Testarea pe parcursul dezvoltării programelor

Prof. univ. dr. ing. Florica Moldoveanu

Curs *Ingineria programelor* – UPB, Automatică şi Calculatoare
2020-2021

Definitii (1)

Scopul testarii: verificarea si validarea programelor.

Testarea in vederea verificarii programului:

- **Este implementarea produsului în conformitate cu specificația sa?**
- Efectuata la mai multe niveluri: unitate functionala, clasa, componenta, subsistem, sistem.
- Sunt verificate atât proprietățile interne cât şi cele externe ale produsului.

Testarea in vederea validarii programului, numita si testare de acceptare:

- **Este produsul în conformitate cu cerințele utilizatorilor/tuturor părților interesate?**
- Se efectueaza in mediul real de operare al programului cu participarea utilizatorilor finali, a clientului şi a celorlalte părți interesate in proiect
- Sunt verificate proprietățile externe ale produsului

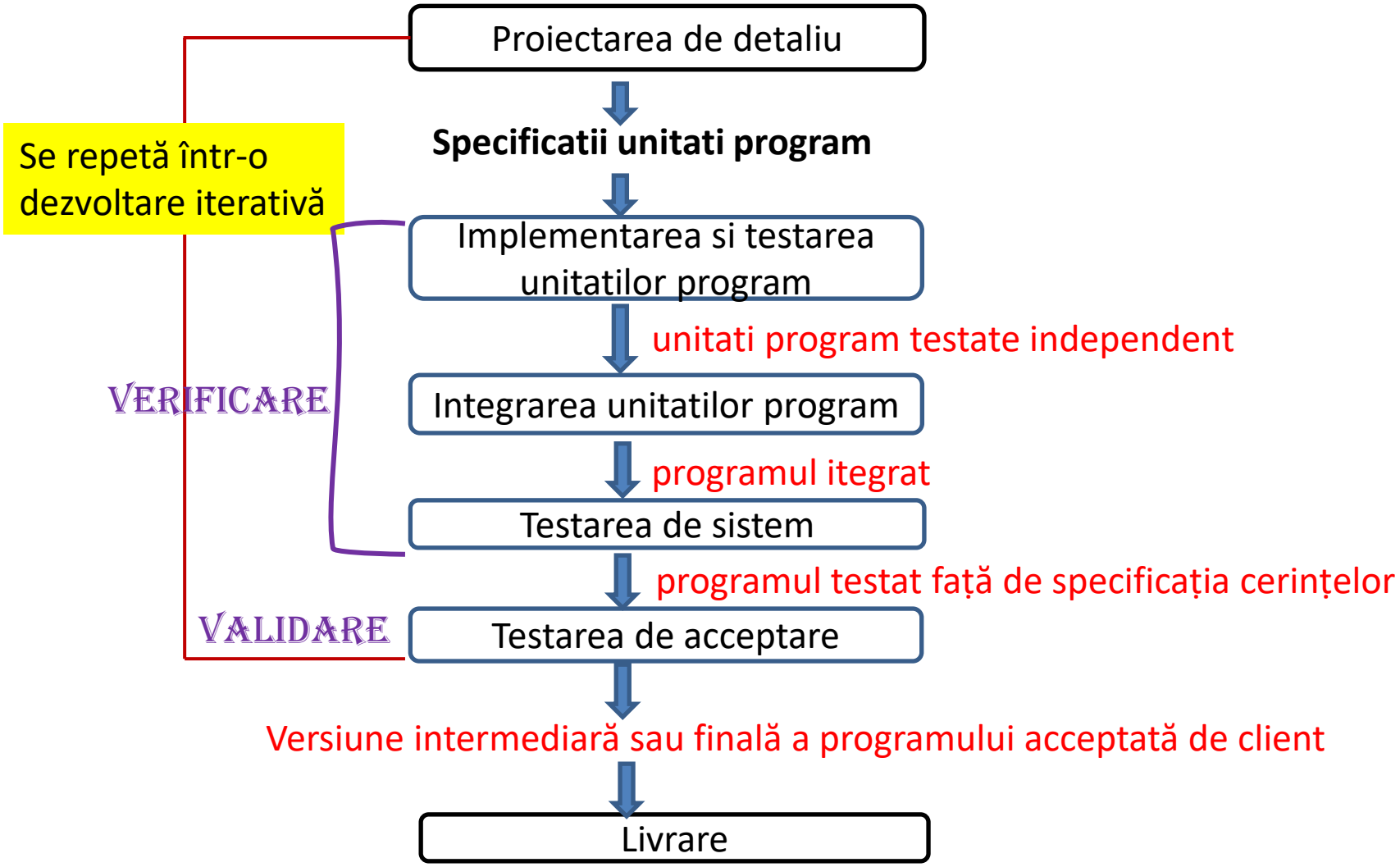
Definitii (2)

