Tehnici de testare a aplicatiilor software.

1.Introducere

- Noile cerinte ale pietei au fortat producatorii de solutii IT sa fie nevoiti sa isi schimbe in intregime modul de constructie a unui produs, dar fara a face compromisuri de calitate.
- Testarea sistemelor informatice complexe presupune testarea pe componente si parti ale componentelor (functii sau clase), urmand un plan de test care trebuie sa contina toate cazurile de test.
- ***Testarea manuala*** :Procedurile de testare manuala, prin natura lor limitata, nu asigura gasirea tuturor defectelor si nu au nicio sansa sa simuleze conditii de utilizare simultana, intensive ale unei aplicatii.
- ***Testarea automata*** executa o secventa de actiuni fara a fi nevoie de interventia umana si pot simula utilizarea unei aplicatii ce presupune o simultaneitate ridicata. In mare parte, majoritatea produselor trebuie sa fie testate de mai multe ori, pe mai multe platforme software si hardware, ca si dupa schimbarile ulterioare sau lansarea unei noi versiuni. Un avantaj considerabil al acestei modalitati de testare este acela ca in urma testarilor repetate, costurile se reduc aproape la zero.

1.1. Ce este testarea?

- Testarea reprezinta procesul, ce contine toate activitatile din timpul ciclului de viata al unui produs, fiind ele statice cat si dinamice. Acest lucru presupune planificarea, pregatirea si evaluarea produselor software pentru :
  - a determina ca satisfac cerintele specificate
  - a determina ca indeplinesc scopul propus
  - a detecta defecte
- *O activitate statica* reprezinta analiza componentelor soft, care nu implica urmarirea efectiva a executiei acestora (spre exemplu scrierea cerintelor, scrierea codului, etc.)
- *O activitate dinamica* reprezinta analiza componentelor soft, care implica urmarirea efectiva a executiei lor (rularea softului).

1.2. Rolul testarii

- Rolul pe care testarea o are este ca:
  - sa se gaseasca defecte
  - sa se reduca impactul defectelor in mediul de lucru al clientului si sa se asigure ca acest lucru nu va afecta costurile si profitul
  - sa se creasca increderea in nivelul calitatii produsului
  - sa se asigure ca cerintele sunt indeplinite
  - sa se valideze produsul ca fiind potrivit pentru ceea ce a fost creat
  - sa se verifice ca standardele cerute si cele legale sunt indeplinite
  - sa nu se afecteze reputatia companiei

1.3. Principii de testare

- In urma testarii se gasesc defecte, dar nu exista siguranta ca nu mai exista si altele; testarea poate numai sa reduca numarul de defecte nesesizate.
- Testara completa este impossibila, de aceea cea mai buna solutie este sa se aplice o strategie si un management al riscului de calitate.
- **Regula Pareto** – de obicei numai 20% din module cotin 80% din defecte.
- Testarea din timp a produsului – testarea trebuie inclusa inca de la inceputul proiectului – fazele de planificare, proiectare si review.
- **Paradoxul Pesticid** – daca acelasi set de teste repetat de mai multe ori nu pune in evidenta noi defecte atunci trebuie sa se regandeasca testele si eventual sa fie modificate corespunzator.

1.4. Documente necesare testarii

- Cele mai importante documente necesare testarii unui produs sunt urmatoarele:
  - documente cu cerintele cerute
  - un document in care sunt descrise scopul, strategia, resursele si programul activitatilor de testare, care poarta numele de **TEST PLAN**.
  - cazurile de testare (teste), care poarta numele de **TEST CASE**.
  - un document care sa descrie rezultatele testarii si construieste o vedere de ansamblu asupra nivelului de calitate al produsului, care poarta numele de **TEST REPORT.**

1.5. Recomandari la testare

- se vor crea clase de test in acelasi pachet ca si codul testat
- pentru fiecare pachet Java (la testarea Java) al aplicatiei se va defini o clasa TestSuite ce contine toate testele pentru validarea codului din pachet
- se va alterna scrierea codului sursa cu operatii de testare pentru a se depista rapid eventuale bug-uri
- se vor scrie teste pentru zonele cu risc maxim de erori
- se vor scrie teste unitare inaintea scrierii codului si se va scrie cod nou numai cand un test esueaza

