

CURS 04.

NIVELURI DE TESTARE

Verificarea și validarea sistemelor soft
[18 Martie 2025]

Lector dr. Camelia Chisăliță-Crețu
Universitatea Babeș-Bolyai

Conținut

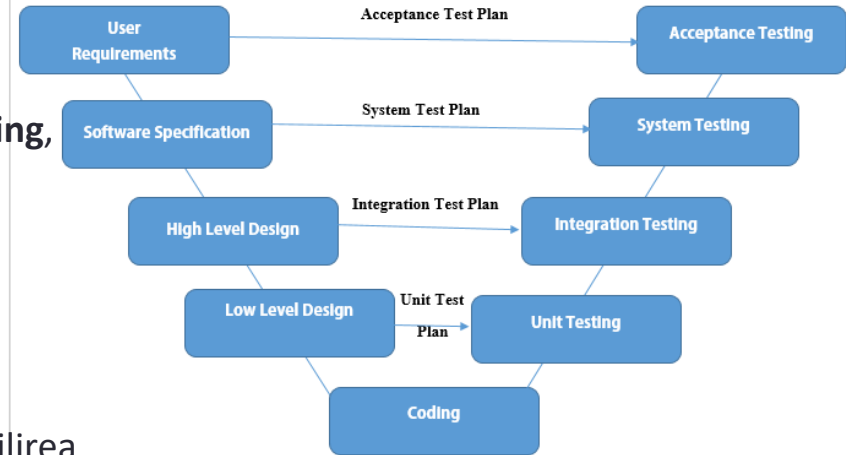
- **Niveluri de testare**
 - Definiție. Clasificare
- **Testare unitară**
 - Definiție. Motivație. Caracteristici
 - Proiectarea cazurilor de testare
 - Tipuri de bug-uri identificate
 - Reguli generale de aplicare
- **Testare de integrare**
 - Definiție. Motivație. Clasificare
 - Integrare non-incrementală.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

NIVELURI DE TESTARE

Definiție. Clasificare

Nivel de testare. Definiție. Clasificare

- **nivel de testare** (*engl. testing level*):
 - o serie de activități 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 cerințelor sistemului = stabilirea obiectivelor de realizat;
 - **testare de acceptare** (*engl. acceptance testing*);
 - etapa: descrierea cerințelor utilizatorului.



V-Model

TESTARE UNITARĂ

Definiție. Motivație. Caracteristici

Proiectarea cazurilor de testare

Tipuri de bug-uri identificate

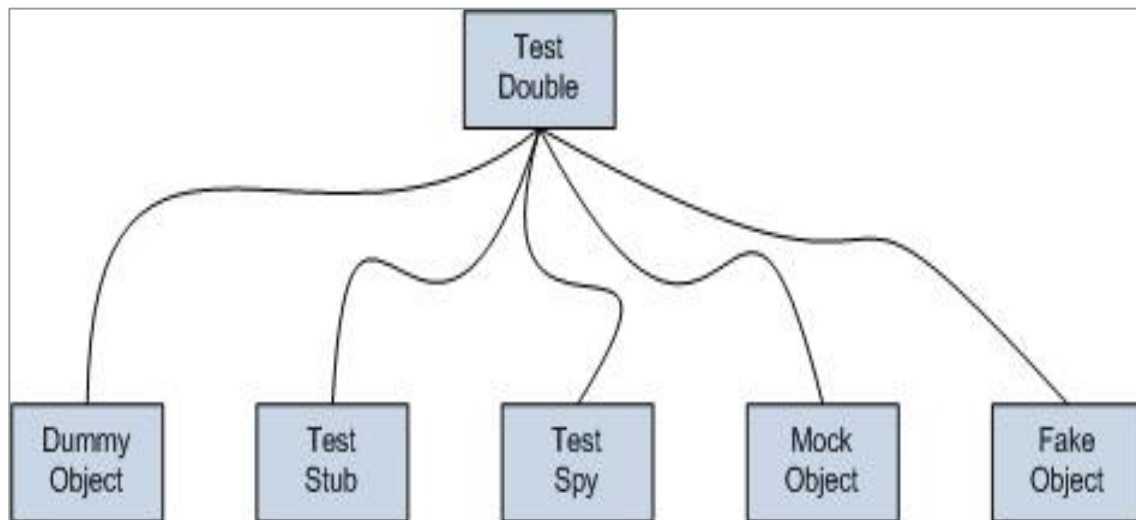
Reguli generale de aplicare

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 evaluarea rezultatelor testării.

Testare unitară. Tipuri de obiecte (1)

- la nivelul testării unitare se folosesc diferiți 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 funcționarea obiectelor utilizate în realitate [\[MeszarosFowler2006\]](#);

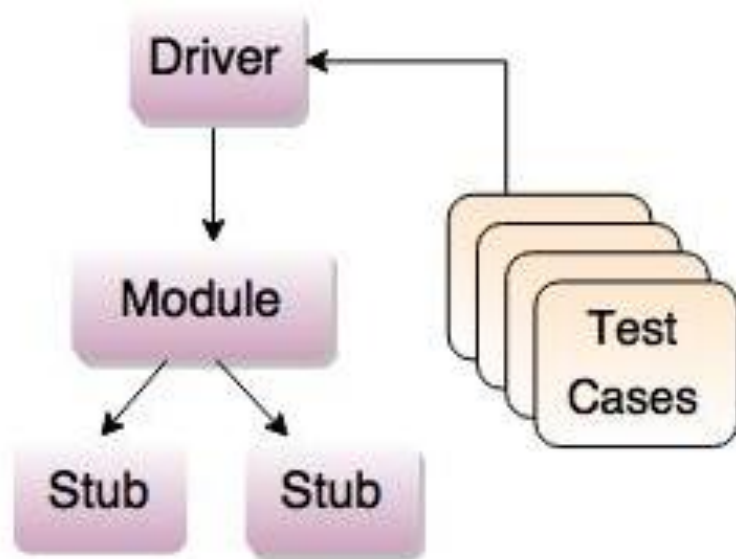


Testare unitară. Tipuri de obiecte (2)

- tipuri de obiecte utilizate de tool-uri:
 - **Dummy** – obiecte care sunt transmise ca parametri dar care **nu sunt folosite de metodele apelate**;
 - e.g., diverși parametri precizați doar pentru a respecta signatura metodei apelate;
 - **Fake** – obiecte cu **implementări funcționale**/utilizabile, dar **simpliste**, care nu sunt adecvate pentru a fi incluse în livrabilul către client;
 - e.g., o colecție de date in-memory;
 - **Stubs** – obiecte sau metode ale unor obiecte care **furnează 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/reține informații referitoare la modul în care au fost folosite;
 - e.g., un serviciu pentru e-mail care reține numărul de mesaje transmise;
 - **Mocks** – obiecte pentru care s-a stabilit **un anumit comportament** (behavior expectations) și sunt utilizate pentru a observa interacțiunea 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 conțin 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 specificații;
 - 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 pași de executat;
2. execuție: planificare și durată;
3. consecvență;
4. atomicitate;
5. responsabilitate unică;
6. izolarea testelor;
7. izolarea de mediul de execuție;
8. izolarea claselor;
9. automatizare completă;
10. *self-descriptive*;
11. fără condiții logice;
12. fără bucle;
13. fără tratarea excepțiilor;
14. utilizarea instrucțiunilor assert;
15. utilizarea de mesaje sugestive;
16. fără testare în codul sursă livrat;
17. separarea pe module și niveluri ale arhitecturii;
18. gruparea testelor în funcție de tipul de testare.

