QUALITY ASSURANCE

CURS 2

AGENDA

- **Software Development Life Cycle**
- **Metode si tipuri de teste**
- **Test case design**
- **Test documents**

SDLC

SOFWARE DEVELOPMENT LYFE CYCLE

SDLC - SOFTWARE DEVELOPMENT LIFE CYCLE

- **Procesul de dezvoltare software pornind de la nevoie de business, analiza, design, implementare si mentenanta**
- **Faze prin care trece**

- **Identificarea unei probleme**
- **Analiza cerintelor**
- **Dezvoltarea**
- **Testarea**
- **Exploatarea**
- **The End**

TEHNICI SI TIPURI DE VERIFICARE

- **Verificari sunt aplicate de-a lungul procesului (e.g., software, hardware, documentation) pentru a asigura aderarea la standarde, proceduri, guidelines, etc**

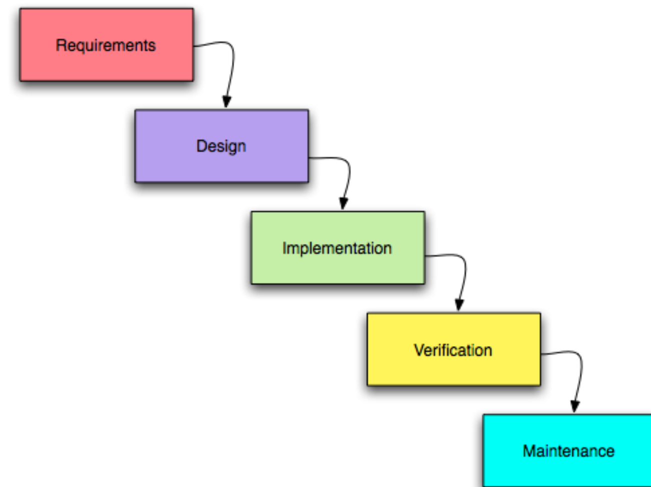
- **Review-uri cat mai devreme si cat mai dese**

- **Metode de verificare**
  - Requirements Review/Walk-through
  - Code Inspection/ Walk-through
  - Requirements tracing (in tot ciclul de viata; requirements versus design; requirements versus tests, etc)
  - Traceability matrix
  - Physical Audit (correct versions of hardware & software components)

WATERFALL

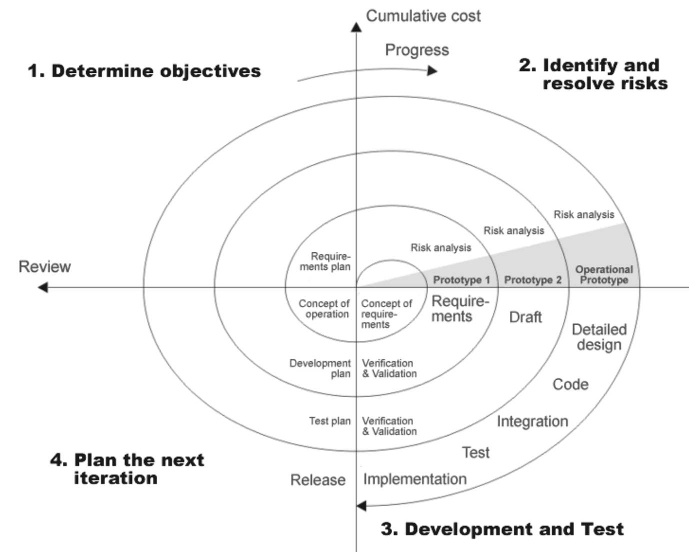
- **Model secvential**

- **Trecere treptata prin toate fazele:**
  - Requirements
  - Design
  - Implementation
  - Verification
  - Maintenance



