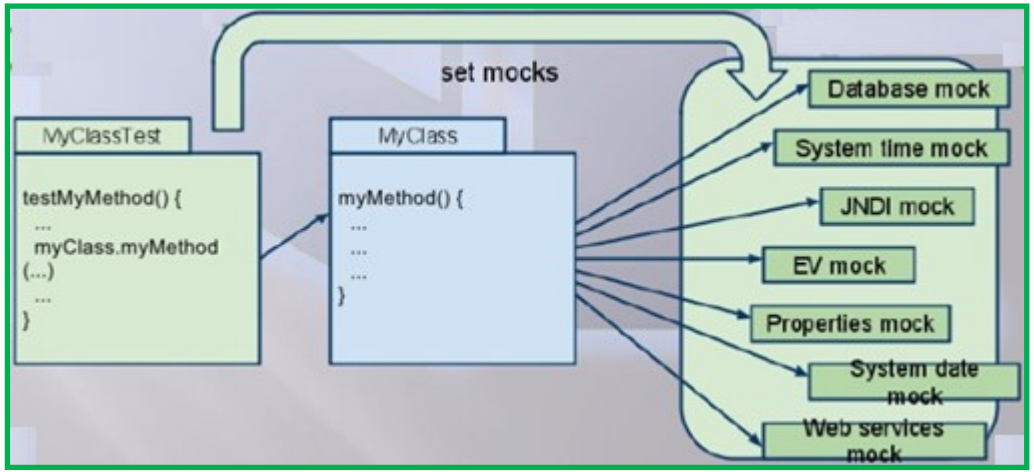


Testare unitară. Reguli generale de aplicare (13)

- **izolarea testelor de mediul de execuţie (context):**

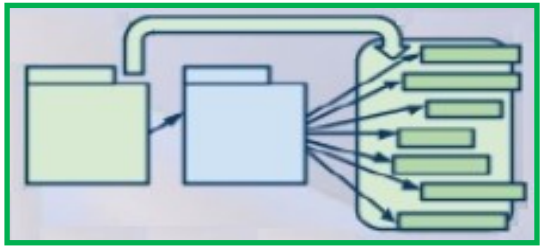
- **soluţie:** se utilizează obiecte mock care indică un anumit mediu de utilizare;

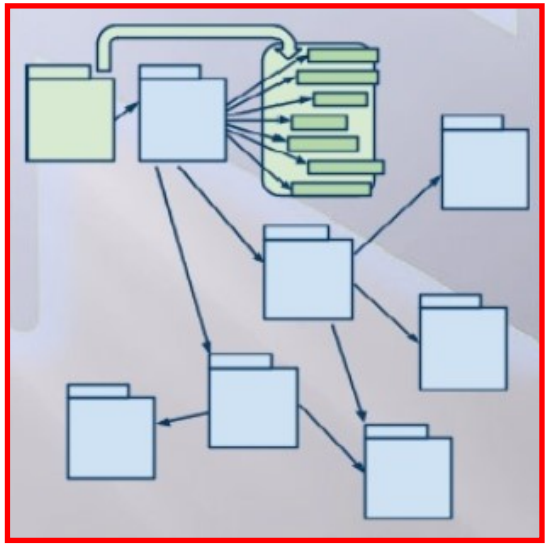
- Avantaje:
  - rapiditate;
  - uşor de dezvoltat;
  - reutilizabilitate;
- E.g.: biblioteci Java:
  - EasyMock;
  - Jmock;
  - Mockito.

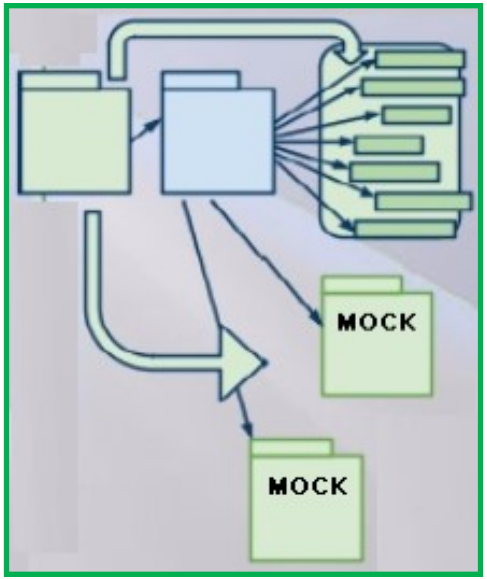


Testare unitară. Reguli generale de aplicare (14)

- **izolarea claselor:**
- clase fără dependenţe
- clase cu dependenţe
  - **soluţie: dependenţe dummy, mock, stub, fake;**







Testare unitară. Reguli generale de aplicare (15)

- **izolarea claselor:**

- dificil de realizat dacă avem un cod sursă greu de testat;
- se recomandă:
  - folosirea **metodelor Factory** sau **injectarea dependenţelor**, în locul apelului constructorilor în cadrul metodelor;
  - **folosirea interfeţelor**.

Testare unitară. Reguli generale de aplicare (16)

- **automatizare completă:**
  - fără paşi executaţi manual pe durata testării.

- se automatizează:
  - execuţia testelor;
  - colectarea rezultatelor testării;
  - stabilirea rezultatului testării (passed/ failed);
  - trasmiterea rezultatelor testării prin: e-mail, IDE integration, etc;

Testare unitară. Reguli generale de aplicare (17)

- **self-descriptive:**
  - la nivelul testării unitare un test reprezintă:
    - documentaţie la nivel de dezvoltare;
    - metodă de specificare care reflectă versiunea actualizată a cerinţelor;
- **caracteristici:**
  - un test trebuie să fie uşor de citit şi înţeles;
    - denumirile variabilelor, metodelor şi claselor trebuie să fie *self-descriptive*;
    - **nu trebuie să conţină condiţii logice sau iteraţii;**
  - numele cazurilor de testare trebuie să fie sugestive pentru a indica condiția de succes sau eșec:
    - `public void canMakeReservation();`
    - `public void cannotAddNewBook();`

Testare unitară. Reguli generale de aplicare (18)

- **fără instrucţiuni logice:**
  - un caz de testare **nu** ar trebui să conţină instrucţiunea `if` sau `switch`;

- **nu** există incertitudini legate de:
  - *datele de intrare* ==> toate datele de intrare sunt cunoscute;
  - *comportamentul aşteptat* ==> comportamentul metodei este predictibil;
  - *datele de ieşire* ==> rezultatele aşteptate ar trebui să fie bine definite.