Testare unitară. Reguli generale de aplicare (2)

- număr de pași de executat:
 - 3-5 pași:
 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 execuție:**
 - frecvența de execuție:
 - 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;
 - execuție după salvare, IDE (*engl. IDE runs tests after save*): la fiecare câteva minute;
 - mod de execuție (singular sau suită de teste):
 - timp de execuție pentru 10 teste = timp de execuție 1 test x 10;
 - **un test care se execută greu, încetinește 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)

- **consecvență:**
 - execuția repetată a aceluiași test ar trebui să returneze în mod repetat același rezultat, dacă nu au avut loc modificări asupra codului sursă;
- cod sursă problematic:
 - `Date current Date = new Date();`
 - `Int value = random.nextInt(100);`
- soluții:
 - utilizarea obiectelor dummy, mock, stub, fake;
 - injectarea dependențelor.

Testare unitară. Reguli generale de aplicare (5)

- **atomicitate:**
 - rezultate posibile ale execuției unui test:
 - **passed;**
 - **failed;**
 - nu există teste care „au trecut” doar parțial;
 - **dacă un punct de execuție 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 execuție;
- se testează comportamentul metodei:
 - **o metodă, mai multe utilizări (comportamente)**
 - ==> mai multe teste și cel puțin un test pentru fiecare comportament;
 - mai multe instrucțiuni `assert` în același test – doar dacă verifică același 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/ responsabilitate 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/ responsabilitate 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)

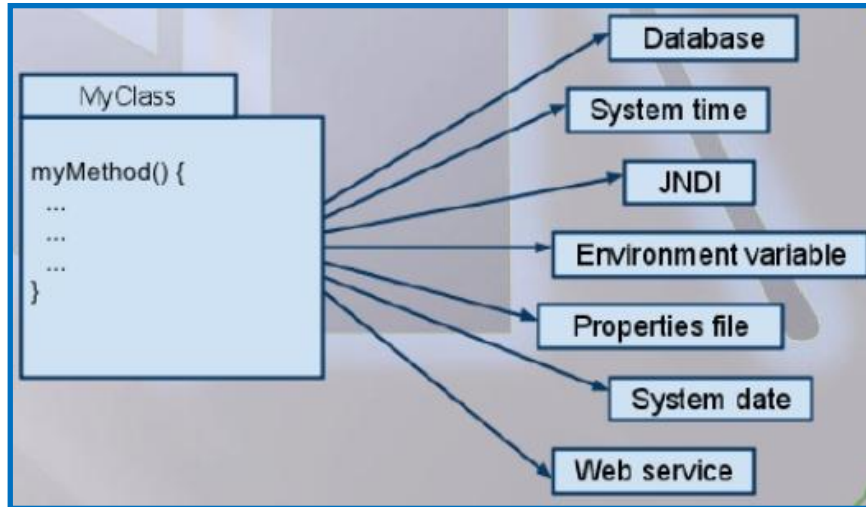
- **izolarea testelor (unele de altele):**
 - testele trebuie să fie independente unele de altele.
 - **la execuții diferite (la momente de timp diferite, în ordine diferită) ale aceluiași test trebuie să se obțină aceleași rezultate.**
 - **nu** se partajează starea/ contextul de execuție între teste;
 - 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 execuție (context):**
 - testele trebuie izolate de influențele mediului de execuție;
 - E.g.,
 - baze de date;
 - apelarea serviciilor web;
 - Java Naming and Directory Interface (JNDI);
 - variabile de mediu definite local;
 - fișiere de proprietăți;
 - configurările de dată și oră ale sistemului.

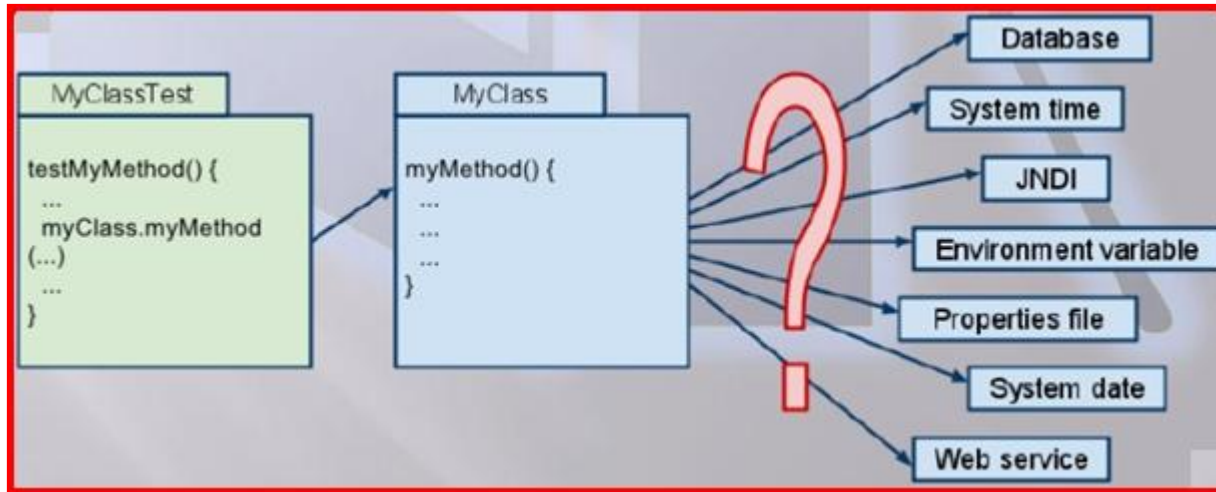
Testare unitară. Reguli generale de aplicare (11)

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



Testare unitară. Reguli generale de aplicare (12)

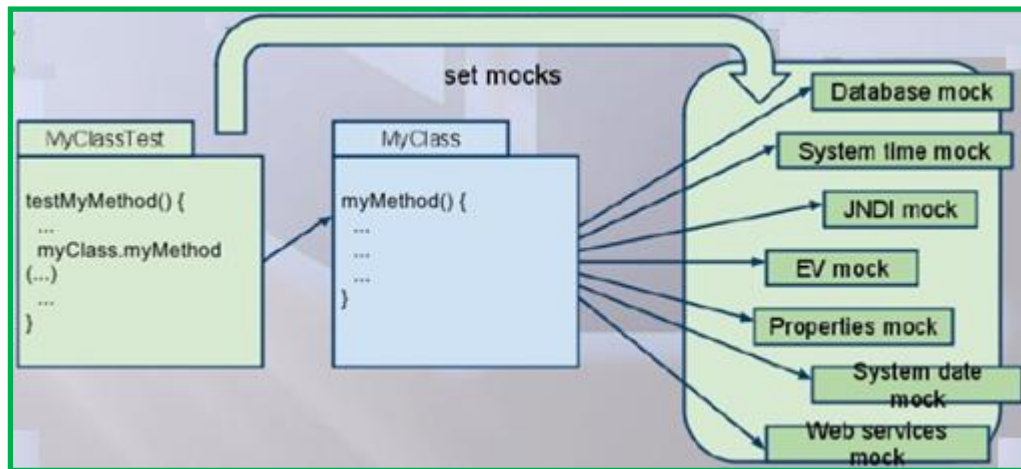
- izolarea testelor de mediul de execuție (context):
 - la testare se folosește un **mediu de testare**, care nu este întotdeauna identic cu **mediul de execuție** de la dezvoltarea codului sursă sau de la client;



Testare unitară. Reguli generale de aplicare (13)