- Un caz de test (test case) specifica o executie de test a unui program, prin:
  - Datele de intrare – numite si date de test
  - Conditiile de executie (ex, preconditii)
  - Procedura de executie a testului (ex, ordinea in care se introduc datele de intrare)
  - Rezultatele asteptate
- O suita de test (test suite): un set de cazuri de test.
- Testarea: activitatea de conceptie a cazurilor de test, de executie a cazurilor de test si de evaluare a rezultatelor executiilor de test, in diferite etape ale ciclului de viata al unui program.
- Cădere (Eroare la o executie) = orice rezultat neconform cu specificatia.
- Defect = "bug" : o anomalie în produs (greşeala logică/ codificare) care are ca efect o cădere.

➢ Prin testare nu se poate demonstra corectitudinea unui program: nu poate fi executat pentru toate seturile de date de intrare posibile.

➢ Cazurile de test se aleg a.i. sa se verifice satisfacerea specificatiilor si descoperirea defectelor.

Testele pe parcursul dezvoltarii unui produs software

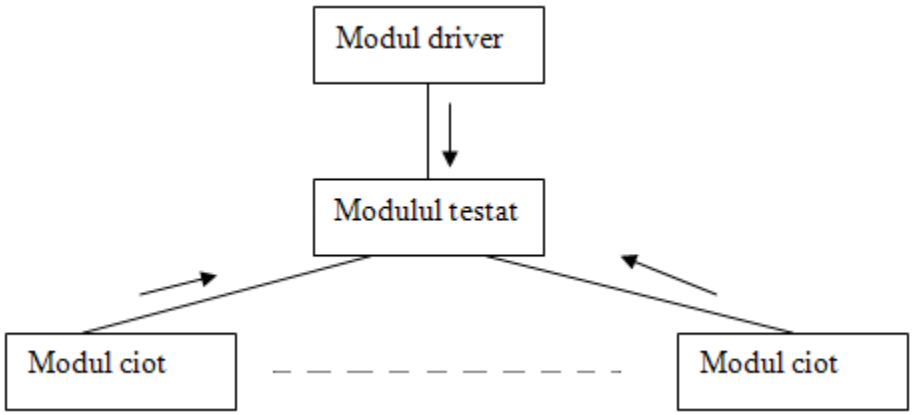


Testele unitare (1)

- Se efectueaza la nivelul unei unitati functionale de program: functie, procedura, functie membru a unei clase.
- Sunt realizate de programatorul care implementeaza unitatea program.

- Unitatea testata este tratata ca o entitate independenta, care nu necesita prezenta altor unitati ale programului.
- Unitatile program cu care interactioneaza unitatea testata sunt simulate prin: module « stub » (ciot) si un modul « driver ».
- Modulele ciot simuleaza modulele apelate de modulul testat. Un modul "ciot" are aceeasi interfata cu modulul real si realizeaza in mod simplificat functia sa.
- Modulul driver simuleaza cazurile de apel ale modulului testat de catre celelalte module ale programului.

Testele unitare (2)



Structura programului executabil pentru testarea izolata a unei unitati program.

- Fiecare executie este specificata printr-un caz de test
- La o executie modulul driver apeleaza modulul testat cu datele de intrare ale cazului de test.

Datele de intrare ale fiecarui caz de test pot fi generate de modulul driver, pot fi preluate dintr-un fisier sau furnizate de tester intr-o maniera interactiva.

- Rezultatele fiecarei executii sunt afisate sau scrise intr-un fisier pentru a fi comparate ulterior cu rezultatele asteptate.
- Cazurile de test in testarea unitara pot fi specificate printr-un tabel de perechi: intrari – rezultate asteptate.

Testele unitare (3)

Exemplu de modul ciot:

Modulul testat apeleaza o functie de sortare a unui vector, cu antetul:

```
void sortare(int n, int *lista);
```

Functia poate fi simulata prin urmatoarea functie ciot:

```
void sortare(int n, int *lista)
        { int i;
          printf(" \n Lista de sortat este:");
          for(i=0; i<n; i++)  printf("%d", lista[i]); //afiseaza lista nesortata
         for(i=0; i<n; i++)  scanf("%d", lista[i]); // se citeste lista sortata, furnizata de tester
         }
```

Modulele ciot si modele driver folosite in testele unitare se pastreaza pentru a fi folosite in **testele regresive (teste efectuate dupa corectarea unor defecte)**.