Testare unitară. Reguli generale de aplicare (19)

- **fără instrucţiuni logice:**
  - mai multe condiţii trebuie să se regăsească în cazuri de testare distincte, nu în acelaşi test, fiind tratate ca şi ramificaţii ale instrucţiunilor alternative, i.e., `if`, `switch`;

```
testMethodBeforeOrAfter(){
    ...
  if (before) {
   assertTrue(behaviour1);
  } else if (after) {
   assertTrue(behaviour2);
  } else { //now
   assertTrue(behaviour3);
  }
}
```

```
testMethodBefore(){
    before = true;
    assertTrue(behaviour1);
}
testMethodAfter(){
    after = true;
    assertTrue(behaviour2);
}
testMethodNow(){
    before = false;
    after = false;
    assertTrue(behaviour3);
}
```

Testare unitară. Reguli generale de aplicare (20)

- **fără instrucţiuni iterative** (i.e., `while`, `do while`, `for`):
  - scenarii tipice pentru iteraţii:
    - **câteva sute de iteraţii:**
      - dacă sunt necesare câteva sute de iteraţii, atunci testul este destul de complicat şi **este necesar să fie simplificat**;
    - **câteva iteraţii:**
      - se recomandă refactorizarea codului care trebuie să se repete într-o metodă, care să fie apelată ulterior explicit de câte ori este necesar;
- **număr de iteraţii necunoscut:**
  - numărul de iteraţii este greu de evitat, indicând faptul că e posibil ca ***testul să fie incorect*** şi se recomandă specificarea mai clară a datelor de intrare;

```
testThrowingMyException(){
    try {
      myMethod(param);
      fail("MyException expected");
    } catch(MyException ex) {
      //OK
    }
}
```

Testare unitară. Reguli generale de aplicare (21)

- **instrucţiunea `assert`:**

- se recomandă:
  - utilizarea instrucţiunilor `assert` disponibile în platforma de testare;
  - crearea propriilor aserţiuni pentru verificarea condiţiilor complexe care se repetă în diferite teste;
  - reutilizarea propriilor metode asertive;
  - utilizarea aserţiunilor în cadrul buclelor.

Testare unitară. Reguli generale de aplicare (22)

- **mesaje sugestive în instrucţiunea `assert`:**
  - prin citirea mesajului din aserţiune se poate recunoaşte uşor problema care a determinat eşuarea testului;

- **avantaje:**
  - permit îmbunătăţirea documentaţiei codului sursă;
  - oferă informaţii asupra problemei dacă testul a eşuat (failed).

Testare unitară. Reguli generale de aplicare (23)

- **codul sursă nu include teste:**

- se recomandă:
  - separarea testelor de codul sursă livrat;
  - clasele să **nu** definească metode şi/sau atribute care sunt folosite doar la testare.

Testare unitară. Reguli generale de aplicare (24)

- **gruparea testelor în funcţie de tipul de testare:**

- după modulul testat:
  - organizarea testelor în pachete corespunzătoare modulelor testate;
  - utilizarea unei abordări ierarhizate;
  - scăderea timpului de execuţie pentru suitele de teste prin împărţirea acestora în suite de dimensiuni mai mici, i.e., suitele de mici dimensiuni pot fi executate mai des/frecvent;
- după tip:
  - scopul / obiectivele testării;
  - frecvenţa de execuţie;
  - momentul execuţiei suitei;
  - acţiunea în caz de eşec.

TESTARE DE INTEGRARE

Definiţie. Motivație. Clasificare

Integrare non-incrementală. Integrare incrementală. Integrare mixtă

Compararea strategiilor de integrare

Testarea interfeţei modulelor. Definiţie. Clasificare

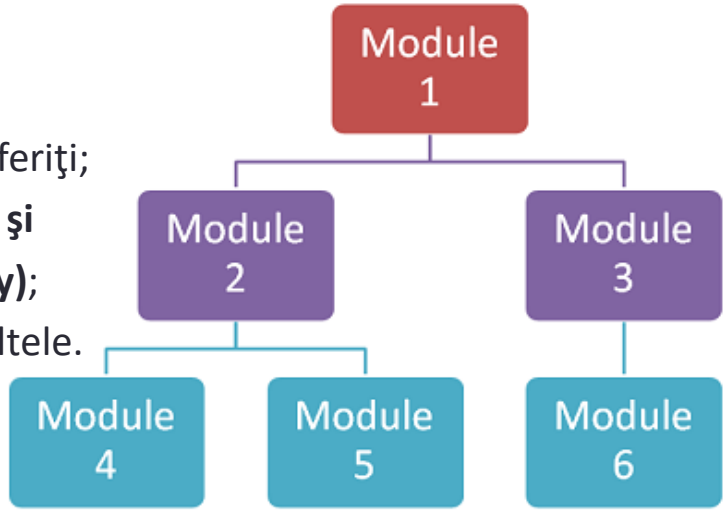
Tipuri de bug-uri identificate

Exemplu

Testare de integrare. Definiţie. Motivaţie

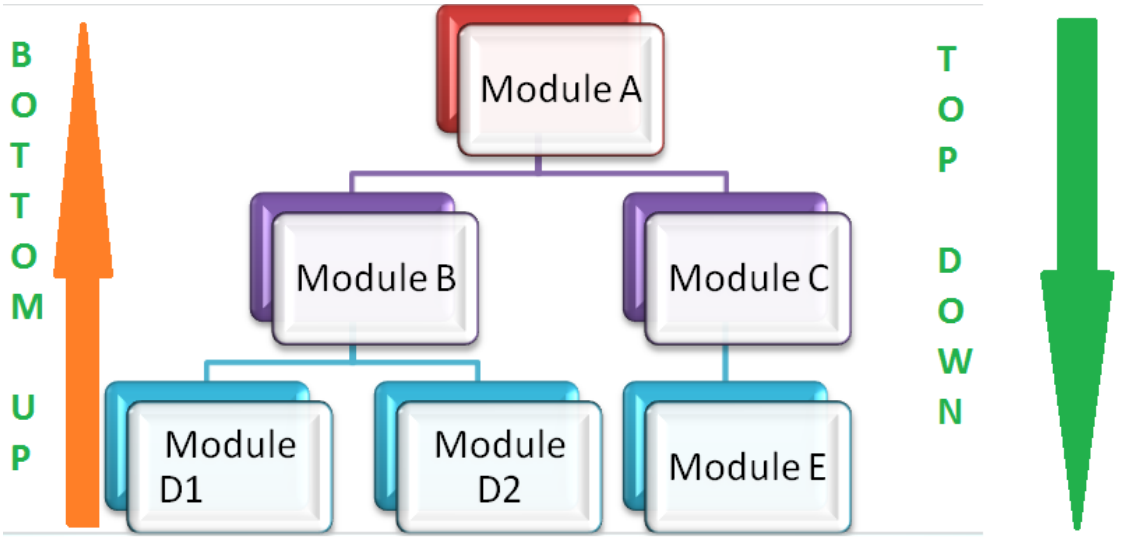
- **testare de integrare** (*engl.* **integration testing**):
  - nivel de testare în care modulele individuale se combină şi se testează ca un grup;
  - permite construirea structurii programului pe măsură ce este testat pentru a **identifica erorile la nivelul interfeţei dintre module**;

- **motivaţie:**
  - module diferite sunt implementate de programatori diferiţi;
  - testarea unitară se desfăşoară într-un **mediu controlat şi izolat**, folosind entităţi **driver** şi **stub (mock, fake, dummy)**;
  - unele module pot genera mai multe defecţiuni decât altele.