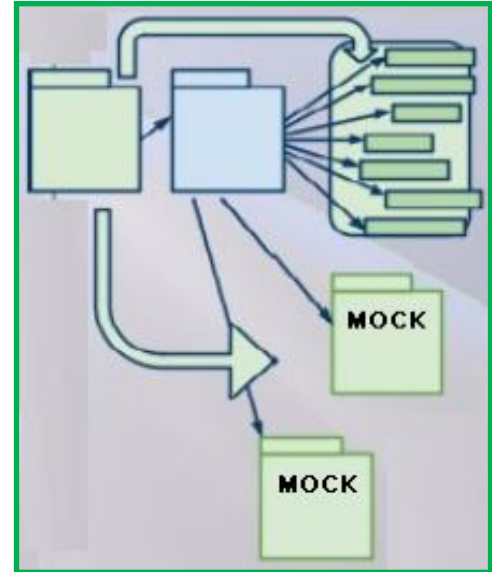
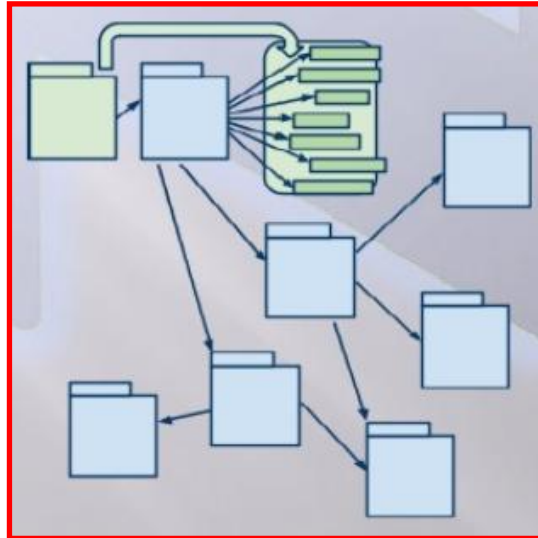
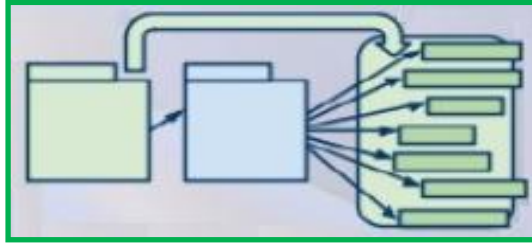
- izolarea testelor de mediul de execuție (context):
- **soluție:** se utilizează obiecte mock care indică un anumit mediu de utilizare;

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



Testare unitară. Reguli generale de aplicare (14)

- izolarea claselor:
- clase fără dependențe
- clase cu dependențe
 - soluție: dependențe 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 dependențelor**, în locul apelului constructorilor în cadrul metodelor;
 - **folosirea interfețelor**.

Testare unitară. Reguli generale de aplicare (16)

- **automatizare completă:**
 - fără pași executați manual pe durata testării.
- se automatizează:
 - execuția testelor;
 - colectarea rezultatelor testării;
 - stabilirea rezultatului testării (**passed**/ **failed**);
 - transmiterea rezultatelor testării prin: e-mail, IDE integration, etc;

Testare unitară. Reguli generale de aplicare (17)

- **self-descriptive:**
 - la nivelul testării unitare un test reprezintă:
 - documentație la nivel de dezvoltare;
 - metodă de specificare care reflectă versiunea actualizată a cerințelor;
- **caracteristici:**
 - un test trebuie să fie ușor de citit și înțeles;
 - denumirile variabilelor, metodelor și claselor trebuie să fie *self-descriptive*;
 - **nu trebuie să conțină condiții logice sau iterații**;
 - 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 instrucțiuni logice:**
 - un caz de testare **nu** ar trebui să conțină instrucțiunea `if` sau `switch`;
- **nu** există incertitudini legate de:
 - *datele de intrare* ==> toate datele de intrare sunt cunoscute;
 - *comportamentul așteptat* ==> comportamentul metodei este predictibil;
 - *datele de ieșire* ==> rezultatele așteptate ar trebui să fie bine definite.

Testare unitară. Reguli generale de aplicare (19)

- **fără instrucțiuni logice:**
 - mai multe condiții trebuie să se regăsească în cazuri de testare distincte, nu în același test, fiind tratate ca și ramificații ale instrucțiunilor 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 instrucțiuni iterative** (i.e., while, do while, for):
 - scenarii tipice pentru iterații:
 - **câteva sute de iterații:**
 - dacă sunt necesare câteva sute de iterații, atunci testul este destul de complicat și **este necesar să fie simplificat**;
 - **câteva iterații:**
 - 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 iterații necunoscut:**
 - numărul de iterații este greu de evitat, indicând faptul că e posibil ca **testul să fie incorect** și 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)

- **instrucțiunea `assert`:**
- se recomandă:
 - utilizarea instrucțiunilor `assert` disponibile în platforma de testare;
 - crearea propriilor aserțiuni pentru verificarea condițiilor complexe care se repetă în diferite teste;
 - reutilizarea propriilor metode asertive;
 - utilizarea aserțiunilor în cadrul buclelor.

Testare unitară. Reguli generale de aplicare (22)

- **mesaje sugestive în instrucțiunea `assert`:**
 - prin citirea mesajului din aserțiune se poate recunoaște ușor problema care a determinat eșuarea testului;
- **avantaje:**
 - permit îmbunătățirea documentației codului sursă;
 - oferă informații asupra problemei dacă testul a eșuat (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 și/sau atribute care sunt folosite doar la testare.

Testare unitară. Reguli generale de aplicare (24)

- **gruparea testelor în funcție de tipul de testare:**
- după modulul testat:
 - organizarea testelor în pachete corespunzătoare modulelor testate;
 - utilizarea unei abordări ierarhizate;
 - scăderea timpului de execuție pentru suitele de teste prin împărțirea acestora în suite de dimensiuni mai mici, i.e., suitele de mici dimensiuni pot fi executate mai des/frecvent;
- după tip:
 - scopul / obiectivele testării;
 - frecvența de execuție;
 - momentul execuției suitei;
 - acțiunea în caz de eșec.

TESTARE DE INTEGRARE

Definiție. Motivație. Clasificare

Integrare non-incrementală. Integrare incrementală. Integrare mixtă

Compararea strategiilor de integrare

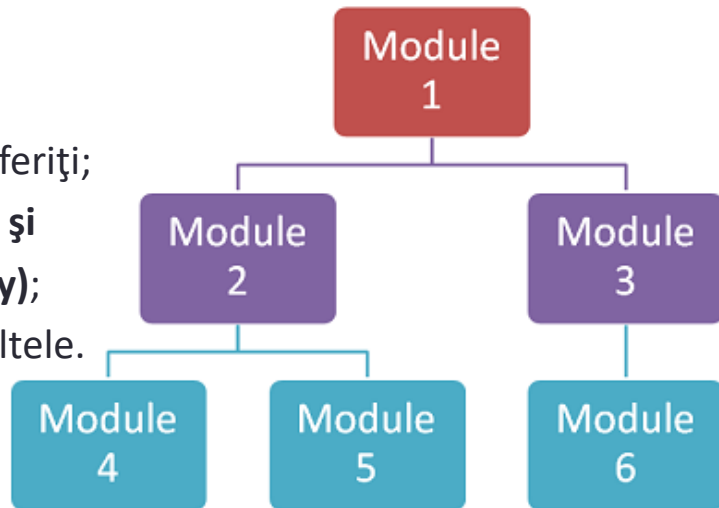
Testarea interfeței modulelor. Definiție. Clasificare