SPIRALA

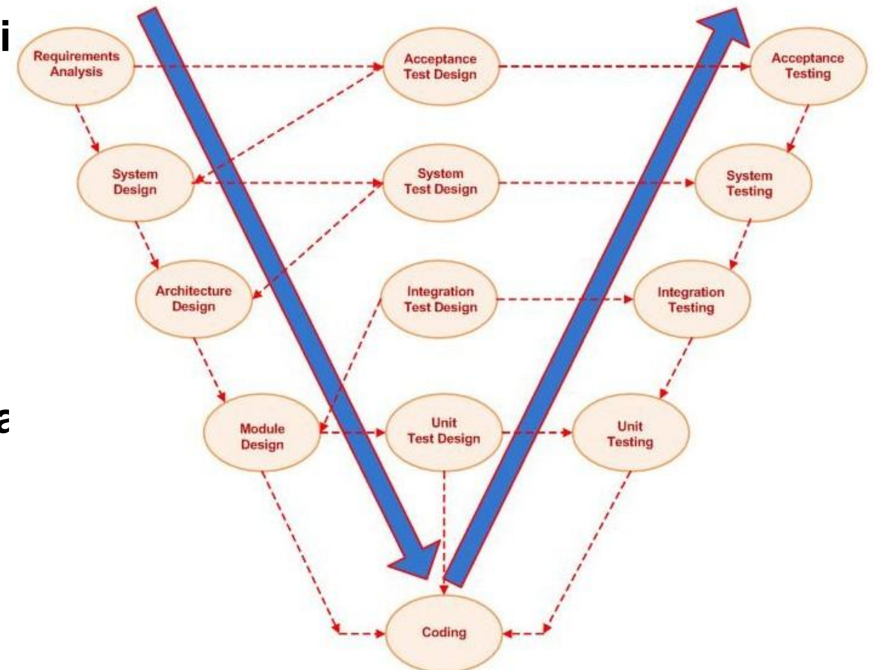
- **Combina elemente de design si prototip in faze succesive**
- **Incercare de a combina avantaje designului de tip top-down si bottom-up**
  - Pasii din spirala pot fi generalizati dupa cum urmeaza:
  - Ceritele sunt adunate (interviuri)
  - Design preliminar
  - Prototip bazat pe designul prelimar (o aproximare a produsului final)
  - Al doilea prototip:
    - Evaluarea primului prototip din perspectiva slabiciunilor, avantajelor, riscurilor
    - Definirea cerintelor pentru al doilea prototip
    - Planificare si design
    - Dezvoltare si testare



- **La alegerea clientului intreg proiectul poate fi oprit daca riscul este prea mare:**
  - Depasiri de costuri
  - Costuri operationale
  - Satisfactie

V MODEL

- **Integreaza activitati de dezvoltare si testare**
- **Activitati dezvoltare pe partea stanga si testare pe partea dreapta**
- **Testing nu este vazut ca o faza la sfarsitul dezvoltarii**
- **Pentru fiecare faza de dezvoltare, o faza de testare trebuie sa aiba loc**
- **Testarea bazata pe documente**
- **Erorile gasite mai devreme sunt mai ieftin de reparat**



UNIT TEST

- **"A fault discovered and corrected in the unit testing phase is more than a hundred times cheaper than if it is done after delivery to the customer."** ***Barry Bohem***
- **Testarea componentelelor individuale (clase, proceduri, etc)**
- **Executat in general de programatorul care a scris codul**
- **Aspecte functionale si nefunctionale (memory leaks)**
- **Defectele sunt reparate pe loc fara a nota incidentele**

- **Entry Criteria**
  - Requirements finalizate in proportie de 80%
  - Technical Design finalizat
  - Mediul de dezvoltare stabilit
  - Dezvoltare modului este terminata
  - Toate test caseurile sunt documentate
- **Exit Criteria**
  - Nu sunt cunoscute defecte majore
  - Aprobarea Pmului
  - Toate testele au trecut cu succes

SYSTEM TEST

- **Testeaza ca sistemul face ceea ce trebuie (functional) si cat de bine face (non-functional)**

- **Doua tipuri**
  - Functional
  - Non-functional : Security, Reliability, Usability, Recovery, Interoperability etc…