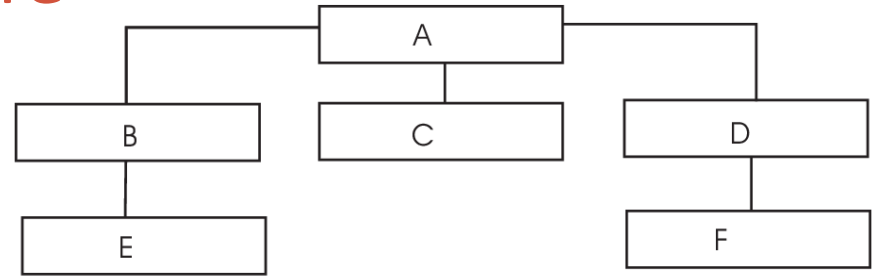


Testare de integrare. Clasificare

- abordări de integrare/ clasificare:
  - non-incrementală
    - **big-bang;**
  - incrementale
    - **top-down;**
    - **bottom-up;**
  - mixtă
    - **sandwich.**



Integrarea Big-bang. Descriere



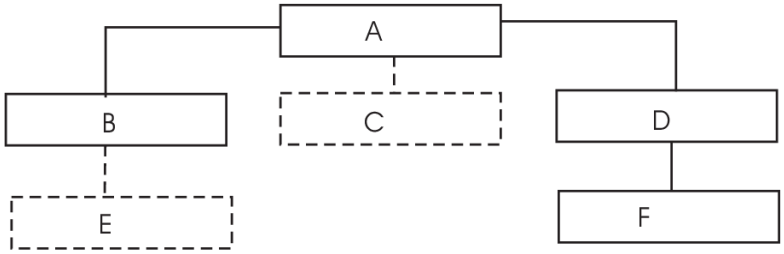
- Procesul de integrare Big-bang:
  1. testare unitară pentru fiecare modul, folosind:
     - un modul driver;
     - câteva module stub;
  2. se combină simultan modulele pentru construirea funcţionalităţii programului;

- **dezavantaje:**
  - necesită un volum de muncă ridicat;
  - depanare dificilă – este dificil de izolat modulul care a determinat defecţiunea;
  - dacă programul este complex – este dificil de urmărit modul în care s-au executat funcţionalităţile.
- **avantaje:**
  - dezvoltare şi testare unitară paralelă.

Integrare incrementală Top-down. Descriere

- **integrare incrementală Top-down** (*engl.* **Top-down incremental integration**):
  - permite construirea şi testarea programului adăugând module noi pe parcurs;
  - integrarea modulelor se face de sus în jos, de la modulul principal, modulele subordonate sunt încorporate succesiv;
  - defectele sunt uşor de izolat şi corectat;
  - testarea este aplicată sistematic.

- tipuri de integrare Top-down:
  - **în adâncime** (*engl.* **depth-first integration**);
  - **pe niveluri** (*engl.* **breadth-first integration**);

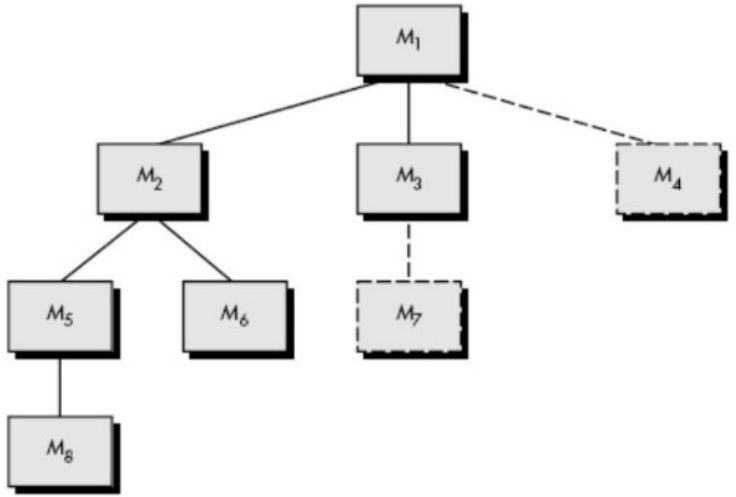


- **integrarea depth-first**
  - integrează toate componentele de pe o ramificaţie majoră a arhitecturii aplicaţiei;
  - ordinea de integrare a ramificaţiilor depinde de caracteristicile aplicaţiei;

- **integrarea breadth-first**
  - integrează toate componentele care se află pe nivelul direct subordonat, i.e., pe orizontală;

Integrare incrementală Top-down. Algoritm

- după testarea unitară a modulului principal folosind **stub-uri** pentru modulele imediat subordonate, procesul de **integrare Top-down** constă în:
  1. în funcţie de tipul de integrare ales (**depth-first/ breadth-first**), se înlocuieşte un stub cu un modul real;
  2. se testează cu modulul subordonat concret care a fost integrat;
  3. pentru integrarea unui nou modul se repetă paşii 1 şi 2.

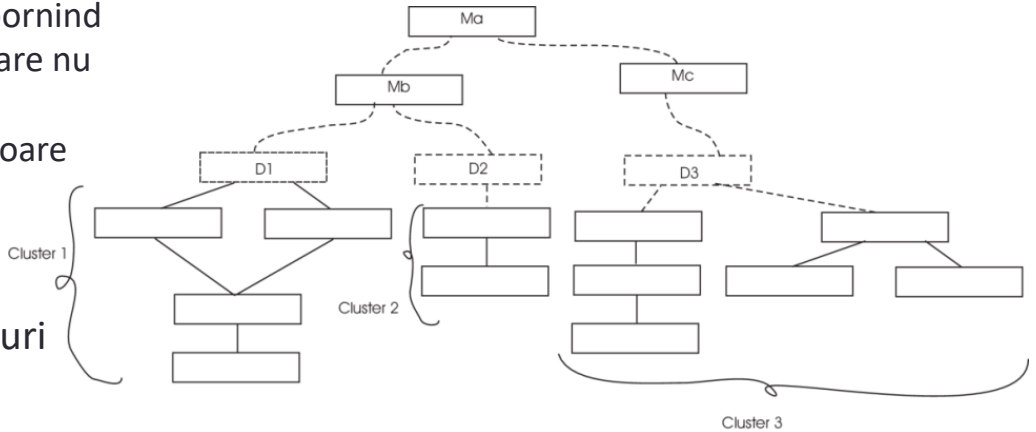


- **driver** = modulul principal; nu se folosesc alte drivere;
- **stub** = înlocuieşte un modul direct subordonat;

- **E.g.,**
  - **depth-first:** M1, M2, M5, M8, M6, M3, M7, M4;
  - **breadth-first:** M1, M2, M3, M4, M5, M6, M7, M8.

Integrare incrementală Bottom-up. Descriere

- **integrare incrementală Bottom-up (*engl.* Botton-up incremental integration):**
  - permite construirea şi testarea programului pornind de la modulele atomice, i.e., componentele care nu au module dependente;
  - defectele modulelor aflate pe nivelurile inferioare sunt uşor de izolat şi corectat.

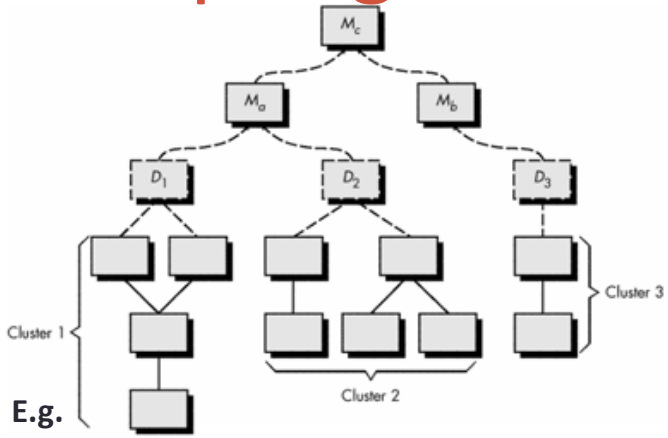


- modulele terminale se pot organiza în grupuri (*engl.* **clusters**);

- **driver** = se folosesc doar pentru modulele terminale (care nu au module subordonate) sau grupurile de module;
- **stub** = nu se folosesc;

Integrare incrementală Bottom-up. Algoritm

- Procesul de integrare Bottom-up:
  - pentru toate modulele terminale sau clusteri de module se descriu drivere şi se testează;
  - *module terminale:*
    - se înlocuieşte driver-ul cu modulul de pe nivelul imediat superior şi se testează, continuând integrarea modulelor spre partea superioară a structurii programului;
  - *clusteri de module:*
    - se înlătură driver-ul, iar clusterul de module se combină cu alte module continuând integrarea spre partea superioară a structurii programului.