Tipuri de bug-uri identificate

Exemplu

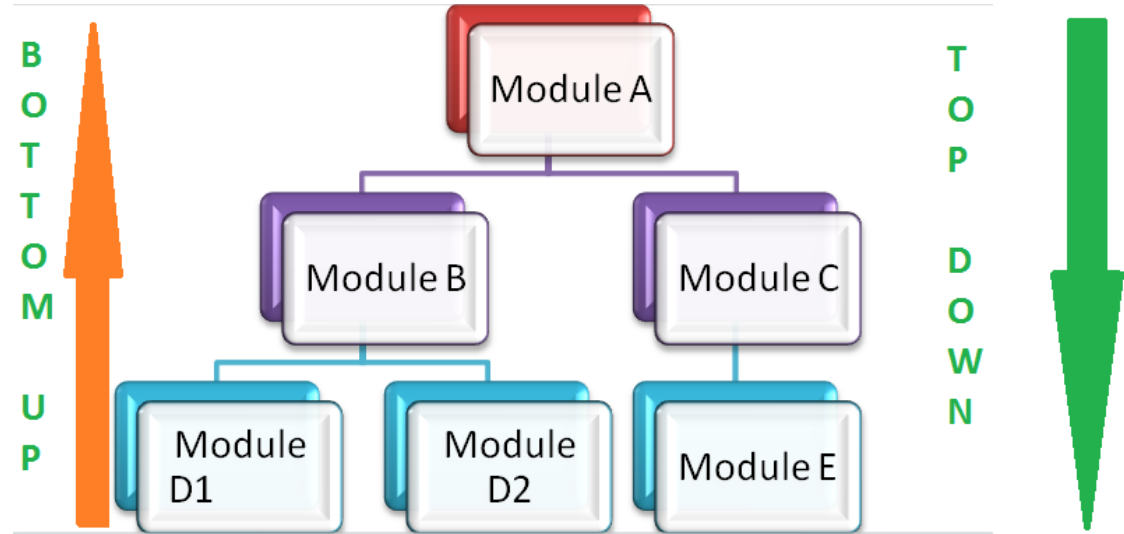
Testare de integrare. Definiție. Motivație

- **testare de integrare** (*engl. integration testing*):
 - nivel de testare în care modulele individuale se combină și se testează ca un grup;
 - permite construirea structurii programului pe măsură ce este testat pentru a **identifica erorile la nivelul interfeței dintre module**;
- **motivație:**
 - module diferite sunt implementate de programatori diferiți;
 - testarea unitară se desfășoară într-un **mediu controlat și izolat**, folosind entități **driver** și **stub (mock, fake, dummy)**;
 - unele module pot genera mai multe defecțiuni decât altele.



Testare de integrare. Clasificare

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



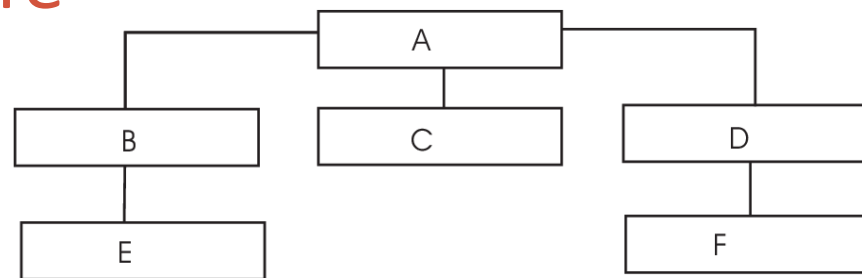
Integrarea Big-bang. Descriere

- 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 funcționalității programului;



- **dezavantaje:**

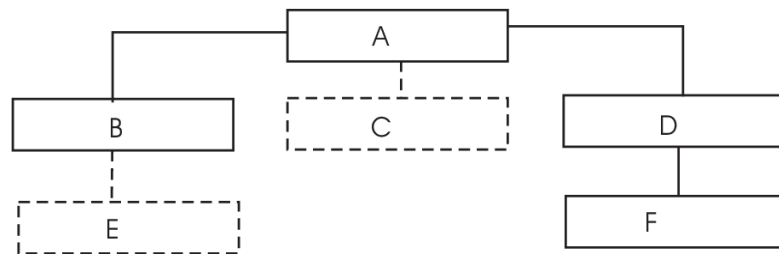
- necesită un volum de muncă ridicat;
- depanare dificilă – este dificil de izolat modulul care a determinat defecțiunea;
- dacă programul este complex – este dificil de urmărit modul în care s-au executat funcționalitățile.

- **avantaje:**

- dezvoltare și testare unitară paralelă.

Integrare incrementală Top-down. Descriere

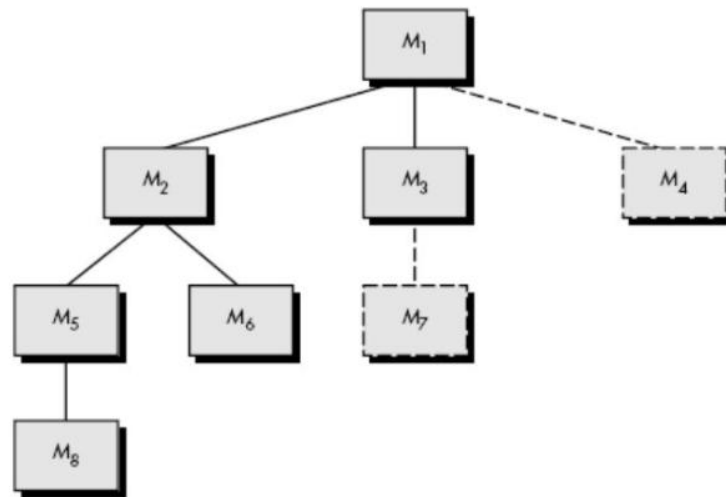
- **integrare incrementală Top-down** (*engl. Top-down incremental integration*):
 - permite construirea și testarea programului adăugând module noi pe parcurs;
 - integrarea modulelor se face de sus în jos, de la modulul principal, modulele subordonate sunt încorporate succesiv;
 - defectele sunt ușor de izolat și 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 ramificație majoră a arhitecturii aplicației;
 - ordinea de integrare a ramificațiilor depinde de caracteristicile aplicației;
- **integrarea breadth-first**
 - integrează toate componentele care se află pe nivelul direct subordonat, i.e., pe orizontală;

Integrare incrementală Top-down. Algoritm

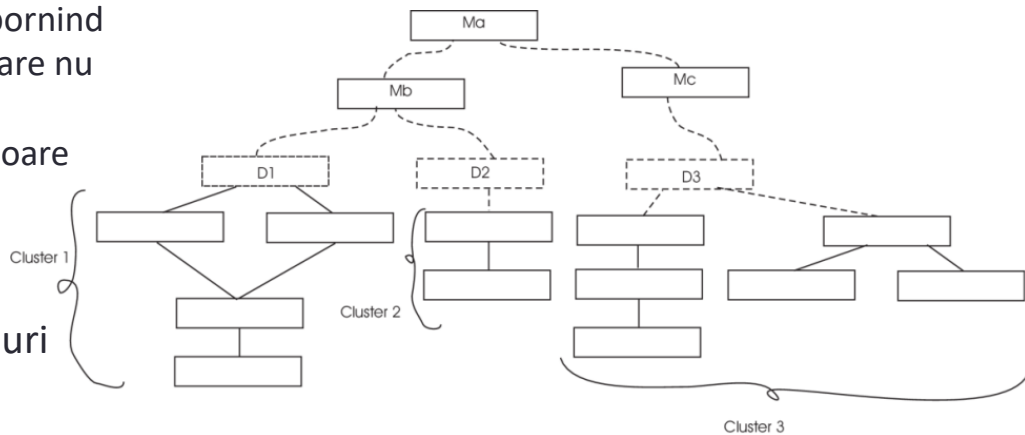
- după testarea unitară a modulului principal folosind **stub-uri** pentru modulele imediat subordonate, procesul de **integrare Top-down** constă în:
 1. în funcție de tipul de integrare ales (**depth-first/breadth-first**), se înlocuiește un stub cu un modul real;
 2. se testează cu modulul subordonat concret care a fost integrat;
 3. pentru integrarea unui nou modul se repetă pașii 1 și 2.
- **driver** = modulul principal; nu se folosesc alte drivere;
- **stub** = înlocuiește un modul direct subordonat;



- **E.g.,**
 - **depth-first:** $M_1, M_2, M_5, M_8, M_6, M_3, M_7, M_4$;
 - **breadth-first:** $M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8$.