- **Entry Criteria**
  - Unit Testing terminat si aprobal
  - Plan de gestiune a defectelor aprobat
  - Mediul de test stabilit (apropiat de mediul de productie)

- **Exit Criteria**
  - Toate cerintele functionale indeplinite
  - Nu exista defecte majore care pot bloca Integration Testing

INTEGRATION TEST

- **Testeaza interfata intre componente si inteactiunea cu diferite parti ale sistemului**
- **Concentrat doar pe integrarea dintre module (comunicarea dintre modulul A si B si nu functionalitatea modulelor)**
- **Mai multe nivele de integrare**
  - Intre componentele sistemului
  - Interactiunea cu alte sisteme
- **Ideal echipa de test ar trebui sa inteleaga arhitectura sistemului**

- **Entry Criteria**
  - Unit testing terminat
  - Defectele identificate si documentate
  - Mediul de test stabilit
- **Exit Criteria**
  - Toate sistemele implicate au trecut testele de integrare
  - Defecte documentate si prezentate stakeholderilor
  - Teste de stress, performanta, load au trecut cu succes

ACCEPTANCE TEST

- **Pentru a determina daca indeplineste criteriile de acceptare… si daca clientul accepta sistemul**
- **Sistemul este testat de catre audienta pentru care a fost construit**
- **Scopul de a stabili daca sistemul/schimbarile rezolva problema initiala**
- **Definirea criteriilor de acceptare**
  - Functionality requirements
  - Performance requirements
  - Interface quality requirements
  - Overall software quality requirements
- **Definirea unui plan de acceptare**
  - Responsabilitatile utilizatorilor
  - Descrierea acceptarii (si durata)
  - Executarea planului de test

- **Entry Criteria**
  - Integration testing aprobat
  - UAT test scripts pregatite
  - Mediul de test stabilit
  - Cerinte de securitate stabilite
- **Exit Criteria**
  - UAT complet si aprobat
  - Cererile de modicare sunt gestionate
  - Sponsorii sunt de acord ca defectele cunoscute nu au impact in productie

METODE SI TIPURI DE TESTE

MISC

- **Testarea este procesul de verificare prin executie a functionalitatii si corectitudinii programului.**

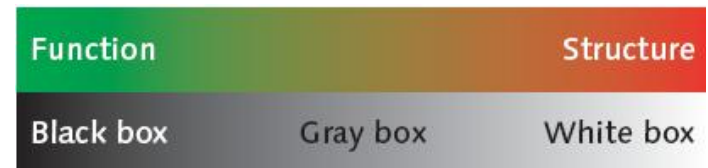
- **Scopul**
  - Gasirea defectelor
  - Verifica ca toate functionalitatile sunt implementate
  - Utilizatorul poate folosi aplicatia in toate procesele de business

- **Eroare**
  - O diferenta intre program si specificatii este eroare… daca specificatiile exista si sunt corecte
  - Categorii: User Interface errors, Functionality errors, Performance errors, Output errors

- **Daca nu avem suficient timp pentru teste analizam riscurile:**
  - Care functionalitati sunt importante
  - Care functionalitati sunt cele mai vizile pentru utilizatori
  - Care parti ale codului sunt cele mai complexe
  - Care tipuri de teste ar putea acoperi cel mai usor toate functionalitatile

METODE

- **Functionale sau structurale**
- **Traditional:**
  - Functionale = black box
  - Structurale = white box
- **Graybox = Intre cele doua un numar mare de tipuri de teste care nu intra in nici o categorie**