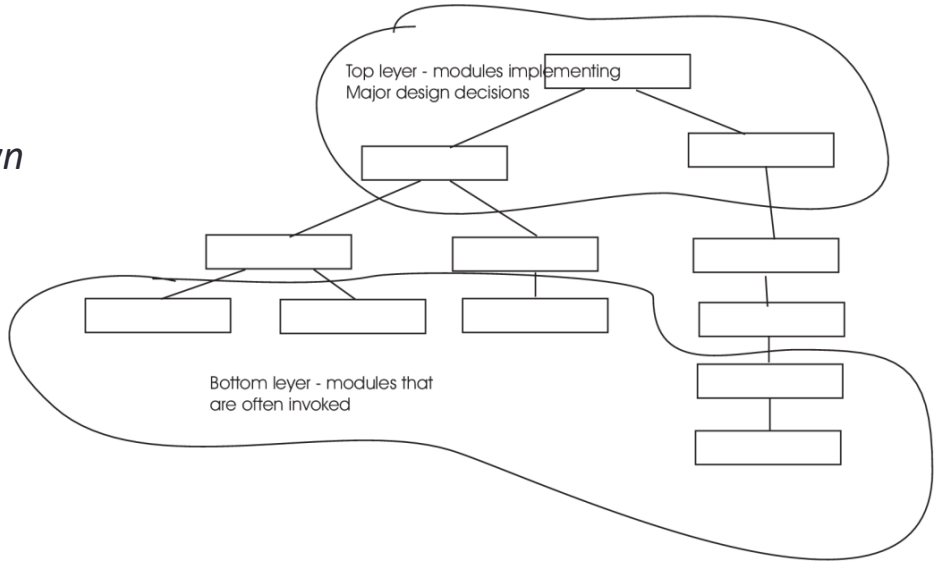


- **E.g.**
  - componentele se combină în clusterii 1 , 2 şi 3;
  - clusterii se testează folosind driver-ele D1, D2 şi D3;
  - se înlătură D1 şi D2, iar clusterul 1 şi clusterul 2 sunt integraţi în Ma;
  - similar, D3 este înlăturat iar clusterul 3 este integrat cu modulul Mb.
  - Ma şi Mb se vor integra în Mc.

Integrare Sandwich. Descriere

- **integrare sandwich** (*engl.* **sandwich integration**):
  - permite construirea şi testarea programului combinând abordările de integrare *top-down* şi *bottom-up*;

- Procesul de integrare Sandwich:
  1. **modulele terminale** (bottom-layer):
     - integrare *bottom-up*;
  2. **modulul principal** (top-layer):
     - integrare *top-down*;
  3. **celelalte module** (middle-layer):
     - integrare *big-bang*.



Testare de integrare. Compararea strategiilor de integrare

Criteriu	Big-bang	Top-down	Bottom-up	Sandwich
Integrare	Târzie	Timpurie	Timpurie	Timpurie
Programul final	Tarziu	Devreme	Târziu	Devreme
Driver	Da	Nu	Da	Da
Stub	Da	Da	Nu	Da
Paralelizare	Mare	Mică	Medie	Medie

Testarea interfeţei modulelor. Definiţie. Clasificare

- **testarea interfeţei modulelor** (*engl.* **interface integration testing**):
  - se realizează la integrarea mai multor module pentru obţinerea unor sisteme de dimeniuni mai mari;
- relevantă în dezvoltarea orientată pe obiecte, comportamentul acestora fiind descris prin intermediul interfeţelor;
- **obiective**: identificarea defectelor care pot apărea la utilizarea unei interfeţe a unui alt modul sau false presupuneri legate de interfaţa unui modul;

- **comunicarea între module** poate fi bazată pe:
  - **parametri**: la transmiterea datelor de la o metodă la alta;
  - **memorie partajată**: o zonă de memoria este partajată între module;
  - **comportament**: un sub-sistem încapsulează un set de metode care sunt apelate de alte sub-sisteme;
  - **mesaje**: un sub-sistem are nevoie de serviciile altor sub-sisteme;

Testarea interfeţei modulelor. Tipuri de bug-uri identificate

- **tipuri de bug-uri [NT2005, pagina 161]:**
  - utilizarea greşită a interfeţei (*engl.* **interface misuse**):
    - un modul apelează un alt modul eronat, e.g., parametrii sunt transmişi în ordinea greşită;
  - înţelegerea greşită a semnificaţiei interfeţei (*engl.* **interface misunderstanding**):
    - un modul face presupuneri greşite referitare la comportamentul unui alt modul apelat, e.g., se interpretează că mai mulţi parametri de acelaşi tip au altă semnificaţie;
  - greşeli de sincronizare (*engl.* **timing errors**):
    - modulul apelat şi modulul apelant folosesc unităţi de timp diferite, e.g., minute, secunde, milisecunde, iar informaţia obţinută îşi pierde acurateţea.

Testare de integrare. Reguli generale de aplicare

- se studiază arhitectura aplicaţiei pentru **identificarea modulelor critice**; aceste module se testează cu **prioritate**;
- testerul este familiarizat cu modulele care se integrează (arhitectura modulelor);
- se aplică o **strategie de integrare** care să permită atingerea **obiectivelor** testării;
- fiecare modul este testat înainte de a execuţia testelor de integrare, i.e., unit testing;
- se utilizează **documentaţia care descrie interacţiunile dintre fiecare două module** (interfeţele acestora şi modul de colaborare).

TESTARE DE SISTEM

Definiţie. Caracteristici

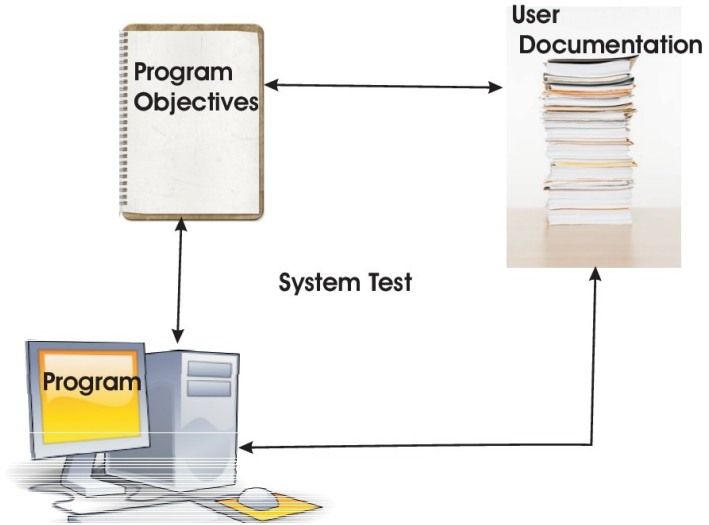
Testare funcţională

Testare non-funcţională

Testare de sistem. Definiţie. Caracteristici

- **testare de sistem** (*engl.* **system testing**):
  - verifică dacă produsul dezvoltat respectă obiectivele iniţiale;