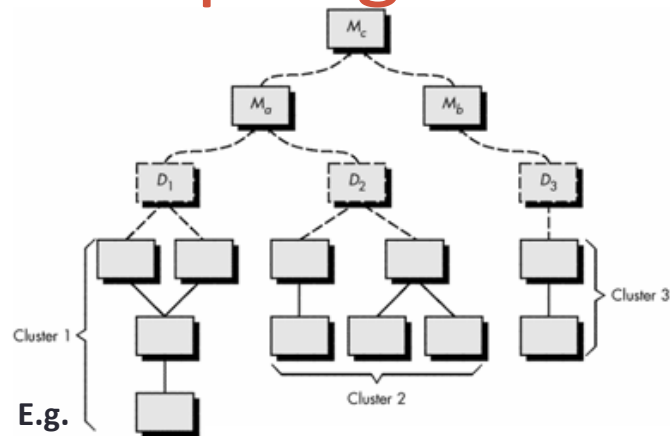
Integrare incrementală Bottom-up. Descriere

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

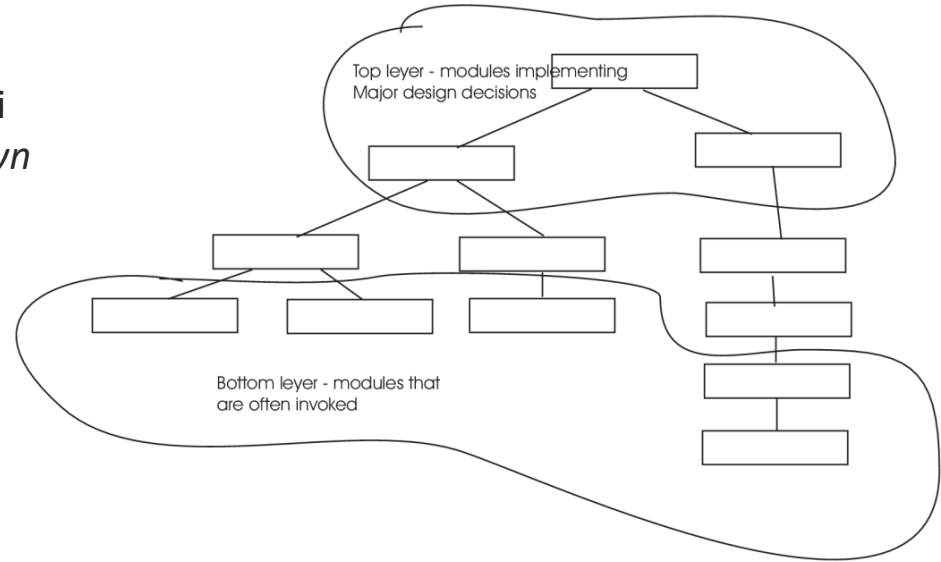
- Procesul de integrare Bottom-up:
 - pentru toate modulele terminale sau clusteri de module se descriu drivere și se testează;
 - *module terminale*:
 - se înlocuiește driver-ul cu modulul de pe nivelul imediat superior și se testează, continuând integrarea modulelor spre partea superioară a structurii 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 și 3;
 - clusterii se testează folosind driver-ele D1, D2 și D3;
 - se înlătură D1 și D2, iar clusterul 1 și clusterul 2 sunt integrați în Ma;
 - similar, D3 este înlăturat iar clusterul 3 este integrat cu modulul Mb.
 - Ma și Mb se vor integra în Mc.

Integrare Sandwich. Descriere

- **integrare sandwich** (*engl. sandwich integration*):
 - permite construirea și testarea programului combinând abordările de integrare *top-down* și *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 interfeței modulelor. Definiție. Clasificare

- **testarea interfeței modulelor** (*engl. interface integration testing*):
 - se realizează la integrarea mai multor module pentru obținerea unor sisteme de dimensiuni mai mari;
- relevantă în dezvoltarea orientată pe obiecte, comportamentul acestora fiind descris prin intermediul interfețelor;
- **objective**: identificarea defectelor care pot apărea la utilizarea unei interfețe a unui alt modul sau false presupuneri legate de interfața unui modul;
- **comunicarea între module** poate fi bazată pe:
 - **parametri**: la transmiterea datelor de la o metodă la alta;
 - **memorie partajată**: o zonă de memorie 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 interfeței modulelor. Tipuri de bug-uri identificate

- tipuri de bug-uri [[NT2005](#), pagina 161]:
 - utilizarea greșită a interfeței (*engl. interface misuse*):
 - un modul apelează un alt modul eronat, e.g., parametrii sunt transmiși în ordinea greșită;
 - înțelegerea greșită a semnificației interfeței (*engl. interface misunderstanding*):
 - un modul face presupuneri greșite referitoare la comportamentul unui alt modul apelat, e.g., se interpretează că mai mulți parametri de același tip au altă semnificație;
 - greșeli de sincronizare (*engl. timing errors*):
 - modulul apelat și modulul apelant folosesc unități de timp diferite, e.g., minute, secunde, milisecunde, iar informația obținută își pierde acuratețea.

Testare de integrare. Reguli generale de aplicare

- se studiază arhitectura aplicației 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 execuția testelor de integrare, i.e., unit testing;
- se utilizează **documentația care descrie interacțiunile dintre fiecare două module** (interfețele acestora și modul de colaborare).

TESTARE DE SISTEM

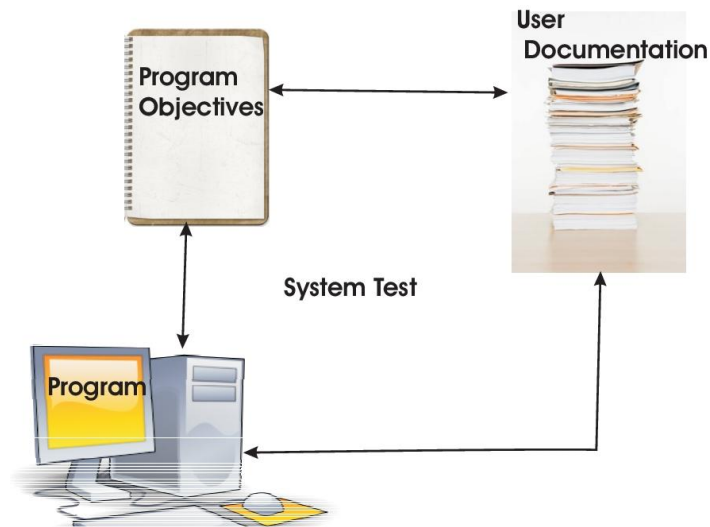
Definiție. Caracteristici

Testare funcțională

Testare non-funcțională

Testare de sistem. Definiție. Caracteristici

- **testare de sistem** (*engl. system testing*):
 - verifică dacă produsul dezvoltat respectă obiectivele inițiale;
- elaborarea cazurilor de testare se bazează pe:
 - **documentul cu specificațiile sistemului?**
 - **nu**, deoarece pot apărea erori în procesul de transpunere a obiectivelor în specificații externe;
 - **documentele cu obiectivele stabilite?**
 - **nu**, deoarece nu conțin descrierea exactă a interfețelor externe ale programului;
 - obiectivele nu oferă informații cu privire la funcționalitatea sistemului (interfețe ale



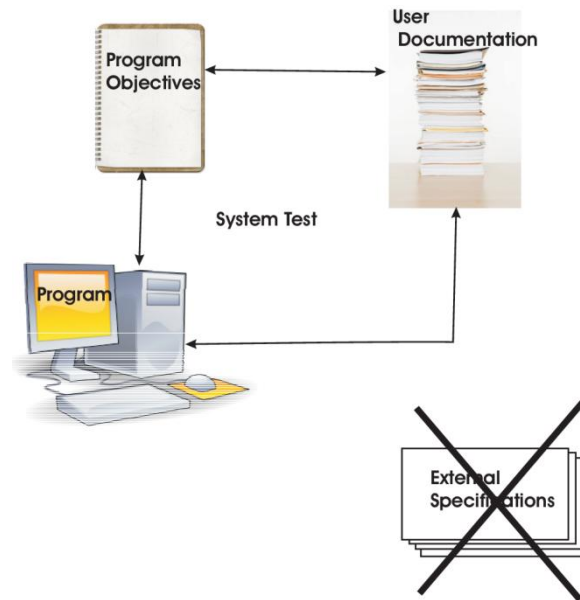
- **documentația de utilizare a produsului soft?**
 - **da**; proiectarea testării de sistem – pe baza analizei obiectivelor;
 - descrierea cazurilor de testare – pe baza analizei documentației de utilizare.