Testele unitare(4)

Metode de alegere a datelor de test pentru testarea unitara:

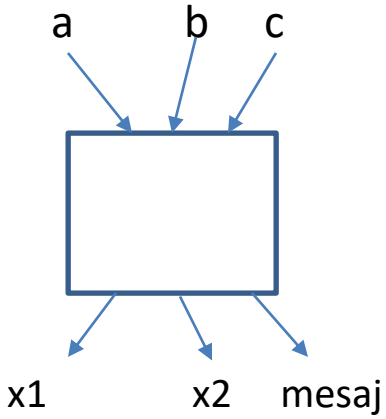
- **Testarea functionala ("black box")**: datele de test se aleg pe baza specificatiei unitatii testate
- **Testarea structurala ("white box)**: datele de test se aleg pe baza codului unitatii testate:
  - Testarea fluxului controlului
  - Testarea fluxului datelor
- **Alte metode**:
  - **Testarea random:** datele de test sunt generate in mod aleator pe baza domeniilor de valori ale variabilelor de intrare
  - **Testarea bazata pe mutatii (Mutation testing):**
    - sunt introduse defecte in cod
    - o suita de test care nu detecteaza defectul introdus este considerata ineficienta

Alegerea datelor de test

Testarea unui program pentru rezolvarea ecuatiilor de grad 2

$a \cdot x^2 + b \cdot x + c = 0$

Testarea functionala
(pe baza specificatiei)

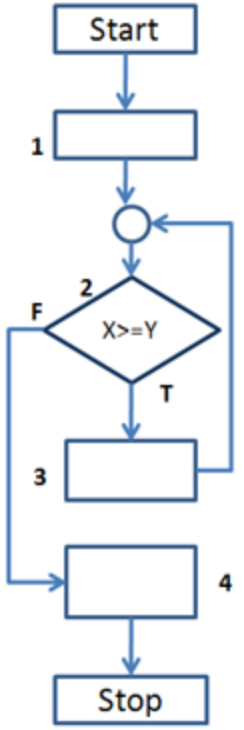


Se aleg cel putin 2 cazuri de test:

1) Valorile pt a,b,c astfel incat sa rezulte radacini reale
   mesajul afisat : radacini reale
   valorile variabilelor x1, x2: cele 2 radacini asteptate
2) Valorile pt a,b,c a.i. sa rezulte radacini complexe
   mesajul afisat : radacini complexe
   valorile variabilelor x1, x2: valorile asteptate pt partea reala si partea imaginara

Testarea structurala
(pe baza codului)

Se aleg valorile pt a,b,c pe baza unui criteriu de acoperire a GFC.



Graful fluxului controlului (GFC)

Testarea claselor (1)

- **Testarea unei clase presupune:**
  - testarea separata a fiecarei functii membru
  - testarea comportarii in timp a obiectelor clasei:

    - se poate folosi o diagrama de stari UML care reda toate starile obiectelor clasei de la creare pana la distrugere

Testarea "Alpha-Omega" – cazurile de test se aleg a.î. un obiect al clasei testate sa treaca prin toate starile sale, de la creare pana la distrugere.

- Fiecare executie de test corespunde unei căi de la nodul de intrare la nodul de iesire in graful starilor.
- Suita de teste trebuie sa acopere executia tuturor metodelor clasei, fiecare cel putin odata.
- Este o testare minimala a unei clase: este urmata de alte teste, bazate pe specificatia functionala a clasei sau pe acoperirea codului.

Testarea claselor (2)

- **Ordinea de apel a metodelor unei clase la o executie de test depinde de tipul clasei**:
  - "Non-modal" – clasa care accepta orice mesaj (apel) in orice stare: ordinea de apel oarecare
  - "Quasi-modal" - clasa in care constrangerile privind ordinea mesajelor se schimba in acelasi timp cu starea obiectelor clasei. Exemplu: clasele de tip container sau colectie.
    - Se doreste acoperire tip "orice mesaj in orice stare" (de ex. push() pentru o stiva plina).
  - "Modal" - clasa care are constrangeri permanente si fixe privind ordinea mesajelor.
    - Prin suita de teste trebuie sa se asigure trecerea unui obiect prin toate starile si toate tranzitiile ("acoperirea" tuturor starilor si a tranzitiilor).