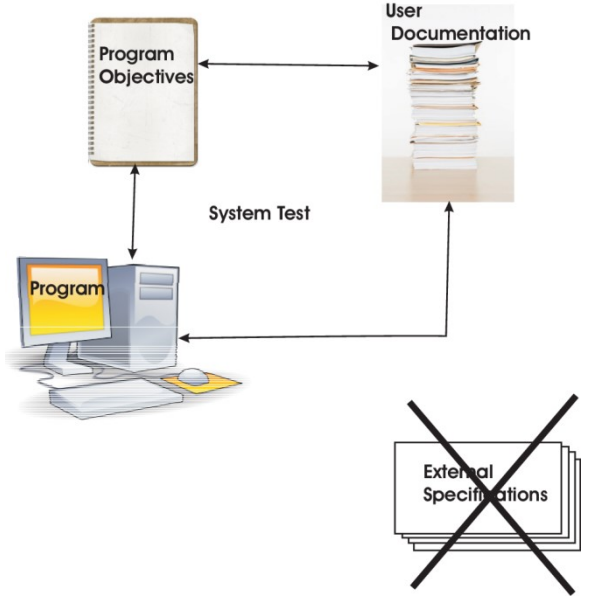
- elaborarea cazurilor de testare se bazează pe:
  - ***documentul cu specificaţiile sistemului?***
    - **nu,** deoarece pot apărea erori în procesul de transpunere a obiectivelor în specificaţii externe;
  - ***documentele cu obiectivele stabilite?***
    - **nu,** deoarece nu conţin descrierea exactă a interfeţelor externe ale programului;
    - obiectivele nu oferă informaţii cu privire la funcţionalitatea sistemului (interfeţe ale



- ***documentaţia de utilizare a produsului soft?***
  - ***da;*** proiectarea testării de sistem – pe baza analizei obiectivelor;
  - descrierea cazurilor de testare – pe baza analizei documentaţiei de utilizare.

Testare de sistem. Caracteristici

- **nu există metodologie pentru proiectarea cazurilor de testare în testarea de sistem;**
  - se proiectează două categorii de teste asociate:
    - cerinţele funcţionale
      ==> **testare funcţională;**
    - cerinţele non-funcţionale
      ==> **testare non-funcţională**
      (*engl.* **non-functional testing**);



- **la acest nivel de testare, proiectarea cazurilor de testare este determinată de obiectivele de testare şi de strategia aleasă, evidenţiind creativitatea şi experienţa.**

Testare de sistem. Testare funcţională a sistemului

- **testare funcţională a sistemului** (*engl.* **function/functional testing, use case testing**):
  - testarea cerinţelor descrise în specificaţiile sistemului;
  - proces care identifică neconcordanţele existente între comportamentul programului şi specificaţia acestuia, **din punctul de vedere al utilizatorului**;

- elaborarea cazurilor de testare se bazează pe **criteriul black-box**;
- foloseşte specificaţia sistemului.

Testare funcţională a sistemului. Proiectarea cazurilor de testare

- **caz de utilizare vs. caz de testare:**
  - cazurile de utilizare descriu fluxul de execuţie al unui proces în cadrul sistemului, folosind **scenarii** de utilizare frecventă;
  - **cazurile de testare sunt derivate din cazurile de utilizare** şi pot indica prezenţa unui bug la utilizarea concretă a sistemului;
  - fiecare caz de testare se bazează de obicei pe un scenariu şi acoperă diferite ramificaţii de execuţie (indicate prin cazuri speciale de date de intrare sau condiţii speciale);
  - pentru un caz de testare specifică:
    - **toate precondiţiile** care este necesar să fie satisfăcute pentru execuţia cazului de utilizare cu succes;
    - **postcondiţiile** impuse asupra rezultatelor obţinute;
    - **descrierea stării finale a sistemului** după ce testarea cazului de utilizare s-a realizat cu succes;

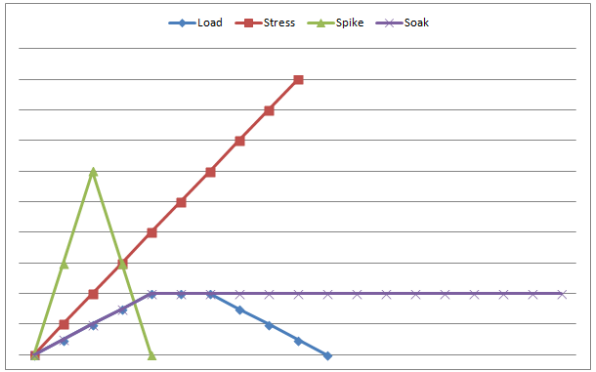
Testare de sistem. Testare non-funcţională. Definiţie

- **testare non-funcţională** (*engl.* **non-functional testing**):
  - verifică modul în care sistemul îndeplineşte cerinţele non-funcţionale;
  - stabileşte dacă sistemul este pregătit pentru a fi efectiv utilizat;

- în [Myers2004, Cap 6] sunt descrise 15 tipuri de testare (non-funcţională) de sistem pentru care se scriu cazuri de testare:
  - Volume testing;
  - Stress testing;
  - Usability testing;
  - Security testing;
  - Performance testing;
  - Storage testing;
  - Configuration testing;
  - Compatibility/Conversion testing;
  - Instability testing;
  - Reliability testing;
  - Recovery testing;
  - Serviceability testing;
  - Documentation testing;
  - Procedure testing;
  - Facility testing.

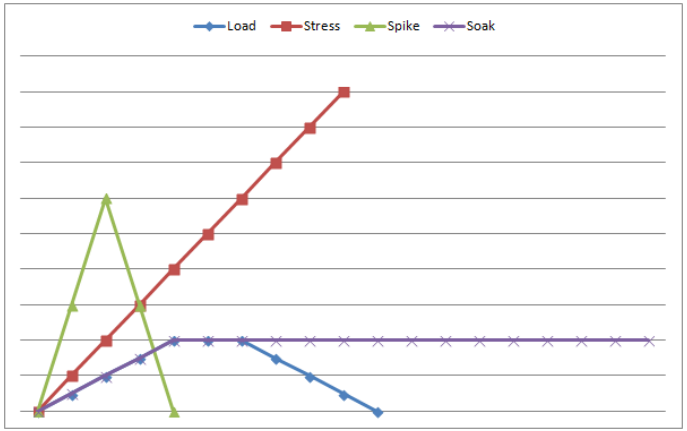
Testare de sistem. Tipuri de testare non-funcţională (1)

- **performance testing:** - cât de bine face sistemul ceea ce trebuie să facă
  - fiecare sistem are cerinţe referitoare la performanţa utilizării;
    - e.g., execuţia unei funcţionalităţi să fie realizată într-un anumit interval de timp, iar resursele să nu fie infinite;
    - bug-urile legate de performanţă indică deficienţe la nivel de proiectare care determină degradarea performanţei sistemului soft în timpul utilizării;
  - **obiectiv:** identificarea blocajelor determinate de reducerea performanţei componentelor şi a sistemelor, i.e., **bottlenecks**;
  - tipuri:
    - **Capacity Testing, Load Testing;**
    - **Volume Testing, Stress Testing;**
    - **Soak Testing, Spike Testing.**



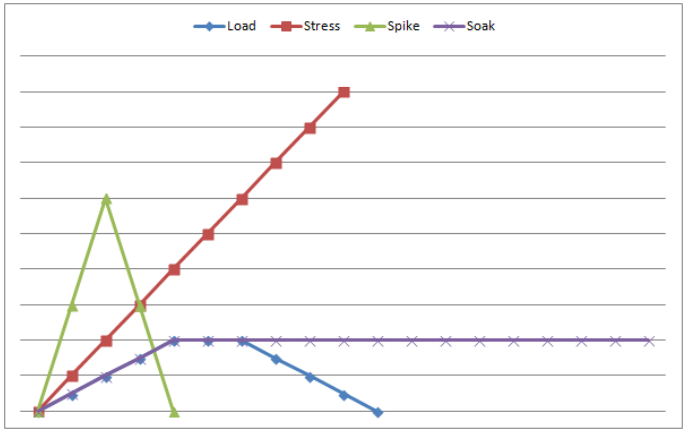
Testare de sistem. Tipuri de testare non-funcţională (2)

- **capacity testing:**
  - permite preveniea eventualelor probleme determinate de creşterea numărului de utilizatori sau a volumului de date;
  - **obiectiv:** obţinerea de informaţii referitoare la **nivelul maxim de utilizatori şi/sau date care nu afectează performanţa:**
    - E.g.: câte fişiere pot fi folosite fără a afecta performanţa.