Testare de sistem. Caracteristici

- nu există metodologie pentru proiectarea cazurilor de testare în testarea de sistem;
 - se proiectează două categorii de teste asociate:

- cerințele funcționale
==> **testare funcțională**;

- cerințele non-funcționale
==> **testare non-funcțională**
(*engl. non-functional testing*);



- la acest nivel de testare, proiectarea cazurilor de testare este determinată de obiectivele de testare și de strategia aleasă, evidențiind creativitatea și experiența.

Testare de sistem. Testare funcțională a sistemului

- **testare funcțională a sistemului** (*engl. function/functional testing, use case testing*):
 - testarea cerințelor descrise în specificațiile sistemului;
 - proces care identifică neconcordanțele existente între comportamentul programului și specificația acestuia, **din punctul de vedere al utilizatorului**;
- elaborarea cazurilor de testare se bazează pe **criteriul black-box**;
- folosește specificația sistemului.

Testare funcțională a sistemului. Proiectarea cazurilor de testare

- **caz de utilizare vs. caz de testare:**

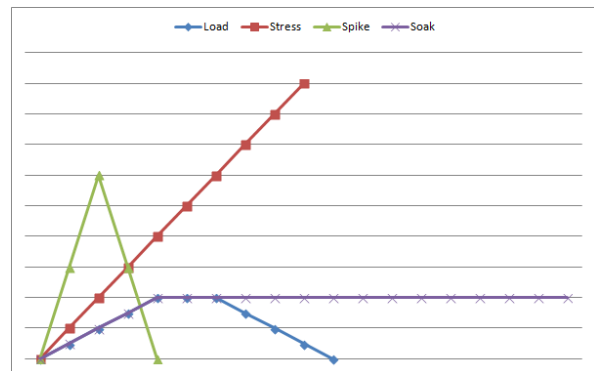
- cazurile de utilizare descriu fluxul de execuție al unui proces în cadrul sistemului, folosind **scenarii** de utilizare frecventă;
- **cazurile de testare sunt derivate din cazurile de utilizare** și pot indica prezența unui bug la utilizarea concretă a sistemului;
- fiecare caz de testare se bazează de obicei pe un scenariu și acoperă diferite ramificații de execuție (indicate prin cazuri speciale de date de intrare sau condiții speciale);
- pentru un caz de testare specifică:
 - **toate condițiile** care este necesar să fie satisfăcute pentru execuția cazului de utilizare cu succes;
 - **postcondițiile** impuse asupra rezultatelor obținute;
 - **descrierea stării finale a sistemului** după ce testarea cazului de utilizare s-a realizat cu succes;

Testare de sistem. Testare non-funcțională. Definiție

- **testare non-funcțională** (*engl. non-functional testing*):
 - verifică modul în care sistemul îndeplinește cerințele non-funcționale;
 - stabilește dacă sistemul este pregătit pentru a fi efectiv utilizat;
- în [[Myers2004](#), Cap 6] sunt descrise 15 tipuri de testare (non-funcțională) 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-funcțională (1)

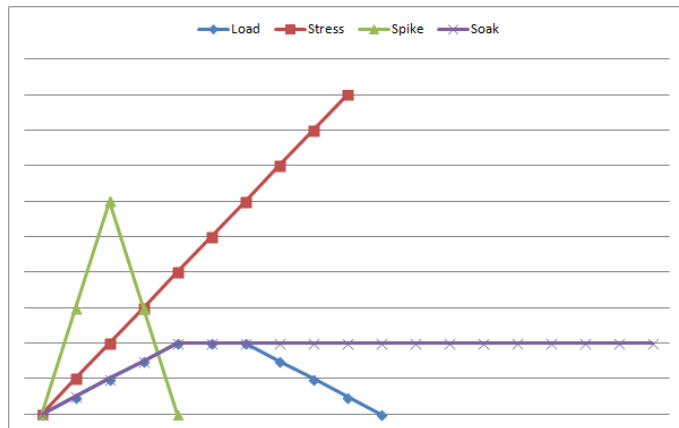
- **performance testing:** - cât de bine face sistemul ceea ce trebuie să facă
 - fiecare sistem are cerințe referitoare la performanța utilizării;
 - e.g., execuția unei funcționalități să fie realizată într-un anumit interval de timp, iar resursele să nu fie infinite;
 - bug-urile legate de performanță indică deficiențe la nivel de proiectare care determină degradarea performanței sistemului soft în timpul utilizării;
 - **obiectiv:** identificarea blocajelor determinate de reducerea performanței componentelor și 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-funcțională (2)

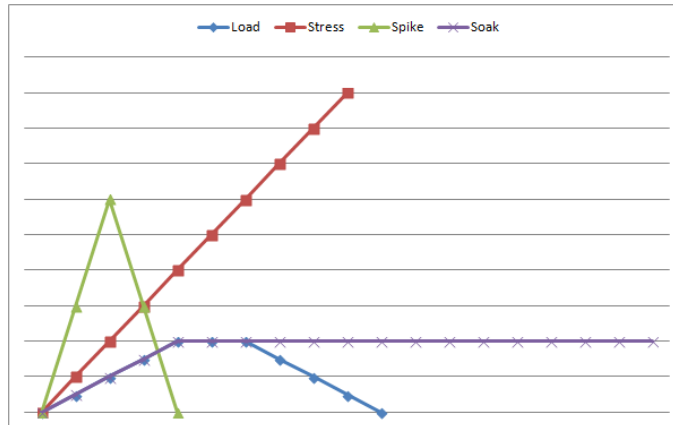
- **capacity testing:**

- permite prevenirea eventualelor probleme determinate de creșterea numărului de utilizatori sau a volumului de date;
- **obiectiv:** obținerea de informații referitoare la **nivelul maxim de utilizatori și/sau date care nu afectează performanța:**
 - E.g.: câte fișiere pot fi folosite fără a afecta performanța.



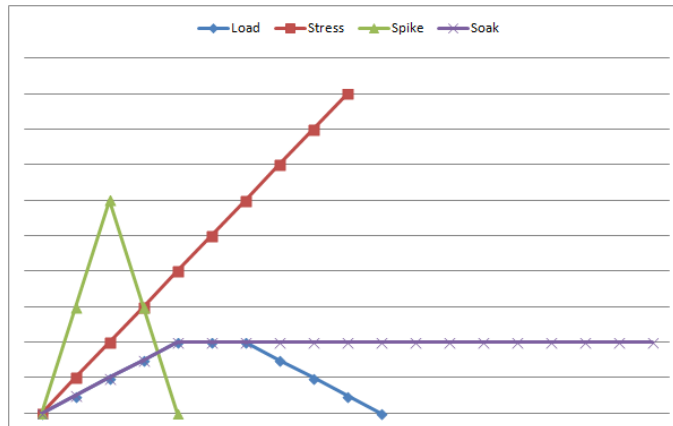
Testare de sistem. Tipuri de testare non-funcțională (3)

- **load testing:**
 - evaluează capacitatea sistemului de opera cu volume de date normale sau în condiții de solicitare (vârf);
 - **obiectiv:** obținerea de informații referitoare la comportamentul sistemului în **anumite condiții de utilizare, i.e., normale și de solicitare:**
 - E.g.: creșterea numărului de fișiere folosite.