Platforme de testare automata a claselor:

- JUnit: platforma (framework) de testare pentru limbajul Java (http://www.junit.org).
- Nunit (pentru C#), PyUnit (pentru Pyton), fUnit (pentru Fortran), CPPUnit (pentru C++), s.a., denumite in mod colectiv xUnit.

Testarea de integrare (1)

Scopul : integrarea modulelor (unitati functionale/clase) dezvoltate si testate independent, intr-un sistem functional.

- Se verifica interactiunile dintre module, grupuri de module, subsisteme, pana la nivel de sistem.
- Clasele sunt testate independent, prin testele unitare. Prin testele de integrare se verifica interactiunea dintre functiile claselor prin diferite suite de test.
- Testele de integrare presupun, ca si testele unitare, realizarea de module "ciot " si module "driver". Numarul de module "driver " şi de module "ciot" necesare in testele de integrare depinde de ordinea in care sunt integrate modulele.
- Testele de integrare necesita instrumente de gestiune a versiunilor si a configuratiilor.

Metoda "big-bang“

- Sunt integrate intr-un program executabil toate modulele existente la un moment dat. Modulele "driver " şi "ciot" necesare sunt de asemenea integrate. Metoda este periculoasa caci toate defectele existente se manifesta in acelasi timp si localizarea lor este dificila.

Testarea de integrare (2)

Integrarea progresiva

- Integrarea modulelor existente in mai multi pasi
- In fiecare pas se adauga ansamblului de module integrate numai un singur modul sau 2-3 module corelate. Astfel, defectele care apar la o executie de test sunt provocate de ultimul modul integrat.

Integrarea ascendenta ("de jos in sus")

- Se incepe prin testarea modulelor simple, care nu apeleaza alte module, apoi se adauga progresiv module care apeleaza numai modulele deja integrate, pana cand este asamblat intregul sistem.
- Metoda necesita implementarea unui modul "driver" pentru fiecare modul/grup de module corelate (si nici un modul "ciot").

Avantajele integrarii de jos in sus

- Nu sunt necesare module "ciot". Modulele "driver" se implementeaza mult mai usor decat modulele "ciot". Exista chiar instrumente care produc automat module "driver".
- De regula modulele de nivel coborat sunt apelate de multe alte module:→ este mai avantajos sa fie implementate decat sa fie simulate prin module ciot.

Testele de integrare(3)

Dezavantajele integrarii de jos in sus

- Modulele principale ale programului, care controleaza fluxul aplicatiei, sunt testate la sfarsit. Descoperirea unor defecte in testarea lor, in general, conduce la reproiectare si reimplementarea unor module deja integrate.
- Un prototip timpuriu al aplicatiei nu este posibil.

Integrarea descendenta ("de sus in jos")

- Se incepe prin testarea modulului principal, care controleaza fluxul controlului in aplicatie, in pasii urmatori se adauga modulele apelate direct de el, si asa mai departe.
- **Metoda presupune implementarea unui singur modul "driver" (care apeleaza modulul principal) si a unui modul "ciot" pentru fiecare alt modul al programului.**

Testarea de integrare(4)

- Integrarea descendenta se efectueaza intr-un proces de dezvoltare descendenta.
- Fiecare modul este testat imediat dupa ce a fost implementat, moment in care nu au fost inca implementate modulele apelate de el.
- In fiecare pas este inlocuit un singur modul "ciot" cu cel real.

Avantajele integrarii descendente

- **Erorile de proiectare sunt descoperite timpuriu**, la inceputul procesului de integrare, atunci când sunt testate modulele principale ale programului→ se evita reproiectarea si reimplementarea majoritatii componentelor de nivel mai coborât, asa cum se intâmpla cand erorile respective sunt descoperite la sfârsitul procesului de integrare.
- **Programul obtinut este mai fiabil** caci modulele principale sunt cel mai mult testate.
- Prin integrarea modulelor de nivel superior se poate considera **ca sistemul în ansamblul său exista dintr-o faza timpurie a dezvoltarii** si deci se poate exersa cu el in vederea validarii.