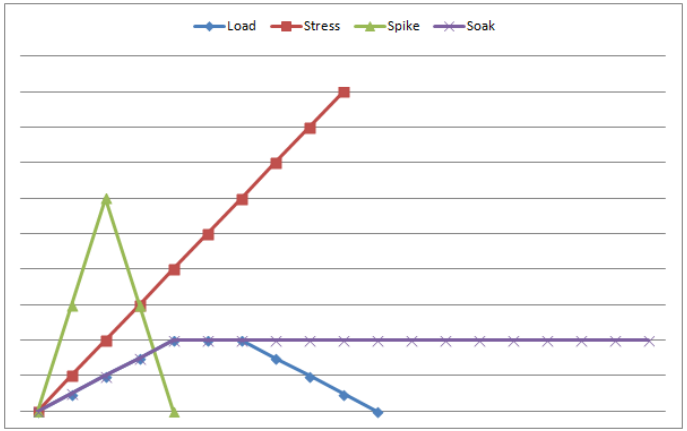
Testare de sistem. Tipuri de testare non-funcţională (3)

- **load testing:**
  - evaluează capacitatea sistemului de opera cu volume de date normale sau în condiţii de solicitare (vârf);
  - **obiectiv:** obţinerea de informaţii referitoare la comportamentul sistemului în **anumite condiţii de utilizare, i.e., normale şi de solicitare:**
    - E.g.: creşterea numărului de fişiere folosite.

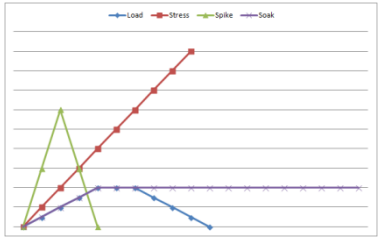


Testare de sistem. Tipuri de testare non-funcţională (4)

- **volume testing:**
  - evaluează comportamentul sistemului atunci când se gestionează volume mari de date;
  - **obiectiv:** obţinerea de informaţii legate de **funcţionarea aplicaţiei cu un volum de date ridicat**:
    - E.g.: creşterea dimensiunii fişierelor folosite.



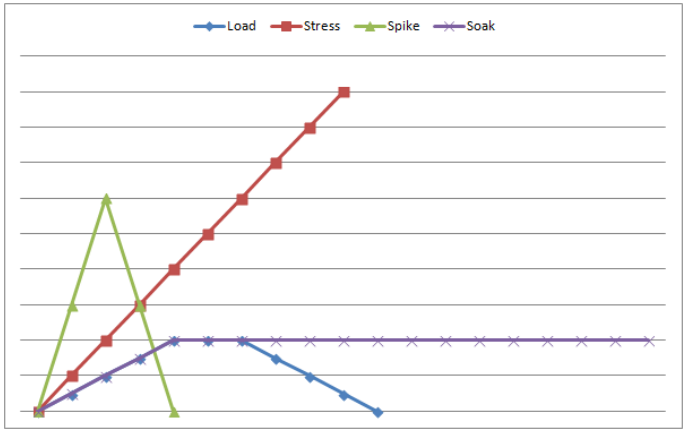
Testare de sistem. Tipuri de testare non-funcţională (5)



- **stress testing:** - pune presiune asupra limitelor sistemului
  - investighează dacă comportamentul softului se degradează în condiţii extreme de utilizare şi când nu are acces la resursele necesare;
  - **este de dorit ca degradarea softului cauzată de creşterea numărului de cereri să se realizeze acceptabil, fără să ducă la un eşec imediat în timpul testării; sistemul nu trebuie să eşueze catrastofal;**
  - relevant pentru sistemele distribuite care pot indica o degradare severă atunci când reţeaua devine încărcată;
  - **obiectiv:** obţinerea de informaţii despre **modul în care sistemul funcţionează în condiţii extreme, dincolo de limitele normale**:
    - E.g.: creşterea numărului de fişiere folosite până la failure şi chiar mai mult;
      - verifică dacă are loc o pierdere inacceptabilă de date şi/sau imposibilitatea de a utiliza anumite servicii.

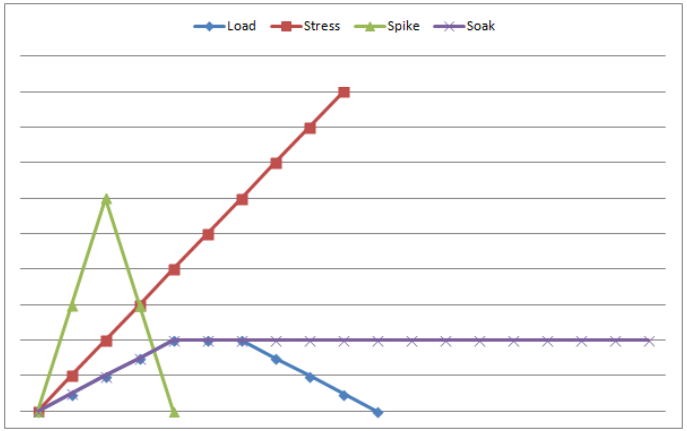
Testare de sistem. Tipuri de testare non-funcţională (6)

- **soak testing:**
  - verifică dacă sistemul face faţă unui volum mari de date pentru o perioadă mai mare de timp;
  - **obiectiv:** obţinerea de informaţii legate de **posibilitatea de a utiliza volum de date şi care sunt consecinţele dacă se depăşeşte vomulul pentru care a fost proiectat**:
    - E.g.: verifică dacă creşterea dimensiunii fişierelor folosite este suportată.



Testare de sistem. Tipuri de testare non-funcţională (7)

- **spike testing:**
  - evaluează comportamentul sistemului la schimbări bruşte şi extreme a condiţiilor de lucru, i.e., puţine sarcini, multe sarcini;
  - **obiectiv:** obţinerea de informaţii legate de **punctele slabe ale sistemului determinate de modificarea rapidă a condiţiilor de lucru**:
    - E.g.: creşterea brusc şi diminuarea rapidă a numărului de fişiere cu care se lucrează.