ALTE TIPURI DE TESTE

- **Sanity Testing**
  - Test initial pentru a determina daca noua versiune este suficient de stabila pentru o runda majora de teste
- **Installation/Uninstallation Testing:**
  - Testarea procesului intreg, partial, upgrade de instalare/dezinstalare
- **Usability Testing:**
  - Testarea factorului de 'user-friendliness'
  - Evident subiectiv
  - Pot fi folosite interviuri, sondaje, inregistrari user sessions
  - A/B testing
- **Stress Testing:**
  - Cauta erori produse de resurse reduse sau competitia pentru resurse
- **Load Testing:**
  - Testarea la cea mai mare rata de procesare a tranzactiilor pentru a verifica rata de realocare a resurselor, database locks etc…
- **Volume testing**
  - Supune aplicatia la volume din ce in ce mai mari de date

ALTE TIPURI DE TESTE

- **Security Testing**
  - Cat de bine se protejeaza sistemul impotriva accesului neautorizat intern si extern
- **Compatibility Testing**
  - Cat de bine se comporta sistemul intr-o anumita configuratie hardware/software/network/etc
- **Ad-hoc Testing**
  - Testarea aplicatiei intr-o maniera “random”
- **Regression Testing**
  - Testarea aplicatiei pentru a verifica daca functionalitati existente sunt corecte dupa adaugarea de noi functionalitati in aplicatie
- **Exploratory testing**
  - O tehnica de testare neformala in care testerii creeaza test cases in timpul executiei testelor
  - Informatia colectata este folosita pentru a creea test cases mai complete si mai bune

REGRESSION TESTING

- **Testarea repetata a unui program deja testa, dupa modificari, pentru a descoperi defecte introduse ca urmare a unor schimbari**
- **Statistic una din sase incercari de corectie are la randul sau defecte**
- **Realizat la schimbari de software, mediu, etc**
- **Se aplica la nivel functional si nefunctional**
- **Aria de cuprindere este legata de riscul implicat de schimbari, marimea sistemului, marimea schimbarilor**
- **Regression testing este un candidat puternic pentru testarea automata**

REGRESSION TESTS (2)

- **Ce ar trebui sa includem:**
  - Refolosim test cases de la unit testing, integration testing si system testing
  - Ne ghidam dupa principiul 80/20 (cele mai utilizate pagini/dialoguri, meniuri sunt primii candidati)

- **Consideratii de risc: anumite defecte nu sunt frecvente dar cand apar pot avea impact mare**

- **Teste aditionale pot fi adaugate in zonele de aplicatie care sunt dificil de intretinut sau au istoric o rata mare a defectelor**

- **Testarea ar trebui sa inceapa de la nivel de unit, unit test-urile pot fi adaptate si rulate dupa schimbarile de la modulul afectat**

- **Ar trebui sa continue cu testele de integrare, sistem, user acceptance**

TEST CASE DESIGN

METODOLOGII

- **Ce subset din multimea posibila a test caseurilor are cea mai mare probabilitate de a detecta cele mai multe erori?**

- **Black Box**
  - Partitionarea in clase de echivalenta
  - Boundary-value analysis

- **White Box**
  - Statement coverage
  - Decision coverage
  - Condition coverage

- **Combinatii ale metodelor de mai sus pentru un test plan riguros**

- **Procedura recomandata este sa faci test case-uri folosind metode black-box si apoi sa adaugi test case-uri suplimentare white-box acolo unde este cazul**

WHITE BOX LOGIC COVERAGE TESTING

- **Scopul final este executia fiecarei cai din program (nerealistic)**
- **A=2, B=0, and X=3 executa fiecare instructiune**
  - Exista o cale in program prin care x ramane neschimbat. Daca asta ar fi o eroare atunci ar trece neobservata.
- **Criteriul "Statement coverage" este atat de slab incat e aproape inutil**
- **Un criteriu mai puternic pentru logic-coverage criterion este decision coverage sau branch coverage.**
- **Decission coverage = fiecare decizie din program ia valorile true/false cel putin o data. Adica, fiecare ramura este traversata cel putin o data.**

```
public void foo(int a, int b, int x) {
    if (a>1 && b==0) {
        x=x/a;
    }
    if (a==2 || x>1) {
        x=x+1;
    }
}
```

