NOȚIUNI TEORETICE DESPRE INGINERIA PROGRAMĂRII

Necesitatea:
- complexitatea software-ului din zilele noaste este foarte mare (milioane de linii de cod);
- s-a constatat prin analiza diverselor proiecte și stadiilor lor de finalizare că doar 2% din sistemele software au funcționat de la predare, 19% au fost folosite dar abandonate și 47% au fost plătite dar niciodată predate;
- constructorii de aplicații software trebuie să procedeze precum inginerii în dezvoltarea programelor (plan, construire machetă, studierea materialelor folosite, rapoarte privind progresul operațiilor).

Definiție (IEEE Standard Glossary of Software Engineering Technology):
Ingineria software, ingineria sistemelor de programe (ingineria programării), reprezintă abordarea sistematică a dezvoltării, funcţionării, întreţinerii, şi retragerii din funcţiune a programelor.

Problema fundamentală a ingineriei software este ***satisfacerea cerinţelor utilizatorului***. Aceasta trebuie realizată nu punctual, nu imediat, ci într-un mod flexibil şi pe termen lung.

Atributele cheie ale unui produs software se referă la:

Mentenabilitate, posibilitatea de a putea fi *întreţinut*: Un produs cu un ciclu de viaţă lung este supus deseori modificărilor, de aceea el trebuie foarte bine documentat;
Fiabilitate: produsul trebuie să se comporte după cerinţele utilizatorului şi să nu „cadă" mai mult decât e prevăzut în specificaţiile sale;
Eficienţă: produsul nu trebuie să folosească în pierdere resursele sistemului ca memoria sau timpul de procesare;
Interfaţa potrivită pentru utilizator: interfaţa trebuie să ţină seama de capacitatea şi cunoştinţele utilizatorului.

Etapele de dezvoltare a sistemelor de programe

1. Ciclul de viață

Există patru faze fundamentale ale metodologiilor ingineriei software:

1. Analiza (ce? dorim să construim)
2. Proiectarea (cum?)
3. Implementarea (construirea propriu-zisă)
4. Testarea

Una dintre ideile dominante în abordarea dezvoltării software-ului este **dezvoltarea pe baza ciclului de viaţă sau modelul « cascadă ».**
În acest context se împarte dezvoltarea unui produs informatic în paşi independenţi, aflaţi într-o anumită succesiune.
Paşii succesivi sunt:

stabilirea cerinţelor

□

specificarea

□

proiectarea

□

implementarea

□

testarea

□

operarea şi întreţinerea

Specialiştii au idei diferite despre ce trebuie să conţină exact fiecare pas în parte, dar principiile modelului ciclului de viaţa sunt :

1. Există o serie succesivă de paşi ;
2. Fiecare pas este bine definit ;
3. Fiecare pas creează un produs definit (adesea doar o foaie de hârtie) ;
4. Corectitudinea fiecărui pas trebuie verificată cu grijă.

1.1 **Faza de analiză**

– definește cerințele sistemului;
– se definește problema pe care clientul dorește să o rezolve (așteptările clientului sau criteriile pe care trebuie să le îndeplinească produsul);

Rezultat final → un document numit specificarea cerințelor care descrie obiecte (identificarea pieselor, părțile componente, constante, nume și relațiilr dintre acestea), acțiuni (ce trebuie să le îndeplinească sistemul, funcțiile sau procedurile), stări (exemple: starea inițială, finală și de eroare), scenarii tipice (secvență de pași pentru îndeplinirea unui scop) și scenarii atipice (numai în cazuri speciale).

1.2 **Faza de proiectare**

Pe baza cerițelor din faza de analiză se stabilește arhitectura sistemului: componente, interfețe și modul de comportare (detalii despre limbaj de programare, medii de dezvoltare, memoria, platforma, algoritmi, etc.).

1.3 **Faza de implementare**

Sistem construit de la zero pe baza documentelor din fazele anterioare. O problemă importantă este îndepărtarea erorilor critice ce pot împiedice sistemul să satisfacă în mod complet scenariile de utilizare.

1.4 **Testarea**

Important pentru verificarea calității produsului final (testare internă pentru fiecare componentă sau funcție – white-box, testarea unităților ca un întreg – black box, testarea aplicației ca întreg, testarea interfeței grafice, testare la stres). În multe metodologii ale ingineriei software, faza de testare este o fază separată, realizată de o echipă *diferită* după ce implementarea s-a terminat. Motivul este faptul că un programator nu-şi poate descoperi foarte uşor propriile greşeli.

Metode de specificare a cerintelor utilizator

- **Limbajul natural – este ambiguu**
- **Limbaj natural structurat**
- **Tabele, formule**
- **Diagrame de flux de date**
- **Elemente de modelare UML**: cazuri de utilizare, diagrame de cazuri de utilizare pentru delimitarea sistemului in mediul sau de operare, diagrame de secventa pentru descrierea scenariilor de utilizare a sistemului, diagrame de stari

UML - limbaj unificat de modelare Unified Modeling Language

Limbajul unificat de modelare, numit prescurtat UML se doreşte a fi un instrument care să ofere o metodologie unificată de modelare, analiză şi proiectare a sistemelor informatice. UML a fost selectat ca standard al limbajelor de modelare orientate obiect (OOML- Object Oriented Modeling Language).

Componentele de bază ale UML sunt elementele model.
Un ***model*** reprezintă o colecţie de *obiecte* (clase, pachete, actori, cazuri de utilizare, componente şi noduri), *relaţii* (de asociere, dependenţă, generalizări) şi *diagrame*.
Modelele sunt de două tipuri:
 modele structurale *(statice)* care accentuează structura obiectelor din sistem, incluzând şi clasele lor, interfeţele, atributele şi relaţiile şi

 modele de comportament *(dinamice)*, care accentuează comportamentul obiectelor din sistem, incluzând şi metodele, interacţiunea, colaborările şi starea lor istorică.

Diagrame şi concepte UML
UML defineşte patru tipuri de diagrame :
1. Diagrama de clase (Class Diagram);
    1.1. Diagrama de obiecte (Object Diagram);
2. Diagrama cazurilor de utilizare (Use Case Diagram);

3. Diagrame de comportament (Behavior Diagrams);
    3.3.1. Diagrama de stare, sau grafice de stări (Statechart Diagram);
    3.3.2. Diagrama de activitate (Activity Diagram);
    3.3.3. Diagrama de interacţiuni (Interaction Diagrams);
    3.3.4. Diagrama secvenţială (Sequence Diagram);
    3.3.5. Diagrama de colaborare (Collaboration Diagram);

4. Diagrame de implementare (Implementation Diagrams);
4.1. Diagrama de componente (Component Diagram);
4.2. Diagrama de aplicaţie (de dezvoltare) (Deployment Diagram).