2.Strategii si tipuri de testare

2.1. Unit testing

Termenul de testare unitara se refera la testarea individuala a unor unitati separate dintr-un sistem software. In timpul proiectarii si codificarii se comit erori care sunt grupate in urmatoarele categorii:

- erori legate de alegerea si descrierea algoritmului: algoritm incorect, sau corect dar inadecvat problemei;
- erori in definirea si utilizarea datelor ce provin din variabile neinitializate, neverificarea datelor de intrare, utilizarea unor cuvinte cheie ca variabile, variabile ilegale ;
- erori produse in tehnica de programare cum sunt variabile si structuri de date globale, acces necontrolat la zone de memorie partajate, interfete program - subprogram nerespectate;
- erori in contextul executiei, datorate memoriei dinamice insuficiente sau nealocata, periferice neoperationale, comunicare defectuoasa cu sistemul de operare.

2.2. Testul de integrare

Are ca obiectiv testarea pe diverse niveluri de integrare a modulelor. Se pune accentul pe functionarea corecta a ansamblului, pe compatibilitatea dintre componente, de asemenea se pune accent pe depistarea erorilor de interfata intre module,eliminarea conflictelor privind accesul la resursele de calcul.

Este indicat ca testele sa fie desfasurate intr-un mediu cat mai apropiat de cel in care sistemul va functiona ulterior.

Sunt practicate trei tipuri de modalitati ale testarii:

- **Testarea de sus in jos, top-down** - folosita numai in cazul unei conceperi descendente a sistemului.;

-se porneste cu modulul radacina si cu unul sau mai multe niveluri de ordin imediat inferior; dupa testarea acestui schelet care probeaza toate posibilitatile legaturilor (interfetelor) se adauga un alt nivel inferior;

-cand s-au adaugat modulele ultimului nivel testarea este terminata.

➢ **Testarea de jos in sus, bottom-up** - o metoda clasica de testare si consta in testarea individuala a modulelor urmata de testarea ansamblului de module ca un tot unitar.

-presupune verificarea sistemului din punct de vedere al specificatiilor sale, dar si al performantelor: timp de raspuns, capacitate de lucru etc.

-un dezavantaj al metodei este dificultatea in stabilirea datelor de test pentru sistemul final.

Testarea mixta - presupune aplicarea simultana a celor doua metode precedente, ramanand predominanta testarea descendenta. Prin urmare se incepe elaborarea proiectului intr-o maniera descendenta, dar simultan se realizeaza module din nivelul de baza al ierarhiei. In acest caz testarea descendenta se desfasoara paralel cu cea ascendenta. Acest procedeu de concepere si testare s-a dovedit eficient in elaborarea multor produse program.

Dupa cum se observa, nici una din metode nu trebuie impusa in mod rigid; fiecare sufera mici variatii in cazul unor proiecte particulare. Sunt si situatii cand ordinea de abordare contine si zigzag.

2.3. Testul de validitate

Validitatea este definita in diferite moduri dar o definitie simpla spune ca validarea este terminata cand aplicatia software functioneaza intr-o maniera rezonabila acceptata de client. Acceptarile rezonabile sunt definite in documentul ce cuprinde specificatia cerintelor si care descrie toate atributele cerute de client.

Un element important al procesului de validare este revederea configuratiei - configuration review. Se verifica daca toate elementele necesare pentru configurare au fost dezvoltate si functioneaza la parametri stabiliti. Este imposibil de imaginat cum clientul va folosi in realitate produsul realizat de o companie de software, chiar daca acesta cuprinde un manual de utilizare.

2.4. Testul de acceptare

Tipul de testare formala, avand la baza cerintele utilizatorului, cerintele si procesele impuse de client, astfel incat, sa se determine daca un sistem sau componenta satisface aceste criterii, pentru a fi acceptat sistemul sau componenta de clientul .