WHITE BOX LOGIC COVERAGE TESTING

- **Decision coverage de obicei satisface statement coverage.**
  - Deoarece fiecare instructiune este o sub-ramura dintr-o instructiune de control (branch statement) sau intrarea in program, fiecare instructiune e executata daca fiecare ramura e executata.
- **Criteriu mai complet este condition coverage. Suficiente cazuri de test astfel incat fiecare conditie dintr-o decizie ia toate valorile posibile cel putin o data**

```
public void foo(int a, int b, int x) {
    if (a>1 && b==0) {
        x=x/a;
    }
    if (a==2 || x>1) {
        x=x+1;
    }
}
```

CLASE DE ECHIVALENTA

- **Un test case bun are probabilitate mare sa gaseasca erori**
- **Testarea tuturor datelor de intrare este (in general) imposibila**
- **Rezulta, atunci cand testezi, esti limitat la a incerca un subset limitat din multimea valorilor de intrare**

- **Vrem sa selectam date de intrare cu cea mai mare probabilitate de a gasi erori**
  - Reduce cu macar unul numarul de alte test case-uri care trebuie create
  - Acopera un set de alte posibile test case-uri. Ne spune ceva despre prezenta/absenta erorilor dincolo de setul specific de valori

- **Implicatii**
  - Fiecare test case ar trebui sa invoce cat mai multe combinatii de date de intrare pentru a reduce numarul de test case-uri
  - Partitionare domeniului intr-un numar finit astfel incat sa poti spune cu ceva certitudine (dar fara a fi absolut sigur) ca un test pentru o valoare din acea clasa e reprezentativ pentru intreaga clasa

CLASE DE ECHIVALENTA

- **Test-case design folosind clase de echivalenta:**
  - Identificarea claselor de echivalenta
  - Definirea test caseurilor

- **Identificarea claselor de echivalenta este in mare un proces euristic**
  - “valoarea poate fi intre 1 si 999” o clasa valida (1 < x < 999) si doua invalide (x < 1 x > 999).
  - Programul accepta si proceseaza diferit o selectie de valori (“tip vehicul BUS, TRUCK, TAXICAB, PASSENGER, or MOTORCYCLE”), alege o clasa de echivalenta pentru fiecare valoare din lista si una invalida (“TRAILER”).
  - Conditia este “primul caracter sa fie litera” avem o clasa valida si una invalida (nu este litera).

BOUNDARY VALUE

- **Boundary values sunt valori exact la, deasupra, dedesubt marginea claselor de echivalenta pentru datele de intrare/iesire**

- **Din exerienta test case-urile care exploreaza boundary conditions sunt cele mai “profitabile”.**

- **Boundary-value difera de partitionarea in clase de echivalenta:**
  - Unul sau mai multe elemente sunt selectate astfel incat marginea clasei este supusa testului
  - Se exploreaza si datele de iesire

- **Exemplu**
  - Domeniul valid −1.0–+1.0 atunci test case-uri pentru −1.0, 1.0, −1.001, and 1.001.

RECOMANDARE

- **Studiati exemplul MTEST din The Art of Software Testing**

STRATEGIE

- **Metodologiile de design ar trebui combinate (puncte slabe in fiecare dintre ele)**

- **Foloseste boundary-value atat pentru date de intrare cat si pentru date de iesire**

- **Identifica clase de echivalenta valide si invalide**

- **Examineaza logica programului si foloseste decision-coverage, condition-coverage, decision/ condition-coverage pentru a satisface criteriul de acoperire**

INVITAT SAPTAMANA VIITOARE

ELVIS CIOCOIU, MANAGER, RED POINT

HAVE FUN!

- **To err is human. To really foul things requires a computer.**
- **Myth #1: The computer only does what you tell it.**
- **The Definition of an Upgrade: Take old bugs out, put new ones in.**
- **Computers make very fast, very accurate mistakes.**
- **“It did what? Well, it's not supposed to do that.”**
- **“My software never has bugs. It just develops random features.”**
- **If debugging is the process of removing bugs, then programming must be the process of putting them in.**