Testare de sistem. Tipuri de testare non-funcţională (8)

- **reliability testing:**
  - investighează capacitatea un soft de a funcţiona conform aşteptărilor utilizatorului, chiar şi atunci când eşuează;
  - **determină cât timp şi cât de eficient poate funcţiona sistemul fără eroare;**

- **security testing:**
  - calitatea, securitatea şi gradul de încredere în aplicaţie (*engl.* **reliability**) sunt caracteristici depedendente;
  - defectele produsului soft pot fi exploatate pentru a identifica breşe de securitate;
  - **presupune simularea unor atacuri la nivel de securitate, având scopul de a identifica vulnerabilităţile softului.**

TESTARE DE ACCEPTARE

Definiţie. Caracteristici. Etape de realizare. Clasificare

Alpha Testing. Beta Testing

Alpha Testing vs Beta Testing

Alte tipuri de testare de acceptare

Dificultăţi de testare

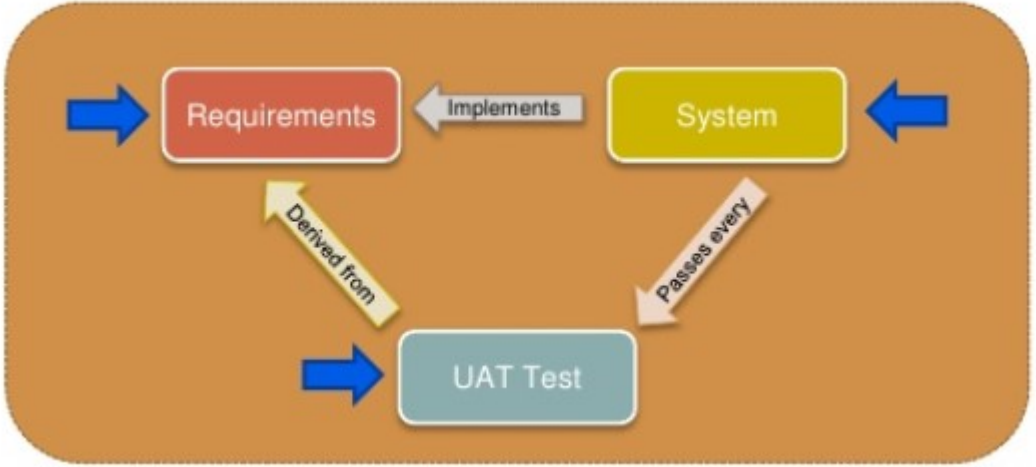
Testare de acceptare. Definiţie. Clasificare

- **testare de acceptare** (*engl.* **user acceptance testing, UAT**):
  - procesul de testare prin care se verifică dacă programul îndeplineşte cerinţele iniţiale şi nevoile curente ale utilizatorului final;
- nu este responsabilitatea dezvoltatorului produsului;
- tester = clientul/ beneficiarul;
- realizată efectiv de către client, care aplică tehnici de testare black-box.

- **tipuri de testare de acceptare:**
  - **alpha testing, beta testing;**
  - contract acceptance testing;
  - regulation acceptance testing;
  - operational acceptance testing.

Testare de acceptare. Etape de realizare

- **etape de realizare:**
  - **definirea criteriilor** prin care produsul soft este considerat funcţional;
  - **creare unei suite de cazuri de testare** pentru UAT;
  - **rularea testelor UAT**;
  - **evaluarea şi raportarea rezultatelor.**



Testare de acceptare. Alfa Testing. Beta Testing

	Alpha testing	Beta testing
Când?	Înainte de livrare	Înainte de livrare
Cine?	**clientul sau alte persoane** desemnate, care testează produsul pe o **platformă instalată la dezvoltator;**	**clientul** testează produsul soft pe o platformă unde dezvoltatorul nu poate interveni, într-un **mediu instalat la beneficiarul produsului soft**;
Unde?	**produsul soft se foloseşte într-un mediu controlat**, unde programatorul poate interveni imediat pentru a elimina bug-urile;	**produsul soft este folosit în mediul pentru care a fost dezvoltat;**
Cum?	se folosesc tehnici de **testare black-box;**	se folosesc tehnici de **testare black-box**;
Raportare bug-uri?	defectele sunt **inventariate** şi **eliminate/rezolvate imediat**;	**clientul inventariază** dificultăţile de utilizare ale produsului soft şi **le raportează programatorului pentru a fi rezolvate**.

Testare de acceptare. Alpha Testing vs Beta Testing

	Alpha testing	Beta testing
Tipuri de testare ?	**reliability testing şi security testing NU se realizează în profunzime;**	**realibility testing, security testing şi robustness testing se realizează în detaliu;**
Eliminare bug-uri ?	**deficienţele majore pot fi rezolvate imediat** de programator;	**deficienţele raportate sunt investigate ulterior**, iar îmbunătăţirile şi corecturile se regăsesc în versiunile ulterioare ale produsului soft;
Relevanţă?	permite **simularea unui mediu real** de utilizare înainte de a fi trimis la client (beta testing);	furnizează un **feedback autentic (real)** din partea utilizatorului final.

Testare de acceptare. Tipuri de de testare de acceptare

- **contract acceptance testing:**
  - produsul soft este testat din perspectiva îndeplinirii unor criterii şi specificaţii care sunt precizate într-un **contract de colaborare** între dezvoltator şi client;
  - criteriile şi specificaţiile de acceptanţă sunt stabilite la semnarea contractului, **înainte** de dezvoltarea softului;

- **regulation acceptance testing, i.e., compliance acceptance testing:**
  - verifică dacă produsul soft dezvoltat respectă regulamentele şi legile în vigoare referitoare de utilizarea şi funcţionarea unui produs soft specfic;

- **operational acceptance testing, i.e., operational readiness testing, production acceptance testing:**
  - verifică dacă există fluxurile informaţionale necesare pentru utilizarea produsului soft de către client (e.g., planuri de backup, training pentru utilizatori, diferite procese de întreţinere şi verificări de securitate).

Testare de acceptare. Dificultăţi de testare

- **dificultăţi care apar la nivelul testării de acceptare:**
  - cerinţe care nu sunt descrise satisfăcător (clar, complet, corect);
  - planificarea târzie (defectuoasă) a activităţilor de testare;
    - testarea nu este realizată riguros, nu este o activitate planificată şi monitorizată;
    - identificarea tardivă a defectelor, cu dificultăţi de eliminare a acestora.

TIP DE TESTARE VS NIVEL DE TESTARE

Tip de testare. Nivel de testare. Definiţie
Obiective de testare. Exemple
Retestare. Definiţie
Testare de regresie. Definiţie
Retestare vs Testare de regresie

Tip de testare. Nivel de testare. Definiţie

- **nivel de testare** (*engl.* **testing level**):
  - o serie de activităţi de testare asociate unei etape din procesul de dezvoltare al produsului soft;
  - **Ce testez?**

- **tip de testare** (*engl.* **testing type**):
  - mijlocul prin care un **obiectiv** al testării, stabilit anterior pentru un **nivel de testare,** poate fi realizat;
  - **Cum testez?**

Tip de testare. Exemple

- exemple de tipuri de testare:
  - **testarea unei metode:**
    - se poate realiza prin aplicarea unor criterii de testare (black-box, white-box), la nivelul *testării unitare* sau *de integrare*;
  - **testarea unei caracteristici non-funcţionale:**
    - se realizează prin aplicarea unui anumit tip de testare, e.g., testare de performanţă, testare de utilizabilitate, cu scopul de a evalua o caracteristică a calităţii produslui soft, la nivelul *testării de sistem*;
  - **testarea după eliminarea unui bug:**
    - se realizeză prin aplicarea **re-testării** (*engl.* **re-testing, confirmation testing**) după depanare, la *orice nivel de testare*;
  - **testarea legată de eliminarea unui bug:**
    - se realizează prin **testarea de regresie** (*engl.* **regression testing**), pentru a verifica dacă eliminarea unui bug nu are efecte secundare asupra softului, la *orice nivel de testare*.