Se efectueaza cu scopul de a valida functional produsul din perspectiva utilizatorului final.

Se apreciaza ca acest tip de testare, care implica si participarea viitorilor beneficiari, este pentru dezvoltator o importanta sursa de informatii privind contextul in care va fi utilizat sistemul.

Un test de acceptare este un test de genul:

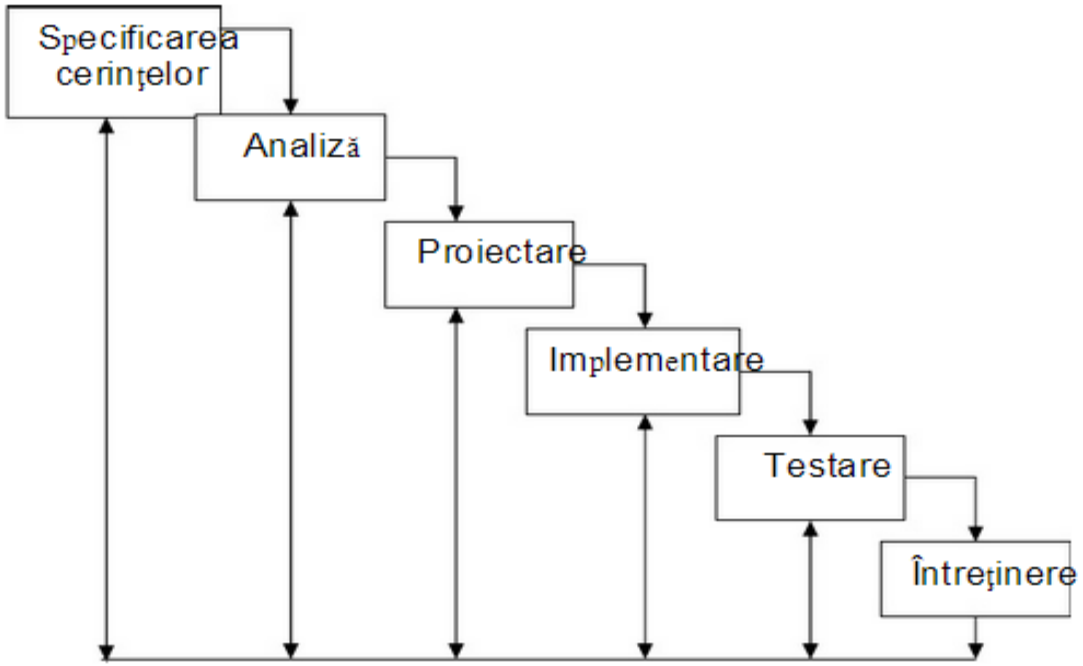
1. Verificam ca aplicatia se instaleaza corect folosind valorile implicite (default)
2. Verificam ca aplicatia porneste folosind un cod valid
3. Verificam ca aplicatia se dezinstaleaza corect (nu raman fisiere in directorul de instalare, este sters directorul de instalare)

Pe langa tipurile de testare prezentate mai sus mai sunt si altele cum ar fi:

- Testarea Ad-Hoc
- Testare regresiva
- Testarea negativa
- Testarea securitatii
- Testarea performantei
- Testarea de integrare

3.Tehnici de testare

Procesul de dezvoltare al aplicatiilor software este alcatuit din urmatoarele etape: specificarea cerintelor, analiza, proiectare, implementare, testare si intretinere. In figura de mai jos se poate observa graficul modelul clasic de dezvoltare software.



3.1 Modelul de testare clasic

Testarea specificatiilor se realizeaza prin metode de analiza statica: inspectii, parcurgeri si analize tehnice.

Se testeaza aplicatia, in faza de proiectare, prin intelegerea domeniului si a scopului aplicatiei;

Testarea in etapa de implementare are rolul de a evidentia diferentele dintre comportamentul produsului din specificatii si cel dat la nivelul utilizatorilor. Un rol important il au verificarea si validarea, prin care se determina daca cerintele unui sistem sau componenta sunt complete si corecte, daca rezultatul fiecarei faze de dezvoltare indeplineste cerintele, sau conditiile impuse in faza anterioara si daca sistemul sau componenta finala este in concordanta cu cerintele specificate.

