

# **MANAGEMENTUL PROIECTELOR SOFTWARE**

## **Sumar curs**

### **Capitolul 1. INTRODUCERE**

#### **1. Obiective. Definiții**

- 1.1 Activitate de producție SW
- 1.2 Cerințe esențiale pentru un proiect SW de succes
- 1.3 Definiții: Proces de producție, Proiect SW

#### **2. Necesitatea Managementului Proiectelor SW**

- 2.1 Rațiuni pentru organizarea dezvoltării unui proiect SW
- 2.2 Descompunerea unui proiect în activități constitutive
- 2.3 Roata calității (Roata lui Deming). Cercul fazelor unui ciclu de dezvoltare

#### **3. Procese, activități și sarcini într-un proiect software**

- 3.1 Standardul ISO/CEI 12207:1995
- 3.2 Definiții: Procese, Activități, Sarcini
- 3.3 Tipuri de procese
- 3.4 Procese primare: Achiziție, Aprovizionare, Dezvoltare, Utilizare, Întreținere
- 3.5 Procese suport: documentare, Managementul configurației software-ului, Asigurarea calității, Testare, Validare, Analiză comună, Audit, Rezolvarea problemelor
- 3.6 Procese organizaționale: Management, Infrastructură, Training, Îmbunătățire

#### **4. Procesul de Dezvoltare**

- 4.1 Activitățile procesului de dezvoltare: (1) Inițializarea procesului, (2) Analiza cerințelor software și de sistem, (3) Proiectare arhitectură de sistem, (4) Proiectare detaliată a software-ului, (5) Codare, (6) Testarea codului scris, (7) Integrare de sistem, (8) Test de integrare, (9) Instalare software, (10) Suport pentru sistemul de validare

#### **5. Cicluri de viață pentru procesul de dezvoltare a proiectelor software**

- 5.1 Ciclul de viață cascadă
- 5.2 Ciclul de viață „V”.
- 5.3 Ciclul de viață prototipizare
- 5.4 Ciclul de viață evolutiv
- 5.5 Ciclul de viață în spirală (Boehm)
- 5.6 Dezvoltarea formală de sistem
- 5.7 IEEE/EIA 12207 „Standardul pentru Tehnologia Informației - Procese ale ciclului de viață al software-ului”
  - 5.7.1 Ierarhia software standard
  - 5.7.2 Procesul de dezvoltare software IEEE/EIA 12207

#### **6. Procesul de Management**

- 6.1 Activități: (1) Inițierea și stabilirea domeniului de aplicare, (2) Planificare, (3) Execuție și control, (4) Analiză și evaluare, (5) Finalizare

#### **7. Leadership (conducere) vs. management**

- 7.1 Definirea termenilor generali
- 7.2 Definirea termenilor specifici

#### **Exercitiul 1**

## **Capitolul 2. TEHNOLOGII PENTRU DEZVOLTAREA PRODUSELOR SW**

### **1. Tehnologia MicroSoft pentru dezvoltarea produselor SW**

- 1.1 Model general
  - 1.1.1 Prezentare MS
  - 1.1.2 Filosofia Microsoft
  - 1.1.3 Obiectivele MS
- 1.2 MS Paradigma: Sincronizări frecvente și stabilizări periodice
  - 1.2.1 Paradigma de bază a abordării MS
  - 1.2.2 Tendințe în industria SW
  - 1.2.3 Abordarea de dezvoltare clasică în cascadă (secvențială).
  - 1.2.4 Facilități dorite pentru procesul de dezvoltare
- 1.3 Strategii și principii
  - 1.3.1 Prezentare generală a abordării dezvoltării sincronizate și stabilizate
  - 1.3.2 Definirea produsului și organizarea procesului de dezvoltare. Prima strategie MS
  - 1.3.3 Dezvoltarea și livrarea produselor. A doua strategie pentru SM
- 1.4 Tehnologia MS
  - 1.4.1 Obiectiv
  - 1.4.2 Reguli
  - 1.4.3 Modul de lucru
  - 1.4.4 Echipele MS
- 1.5 Concluzii
  - 1.5.1 Inovații MS
  - 1.5.2 Strategii competitive ale statelor membre
  - 1.5.3 Punctele slabe ale abordării de dezvoltare Microsoft.
  - 1.5.4 Avantajele tehnologiei MS

### **2.Tehnologia ORACLE pentru dezvoltarea produselor SW**

- 2.1 Probleme actuale ale industriei IT
  - 2.1.1 Probleme în industria IT
  - 2.1.2 Cauze evidențiate de experiența ORACLE
  - 2.1.3 Concluzie
- 2.2 ORACLE set de metodologii de dezvoltare
  - 2.2.1 Idee fundamentală
  - 2.2.2 ORACLE set de metodologii de dezvoltare
- 2.3 Responsabilitățile managerului de proiect
  - 2.3.1 Responsabilități
  - 2.3.2 Sarcini majore
- 2.4 Etapele proiectului ORACLE
  - 2.4.1 Planificare
  - 2.4.2 Execuție
  - 2.4.3 Finalizare
- 2.5 Procesele proiectului ORACLE
  - 2.5.1 Definiția procesului. Tipuri de procese
  - 2.5.2 Procesele în tehnologia ORACLE
  - 2.5.3 Observații generale
- 2.6 Concluzii