Testele de integrare(5)

Dezavantajele integrarii descendente

- Este necesar sa se implementeze cate un modul "ciot" pentru fiecare modul al programului, cu exceptia modulului principal.
- Este dificil de simulat prin module "ciot" componente complexe si componente care contin in interfață structuri de date.
- In testarea componentelor principale, care de regula nu afiseaza rezultate, este necesar sa se introduca instructiuni de afisare, care apoi sunt extrase, ceea ce presupune o noua testare a modulelor.

Testele de integrare(6)

Tehnici de integrare hibride

- Integrarea nu trebuie sa fie strict descendenta sau strict ascendenta.
- Dezavantajele integrarii descendente pot fi reduse aplicand tehnici hibride, de exemplu, folosind in locul unor module "ciot", modulele reale testate.
- Experienta arata ca este foarte util sa se inceapa prin integrarea modulelor de interfata utilizator. Aceasta permite continuarea integrarii in conditii mai bune de observare a comportarii programului.

Integrarea « sandwich »

- Se integreaza modulele de pe nivelul ierarhic cel mai coborat (cel mai mult apelate) prin tehnica « de jos in sus »
- Se integreaza modulele de pe nivelul ierarhic cel mai ridicat prin tehnica « de sus in jos »
- Restul modulelor se integreaza intr-o ordine convenabila.

Testarea de sistem (1)

Obiectivul: se verifică dacă produsul obtinut (hardware+software) satisface specificația cerințelor; dacă este o versiune intermediară, se verifică satisfacerea cerințelor acelei versiuni.

- Sunt teste ale sistemului de programe şi echipamente. Poate fi necesar ca sistemul să fie instalat şi testat în mediul său real de operare ; dacă nu este posibil, se simuleaza mediul real.
- Adesea, testele de sistem ocupă cel mai mult timp din întreaga perioadă de testare.
- Toate testele sunt de tip «cutie neagră» (cazurile de test sunt alese pe baza specificației, fără cunoaşterea codului).
- Este efectuată de o echipă a companiei dezvoltatoare.

Testele de sistem: derivate din cerințele funcționale şi nefuncționale (fiabilitate, portabilitate, adaptabilitate, performanţă, disponibilitate, securitate, siguranta in functionare)

Teste funcționale - derivate din cerințele funcționale

- Cazurile de test se definesc plecand de la cazurile de utilizare
- Includ şi testele de interfață utilizator
- In funcție de tipul sistemului, testele funcționale pot fi combinate cu alte tipuri de teste, derivate din cerințele nefuncționale.

Testarea de sistem(2)

Teste de performanța

- Se verifica daca performanta sistemului corespunde celei cerute:
  - In conditii de operare normale (performanta nominala)
  - In conditiile cele mai defavorabile

  Parametrii de performanta masurati difera de la sistem la sistem. Exemple:

  - “timpul de raspuns trebuie sa fie sub o milisecunda, 90% din timpul operarii”
  - “timpul de raspuns la o tranzactie on-line sa fie sub o secunda”

Teste de comunicare (interfete externe)

- Se verifica fluxul datelor si al controlului prin interfetele externe
- Protocoalele de comunicatie

Teste ale utilizarii resurselor

- Utilizare CPU, memorie interna, spatiu pe disc, utilizarea retelei

Testarea de sistem(3)

Teste de fiabilitate

- Se verifica satisfacerea cerintelor de fiabilitate, de ex. MTBF (Mean Time Between Failures) – perioada de timp medie intre 2 caderi (functionari neconforme cu specificatia)
- **Se folosesc teste statistice operationale**: datele de test sunt generate aleator, conform unei legi de probabilitate care corespunde profilului operational (de utilizare) al sistemului.
- Testarea de fiabilitate se termina atunci cand rata caderilor ajunge la valoarea ceruta.

Teste de securitate

- Se verifica daca sistemul este protejat impotriva atacurilor:
  - Acces neautorizat la resursele sau functiile sistemului
  - Incercare de acces la fisierele altor utilizatori
  - Oprirea executiei proceselor rulate de alti utilizatori
  - Securitatea comunicarii in retea

Testarea de sistem(4)

Teste de portabilitate

- Se ruleaza teste selective pe toate platformele prevazute

Teste de siguranta in functionare

- Se verifica daca sistemul protejeaza utilizatorii sai impotriva daunelor provocate de caderi ale sistemului:
  - Nu se pierd fisiere deschise
  - Nu se pierd informatii din baza de date
  - Nu se pierd informatii de care depinde buna functionare a sistemului
- Se fac teste cauzand probleme, cum ar fi: intreruperea curentului electric, inchiderea calculatorului in timpul unor operatii, etc.