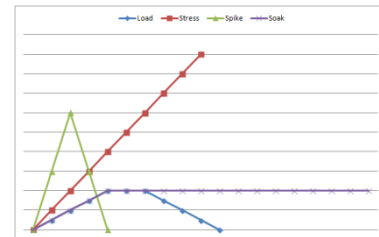
Testare de sistem. Tipuri de testare non-funcțională (4)

- **volume testing:**
 - evaluează comportamentul sistemului atunci când se gestionează volume mari de date;
 - **obiectiv:** obținerea de informații legate de **funcționarea aplicației cu un volum de date ridicat**:
 - E.g.: creșterea dimensiunii fișierelor folosite.



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

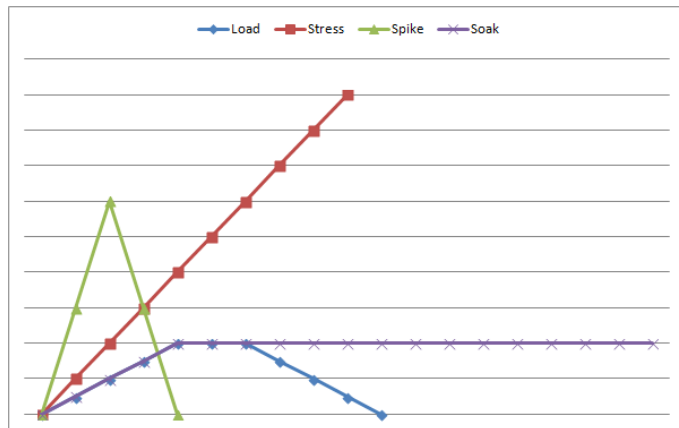
- **stress testing:** - pune presiune asupra limitelor sistemului
 - investighează dacă comportamentul softului se degradează în condiții extreme de utilizare și când nu are acces la resursele necesare;
 - **este de dorit ca degradarea softului cauzată de creșterea numărului de cereri să se realizeze acceptabil, fără să ducă la un eșec imediat în timpul testării; sistemul nu trebuie să eșueze catastrofal;**
 - relevant pentru sistemele distribuite care pot indica o degradare severă atunci când rețeaua devine încărcată;
 - **obiectiv:** obținerea de informații despre **modul în care sistemul funcționează în condiții extreme, dincolo de limitele normale:**
 - E.g.: creșterea numărului de fișiere folosite până la failure și chiar mai mult;
 - verifică dacă are loc o pierdere inacceptabilă de date și/sau imposibilitatea de a utiliza anumite servicii.



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

- **soak testing:**

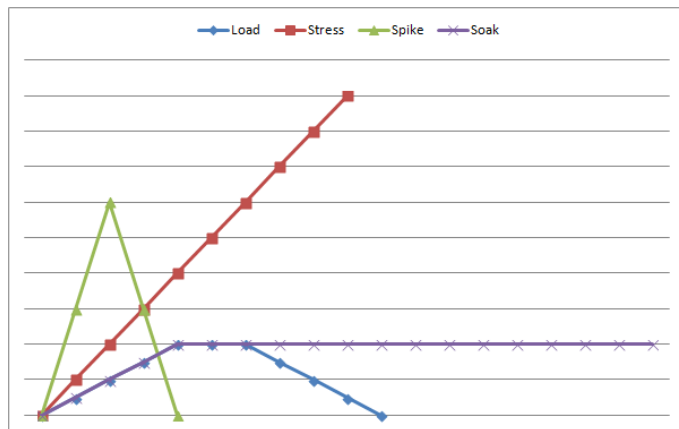
- verifică dacă sistemul face față unui volum mari de date pentru o perioadă mai mare de timp;
- **obiectiv:** obținerea de informații legate de **posibilitatea de a utiliza volum de date și care sunt consecințele dacă se depășește vomulul pentru care a fost proiectat:**
 - E.g.: verifică dacă creșterea dimensiunii fișierelor folosite este suportată.



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

- **spike testing:**

- evaluează comportamentul sistemului la schimbări bruște și extreme a condițiilor de lucru, i.e., puține sarcini, multe sarcini;
- **obiectiv:** obținerea de informații legate de **punctele slabe ale sistemului determinate de modificarea rapidă a condițiilor de lucru:**
 - E.g.: creșterea brusc și diminuarea rapidă a numărului de fișiere cu care se lucrează.



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

- **reliability testing:**

- investighează capacitatea un soft de a funcționa conform așteptărilor utilizatorului, chiar și atunci când eșuează;
- **determină cât timp și cât de eficient poate funcționa sistemul fără eroare;**

- **security testing:**

- calitatea, securitatea și gradul de încredere în aplicație (*engl. **reliability***) sunt caracteristici dependente;
- defectele produsului soft pot fi exploatare pentru a identifica breșe de securitate;
- **presupune simularea unor atacuri la nivel de securitate, având scopul de a identifica vulnerabilitățile softului.**

TESTARE DE ACCEPTARE

Definiție. Caracteristici. Etape de realizare. Clasificare

Alpha Testing. Beta Testing

Alpha Testing vs Beta Testing

Alte tipuri de testare de acceptare

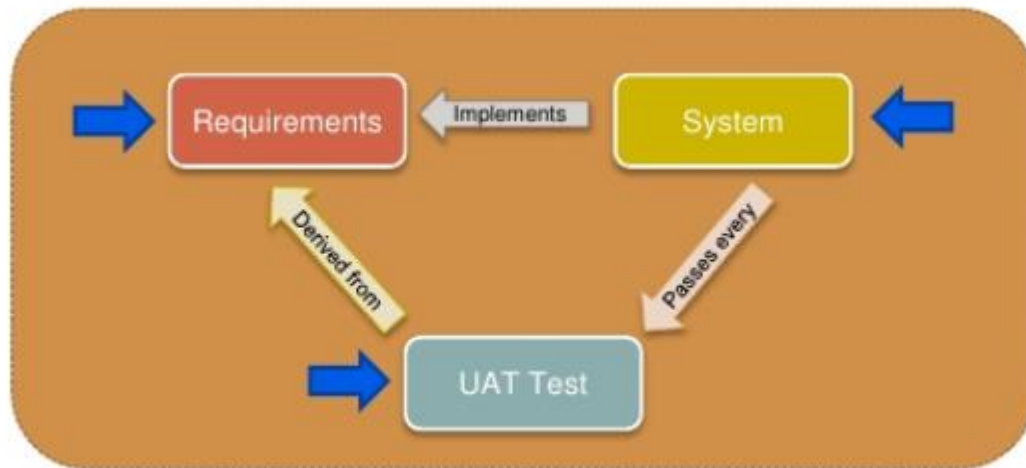
Dificultăți de testare

Testare de acceptare. Definiție. Clasificare

- **testare de acceptare** (*engl. user acceptance testing, UAT*):
 - procesul de testare prin care se verifică dacă programul îndeplinește cerințele inițiale și 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 funcțional;
 - **creare unei suite de cazuri de testare** pentru UAT;
 - **rularea testelor UAT**;
 - **evaluarea și 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 folosește 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 și eliminate/rezolvate imediat ;	clientul inventariază dificultățile de utilizare ale produsului soft și le raportează programatorului pentru a fi rezolvate .