VERIFICAREA este multimea activitatiilor care asigura o aplicatie software, implementeaza corect o anumita functie, iar VALIDAREA este multimea activitatiilor care asigura ca o aplicatie software realizata este in concordanta cu cerintele beneficiarului.

Testarea dinamica presupune examinarea aplicatiilor software, in scopul generarii datelor de test cu care acestea vor fi executate si rularea aplicatiilor cu seturile de date de test obtinute. Se observa ca spre deosebire de testarea statica, testarea dinamica presupune executia aplicatiei care se testeaza.

3.2 Black Box

Testarea functionala (black-box testing) este o strategie de testare care necesita cunoasterea comportamentului extern al programului pe baza specificatiilor, nu necesita cunoasterea structurii interne a programului sau cunostinte despre modul in care este implementat programul sau modulul.

Testarea cu **intrari aleatoare** - generarea de date de test in mod aleator, in cadrul domeniului acestora. Aceasta tehnica de testare a fost dezvoltata, astfel incat datele de test sunt alese aleatoriu din domeniu si sunt filtrate astfel incat sa indeplineasca anumite conditii, ca de exemplu producerea unui rezultat dintr-o clasa de iesiri posibile.

Partitionarea pe **clase echivalente** - consta in impartirea domeniului datelor de intrare in subdomenii, astfel incat comportamentul programului sa fie acelasi, cel putin teoretic, pentru orice data de intrare din subdomeniul corespunzator.

3.3 White box sau testarea structurata

Testarea structurata (white box testing) este o strategie de testare care necesita accesul la codul sursa si la structura programului si pune accentul pe acoperirea prin teste a cailor, ramificatiilor si fluxurilor programului.

Strategiile de testare bazate pe cai utilizeaza **fluxul de control al programului**. Acestea reprezinta o familie de tehnici de testare bazate pe selectarea cu atentie a unei multimi de cai din program. Cu ajutorul acestor tehnici se detecteaza erorile care cauzeaza executia unei alte cai a programului decat aceea care trebuia sa se execute.

Criteriul pentru acoperirea instructiunilor este ca fiecare instructiune sa fie executata cel putin o data in cadrul unor teste. Daca se realizeaza acest obiectiv, atunci declaratiile sunt acoperite in proportie de 100%. Acest lucru este echivalent cu o acoperire a nodurilor fluxului de control asociat programului in proportii de 100%.

Acoperirea ramificatiilor are drept criteriu ca fiecare ramura a unei decizii sa fie executata cel putin odata. Se ruleaza un numar suficient de teste pentru a se executa toate ramificatiile din program cel putin o data.

3.4 Grey Box

Este tehnica de testare care implica si accesul la parti interne ale sistemului sau componentei supusa testar

Testul (folosind tehnica grey box) va fi:

- generam niste coduri valide si niste coduri invalide;
- introducem in dialogul de start cod valid => aplicatia porneste si verificam baza de date pentru a vedea ca acest cod nu mai poate fi folosit inca o data (s-a setat valoarea TRUE in campul: “FOLOSIT”).
- introducem in dialogul de start cod invalid => aplicatia nu porneste si verificam baza de date pentru a vedea ca pentru nici o intrare din campul “FOLOSIT” nu a fost updatata.

3.5 Testarea software orientate pe obiecte

Programarea orientata pe obiecte presupune definirea de clase si obiecte, construirea de ierarhii, precum si referirea obiectelor create.

Testarea claselor evidentiaza gradul de mostenire si de acoperire probabila a tipologiilor de clase de probleme prin constructori/destructori definiti.