Testarea de sistem(5)

Teste de stres

- Sistemul este pus in situatii de operare exceptionale: rata foarte mare a intrarilor, numar de utilizatori foarte mare conectati simultan la sistem, consum maxim de resurse, trafic maximal in retea, etc.
- Se urmareste evaluarea limitelor de performanta ale sistemului.
- **Sunt teste de robustete**: sistemul nu trebuie sa se degradeze, ci sa-si mentina starea de functionare in conditii neprevazute la crearea sa.

Testarea de acceptare (1)

- **Scopul: verificarea conformitatii produsului obtinut cu produsul solicitat, conform contractului cu clientul** (Specificatia cerintelor utilizator).

- Clientul şi utilizatorii finali participă la efectuarea acestor teste.
- Uneori testele de acceptare sunt conduse de client.
- Se efectuează în mediul operațional al produsului.
- Pentru unele produse software, testarea de acceptare a versiunii finale are loc în două etape:

  Testarea alfa: se efectuează folosindu-se specificatia cerintelor utilizator, până cand clientul şi dezvoltatorul cad de acord că programul este o reprezentare satisfăcătoare a cerințelor.

  Testarea beta: programul este distribuit unor utilizatori selecționați, realizandu-se astfel testarea lui în condiții reale de utilizare → se reduce riscul căderilor software în timpul operării şi se asigură o calitate crescută a produsului software prin validarea de către utilizatorii finali.

Testarea de acceptare (2)

❖ Criteriul de acceptare: produsul trebuie să aibă funcționalitățile şi calitățile asteptate de client şi utilizatorii țintă.

- De ce este necesară testarea de acceptare după ce sistemul a fost deja testat pentru conformitatea cu specificatia cerintelor?

1. Documentul de specificație a cerințelor cuprinde o descriere detaliata a cerințelor viitorului produs, realizata de dezvoltator dupa extragerea cerințelor utilizator, pe baza acestor cerințe (care pot fi incluse în acelasi document).

Unele cerințe utilizator pot fi înțelese de dezvoltator în mod diferit de fața de client.

→ prin testarea de acceptare se verifică faptul că cerințele utilizator au fost corect întelese de dezvoltator iar produsul obținut satisface asteptările clientului şi ale utilizatorilor finali.

2. Este posibil ca unele schimbari ale cerințelor pe parcursul dezvoltarii să nu fi fost comunicate dezvoltatorului.

Testarea de acceptare(3)

Tipuri de teste:

- Functionalitatea: produsul trebuie sa ofere toate funcțiile prevazute în contract şi să le execute corect.
- Acuratetea: precizia rezultatelor.
- Integritatea datelor: datele dintr-o baza de date nu sunt alterate prin operatii de interogare, actualizare, restaurare.
- Recuperererea datelor: se verifica daca datele pot fi recuperate dupa o cadere
  - Sunt toate datele recuperate?
  - Sunt consistente datele recuperate?

Testarea de acceptare (4)

Tipuri de teste(cont):

- Usurinta de utilizare: usurinta de invatare, de operare, calitatea interfetei utilizator, prezenta help-ului on-line, s.a.
- Usurinta de instalare si configurare
- Performanta: se verifica parametrii de performanta specificati.
- Fiabilitatea: rata caderilor in timpul testelor de acceptare
- Usurinta de intretinere si disponibilitatea sistemului: timpul mediu necesar reparării defectelor (MTTR) in urma caderilor din timpul testelor de acceptare
- Interoperabilitatea

si altele

Testele regresive

- Se numesc astfel testele executate dupa corectarea defectelor, pentru a se verifica daca in cursul corectarii nu au fost introduse alte defecte.

- Aceste teste sunt efectuate atat in timpul dezvoltarii, pe parcursul testelor, cat si in timpul perioadei de operare a sistemului.

- Pentru facilitarea lor este necesar sa se arhiveze toate cazurile de test executate in timpul dezvoltarii programului, ceea ce permite, în plus, verificarea automata a rezultatelor testelor regresive.

Lecturi suplimentare

1. Kshirasager Naik, Priyadarshi Tripathy, “Software Testing and Quality Assurance – Theory and Practice”, Publishe by Wiley John Wiley & Sons, New Jersey, 2008.
2. https://www.guru99.com/software-testing-introduction-importance.html
3. https://www.guru99.com/software-testing.html
4. https://www.tutorialspoint.com/software_testing/software_testing_types.htm