Testare de acceptare. Alpha Testing vs Beta Testing

	Alpha testing	Beta testing
Tipuri de testare ?	reliability testing și security testing NU se realizează în profunzime;	realibility testing, security testing și robustness testing se realizează în detaliu;
Eliminare bug-uri ?	deficiențele majore pot fi rezolvate imediat de programator;	deficiențele raportate sunt investigate ulterior , iar îmbunătățirile și corecturile se regăsesc în versiunile ulterioare ale produsului soft;
Relevanță?	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 și specificații care sunt precizate într-un **contract de colaborare** între dezvoltator și client;
 - criteriile și specificațiile de acceptanță sunt stabilite la semnarea contractului, **înainte** de dezvoltarea softului;
- **regulation acceptance testing, i.e., compliance acceptance testing:**
 - verifică dacă produsul soft dezvoltat respectă regulamentele și legile în vigoare referitoare de utilizarea și funcționarea unui produs soft specific;
- **operational acceptance testing, i.e., operational readiness testing, production acceptance testing:**
 - verifică dacă există fluxurile informaționale necesare pentru utilizarea produsului soft de către client (e.g., planuri de backup, training pentru utilizatori, diferite procese de întreținere și verificări de securitate).

Testare de acceptare. Dificultăți de testare

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

TIP DE TESTARE VS NIVEL DE TESTARE

Tip de testare. Nivel de testare. Definiție

Obiective de testare. Exemple

Retestare. Definiție

Testare de regresie. Definiție

Retestare vs Testare de regresie

Tip de testare. Nivel de testare. Definiție

- **nivel de testare** (*engl. testing level*):
 - o serie de activități 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-funcționale:**
 - se realizează prin aplicarea unui anumit tip de testare, e.g., testare de performanță, testare de utilizabilitate, cu scopul de a evalua o caracteristică a calității produsului soft, la nivelul *testării de sistem*;
 - **testarea după eliminarea unui bug:**
 - se realizează 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*.

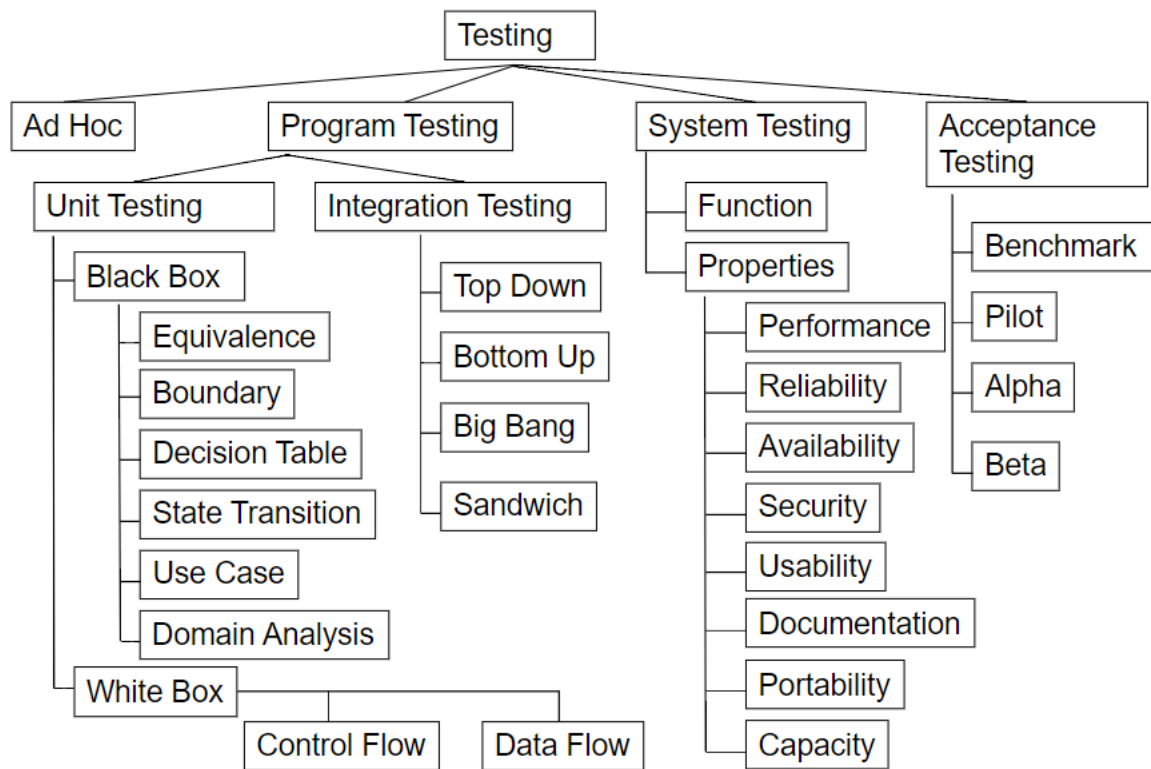
Re-testare. Definiție

- **re-testare** (*engl. re-testing, confirmation testing*):
 - re-execuția testelor care au pus în evidență 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. Definiție

- **testare de regresie** (*engl. regression testing*):
 - re-execuția 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 în teste de regresie, care permit testarea:
 - tuturor funcționalităților sistemului;
 - funcționalităților 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. Definiții și caracteristici:**
 - testare unitară;
 - testare de integrare;
 - 4 strategii (big-bang, top-down, bottom-up, sandwich), descriere, comparare;
 - testare de sistem;
 - testare funcțională;
 - 5 tipuri de testare non-funcțională (volume, stress, load, usability, security) [\[Mye04\]](#).
 - testare de acceptare;
 - alpha testing, beta testing.
- **Tip de testare vs Nivel de testare. Definiții și 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: Marți, 25 Martie 2025;**
 - **Orele: 08:00-10:00;**
 - **Desfășurare: Sala 6/II (Gh. Călugareanu), Clădirea Centrală a UBB.**

Cursul următor...

- **Curs 06:**
 - Demo Mockito

Seminar 04. Create and Solve a Puzzle (1)

- **Create and Solve a Puzzle**
 - **termen:** **02 mai 2025, orele 20:00;**
 - **echipe:** max. 3 studenți/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** pentru activitatea **Games** fiecărui membru din echipă;
 - **Task 01. Create a Puzzle:** cu min. 20 concepte studiate la VVSS;
 - se va accesa pagina <https://www.puzzle-maker.com/> și se va alege tipul de joc **crossword**;
 - se va crea un puzzle folosind min. 20 termeni studiați în cadrul disciplinei VVSS;
 - se va genera puzzle-ul pe fundal alb și se va posta pe channel-ul **#Games**;
 - echipa se va înscrie în fișierul [Seminar04-Games](#) pentru a putea primi punctajul pentru crearea puzzle-ului;
 - pentru acest task se poate folosi orice alt tool care permite obținerea 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 unul sau mai multe puzzle-uri propuse de alte echipe; task-ul se va puncta o singură dată, adică doar una dintre soluțiile propuse (cea cu punctaj maxim) va primi puncte;
 - o echipă poate rezolva un puzzle chiar dacă nu a propus anterior un puzzle;
 - după rezolvarea puzzle-ului (prin editarea fișierului 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 primește 0.10 puncte (0.10x20 întrebări = 2 puncte);
 - echipa (A, B, C) evaluează soluția primită și completează în fișierul [Seminar04-Games](#) componenta echipei care a oferit soluția și punctajul corespunzător (e.g., 2 puncte, 1.8. puncte);
 - după ce au fost evaluate soluții oferite de max. 6 echipe participante, echipa care a propus puzzle-ul va posta soluția la puzzle în thread-ul postării inițiale pe channel-ul **#Games**, indicând prin aceasta faptul că nu mai poate primi spre evaluare alte soluții.

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>