Re-testare. Definiţie

- **re-testare** (*engl.* **re-testing, confirmation testing**):
  - re-execuţia testelor care au pus în evidenţă anterior un bug ce se presupune că a fost eliminat;

- **scop:** confirmarea că defectul a fost eliminat;
- cazurile de testare re-executate sunt identice cu cele rulate anterior;

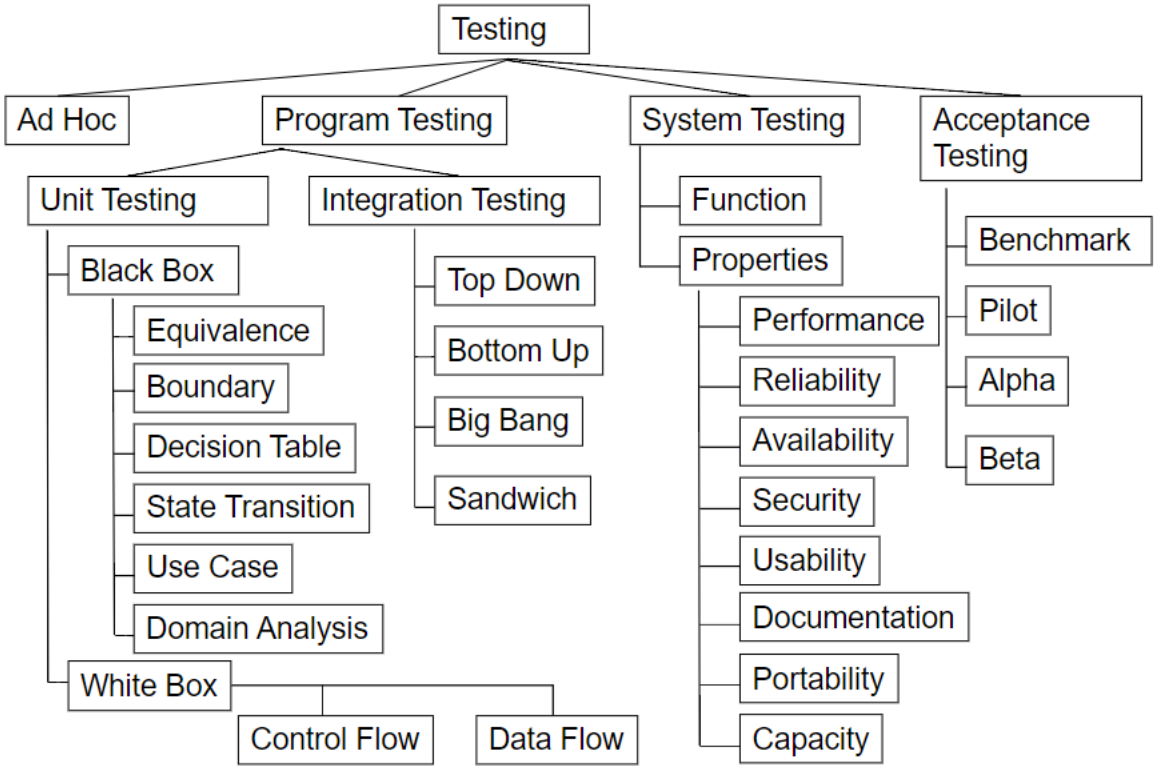
Testare de regresie. Definiţie

- **testare de regresie** (*engl.* **regression testing**):
  - re-execuţia unor teste care au fost rulate anterior cu succes;

- **scop:** identificarea efectelor secundare (bug-uri) care pot apărea în urma modificării unor module;
- cazurile de testare se pot organiza in teste de regresie, care permit testarea:
  - tuturor funcţionalităţilor sistemului;
  - funcţionalităţilor cu probabilitate ridicată de a fi afectate de modificări;
  - comportamentului componentelor sistemului care au fost modificate.

- **testare de regresie ≠ re-testare;**

Niveluri de testare. Tipuri de testare



PENTRU EXAMEN…

Pentru examen...

- **Niveluri de testare. Definiţii şi caracteristici:**
  - testare unitară;
  - testare de integrare;
    - 4 strategii (big-bang, top-down, bottom-up, sandwich), descriere, comparare;
  - testare de sistem;
    - testare funcţională;
    - 5 tipuri de testare non-funcţională (volume, stress, load, usability, security) [Mye04].
  - testare de acceptare;
    - alpha testing, beta testing.
- **Tip de testare vs Nivel de testare. Definiţii şi caracteristici:**
  - re-testare;
  - testare de regresie.

Cursul următor...

- **Curs 05:**
  - **Tematică**
    - **Test Automation Demo: Selenium WebDriver + Serenity BDD**
    - **Performance Testing**
  - **Companie IT invitată: Evozon**
  - **Data: Marţi, 28 Martie 2023;**
  - **Orele: 08:00-10:00;**
  - **Desfăşurare: Sala 2/I (N. Iorga), Clădirea Centrală a UBB.**

Seminar 04. Create and Solve a Puzzle (1)

- **Create and Solve a Puzzle**
  - **termen: 26 aprilie 2023;**
  - **echipe:** max. 3 studenţi/echipă, i.e., echipe de forma (A, B, C) sau (A, B);
  - echipele participante pot rezolva unul sau ambele task-uri de mai jos; pentru fiecare task rezolvat corect se acordă **câte 2 puncte** de activitate la seminarul 4 fiecărui membru al echipei;
  - **Task 01. Create a Puzzle:** cu min. 20 concepte studiate la VVSS;
    - se va accesa pagina https://www.puzzle-maker.com/ şi se va alege tipul de joc ***crossword***;
    - se va crea un puzzle folosind min. 20 termeni studiaţi în cadrul disciplinei VVSS;
    - se va genera puzzle-ul pe fundal alb şi se va posta pe channel-ul **#Games**;
    - echipa se va înscrie în fişierul Seminar 04 pentru a putea primi punctajul pentru crearea puzzle-ului;
    - pentru acest task se poate folosi orice alt tool care permite obţinerea puzzle-ului în forma cerută.

Seminar 04. Create and Solve a Puzzle (2)

- **Create and solve a puzzle**
  - **Task 02. Solve a Puzzle**;
    - o echipă poate rezolva un puzzle propus de o altă echipă;
    - o echipă poate rezolva un puzzle chiar dacă nu a propus anterior un puzzle;
    - după rezolvarea puzzle-ului (prin editarea fişierului sau listare, apoi rezolvare, apoi poză/scan) de către echipa (X, Y, Z), acesta se trimite **doar** echipei (A, B, C) care a propus puzzle-ul, pentru a fi evaluat;
      - fiecare răspuns corect primeşte 0.10 puncte (0.10x20 întrebări = 2 puncte);
      - echipa (A, B, C) evaluează soluţia primită şi completează în fişierul Seminar 04 componenţa echipei care a oferit soluţia şi punctajul corespunzător (e.g., 2 puncte, 1.8. puncte);
      - după ce au fost evaluate soluţii oferite de max. 6 echipe participante, echipa care a propus puzzle-ul va posta soluţia la puzzle în thread-ul postării iniţiale pe channel-ul **#Games**, indicând prin aceasta faptul că nu mai poate primi spre evaluare alte soluţii.

Referințe bibliografice

- **[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.
- **[MeszarosFowler2006]** Meszaros, G., Fowler, M., *Test Doubles*, https://martinfowler.com/bliki/TestDouble.html