Testarea software orientata pe obiecte are pe langa obiectivul general al stabilirii masurii in care produsul software realizeaza sarcinile prezentate in specificatii, obiective specifice legate de:

- testarea metodelor membre ale fiecarei clase;
- testarea gradului de incapsularesi a efectelor acestuia;
- testarea efectelor induse de nivelurile de mostenire/derivare;
- testarea efectelor induse de polimorfismul metodelor membre;
- testarea interactiunilor dintre clase.

4. Testarea Unitara

- Testarea unitara s-a impus in ultima perioada in dezvoltarea proiectelor scrise in diferite limbaje de programare
- A dus la cresterea vitezei de programare si la miscorarea drastica a numarului de bug-uri
- Reprezinta procedeul de testare a unitatilor elementare din program. In abordarea obiectula, acestea reprezinta metodele unei clase.
- Ideea este **izolarea** fiecarui modul de functionalitate a unui program si asigurarea corectitudinii **individuale**.

4.1 JUnit

- Cel mai folosit utilitar pentru testarea unitara a claselor Java
- La realizarea unui proiect software complex codul sursa al aplicatiei este foarte important.
- Calitatea codului sursa al unei aplicatii software este data de :

- corectitudinea sintaxei codului sursa. Pentru asigurarea acestei corectitudini este responsabil compilatorul de Java.

- corectitudinea comportamentului, adica daca fiecare metoda realizeaza ceea ce s-a specificat in documentatia proiectului.

- Pentru asigurarea unui comportament corespunzator se vor utiliza teste unitare realizate cu ajutorul utilitarului JUnit.

4.2 Avantajele utilizarii JUnit

- Este imbunatatita viteza de scriere a codului, si in acelasi timp, creste si calitatea acestuia, deoarece prin scrierea testelor unitare se micsoreaza timpul de depanare, permitand depistarea imediata a eventualelor erori inserate in codul modificat
- Clasele de test sunt usor de scris si de modificat, pe masura ce codul sursa se mareste, putand fi compilate impreuna cu codul sursa al proiectului. Compilatorul testeaza sintaxa codului sursa, in timp ce clasele de test valideaza integritatea codului
- Clasele de test JUnit pot fi rulate automat, rezultatele fiind vizibile imediat, putandu-se crea ierarhii de suite de test, care pot fi testate impreuna sau separat, in functie de cerintele proiectului
- Clasele de test maresc increderea programatorului in codul sursa scris si ii permit sa urmareasca mai usor cerintele de implementare ale proiectului, putand constitui si o parte a documentatiei finale, care trebuie transmisa clientului.

4.3 Unit Test Case

- Un caz de testare este o parte a codului care asigura ca alta parte a codului (metoda) functioneaza conform asteptarilor.
- Pentru a obtine rezultatele dorite, repede, este necesar un framework.
- Un caz de testare formal este caracterizat de o intrare cunoscuta si de o iesire asteptata
- Datele de intrare cunoscute, trebuie sa testeze o preconditie si iesirea asteptata trebuie sa testeze o postconditie.
- Ar trebui sa fie cel putin doua cazuri de test pentru fiecare cerinta: un **test pozitiv** si un **test negativ**. Daca o cerinta are si subcerinte, fiecare subcerinta ar trebui sa aiba cel putin doua cazuri de test, pozitiv si negativ.

4.4 Terminologia JUnit

- Test runner – software care ruleza teste si afiseaza rezultate
- Test suite – colectie de cazuri de test
- Test case – testeaza metodele pe seturi particulare de intrari
- Unit test – testarea celui mai mic element a codului care poate fi testat (o singura clasa)
- Test fixture – mediul in care este rulat testul
- Integration test – test care arata cat de bine lucreaza clasele impreuna

Actualmente se foloseste **JUnit5** care e compus din:

JUnit 5.x = *JUnit Platform* + *JUnit Jupiter* + *JUnit Vintage*

http://junit.org/junit5/docs/current/user-guide/

https://www.tutorialspoint.com/junit/junit_test_framework.htm

4.4 Utilizare JUnit (pasi de baza)

Pasul 1: Se creaza un proiect in Eclipse

Pasul 2: Se creaza o clasa Java

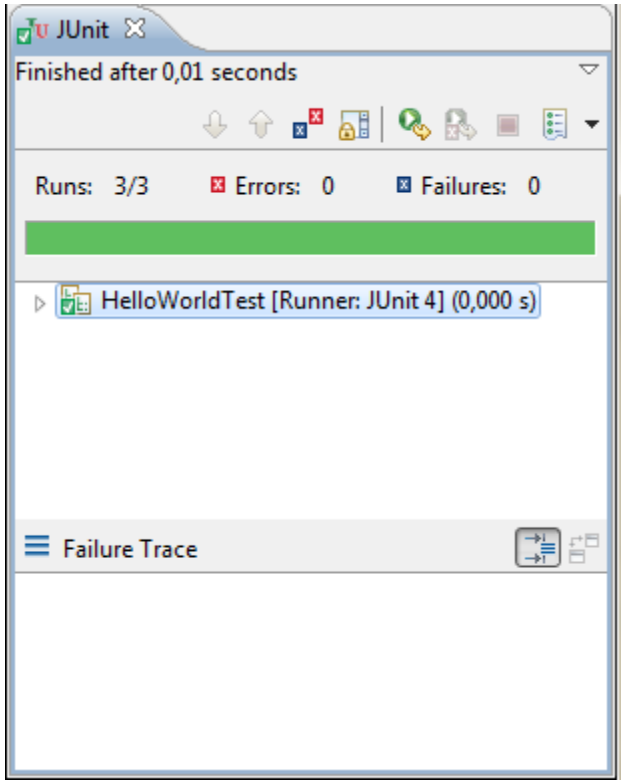
Pasul 3: Se adauaga arhiva *junit.jar* in biblioteca proiectului creat (se face automat interactiv la crearea Test Case)

Pasul 4: Se creaza o clasa de test (JUnit Test Case), in care sunt definite functii pentru crearea tuturor functiilor din clasa care se doreste a fi testate, se alege ca si optiune adecvata din New

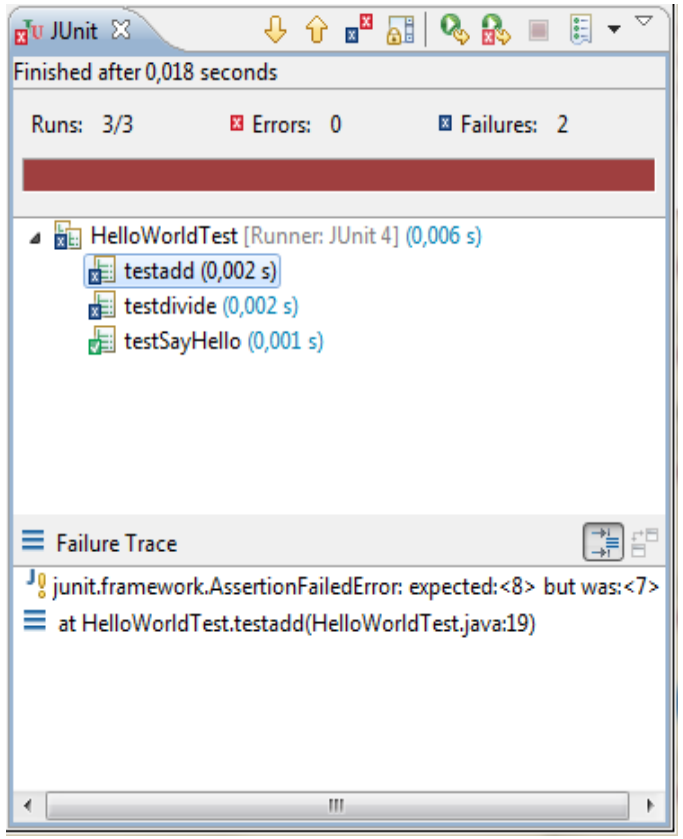
Pasul 5**:** Se compileaza clasa de test si se lanseaza in executie

Utilizare JUnit

- Rezultatele obtinute:



Rezultat fara erori



Rezultat cu esecuri

Rezultat cu erori