### **3. Procesul de dezvoltare Rational**

- 3.1 Calea Rațională
- 3.2 Ciclul de viață general al software-ului
  - 3.2.1 Cele două perspective
  - 3.2.2 Cicluri și faze
  - 3.2.3 Iterații
  - 3.2.4. Reconcilierea celor două perspective

- 3.2.5 Discriminatori
- 3.2.6 Efort și program
- 3.3 Fazele procesului rațional
  - 3.3.1 Faza de început
  - 3.3.2 Faza de elaborare
  - 3.3.3 Faza de construcție
  - 3.3.4 Faza de tranzitie
  - 3.3.5 Cicluri de evoluție
- 3.4 Activități în Procesul Rațional
- 3.5 Artefacte ale ciclului de viață
  - 3.5.1 Artefacte de management
  - 3.5.2 Artefacte tehnice
  - 3.5.3 Cerințe
- 3.6 Exemple de proces rațional
  - 3.6.1 Proces rațional pentru dezvoltarea software contractuală mare
  - 3.6.2 Proces rațional pentru un produs software comercial mic
- 3.7 Concluzie
- 3.8 Glosar
- 3.9 Acronime

#### **Exercițiul #2**

### **Capitolul 3. GESTIONAREA PROIECTELOR SOFTWARE**

#### **1. Introducere**

- 1.1 Scurtă istorie
- 1.2 Managementul Afacerii
- 1.3 Terminologie
- 1.4 Stabilirea regulilor de bază
- 1.5 Contractul
  - 1.5.1 Şablonul de contract
  - 1.5.2 Tipuri de preţuri
- 1.6 Drepturile și responsabilitățile clientilor
  - 1.6.1 Cine este Clientul
  - 1.6.2 Parteneriatul Client – Dezvoltare
  - 1.6.3 Drepturile clientului
  - 1.6.4 Responsabilitățile clientului
  - 1.6.5 Ce este despre Sign-Off
- 1.7 Dezvoltare de sus în jos
- 1.8 Un proiect ideal
- 1.9 Ciclul de viață al proiectului
- 1.10 Câteva documente cheie
  - 1.10.1 Testarea documentelor

#### **Exercițiul #4**

### **Capitolul 4. FAZA DE DEFINIRE**

#### **1. Faza de definire a obiectivelor**

#### **2. Analiza problemei**

- 2.1 Recomandări pentru activitatea de analiză
- 2.2 Documentul de specificare a problemei
- 2.3. Sarcinile analiștilor

#### **3. Activități de planificare a proiectului**

- 3.1 Sistemul
  - 3.1.1 Definiție
  - 3.1.2 Caracteristicile unui sistem
- 3.2 Instrumente de planificare

- 3.2.1 Schița planului de proiect
- 3.2.2 Diagrame cu bare
- 3.2.3 Diagrame de reper
- 3.2.4 Rețele de activitate

#### **4. Estimarea dimensiunii software-ului**

- 4.1 Context
- 4.2 Cadrul de estimare a mărimii
  - 4.2.1 Relația dimensiune-resurse
  - 4.2.2 O anumită experiență de estimare
  - 4.2.3 Criterii de estimare a mărimii
- 4.3 Metode de estimare a mărimii
  - 4.3.1 Metrici orientate spre dimensiune
    - 4.3.1.1 Metoda Wideband-Delphy
    - 4.3.1.2 Metoda Fuzzy-Logic
    - 4.3.1.3 Metoda standard-componentă
  - 4.3.2 Metrici orientate pe funcții
    - 4.3.2.1 Metoda Funcție-Punct
    - 4.3.2.2 Conversia punctului de funcție în SLOC
    - 4.3.2.3 Metoda punctului caracteristic
    - 4.3.2.4 Estimare bazată pe proxy

#### **5. Estimarea costurilor software**

- 5.1 Obiective de estimare a costurilor
- 5.2 Resursele unui proiect software
- 5.3 Elementele de cost ale unui proiect software
- 5.4 Tehnici de estimare a costurilor software
  - 5.4.1 Tehnici de analogie
  - 5.4.2 Tehnici de judecată expert
  - 5.4.3 Tehnici de jos în sus
  - 5.4.4 Tehnici de sus în jos
  - 5.4.5 Metoda combinată
  - 5.4.6 Legea Parkinson
  - 5.4.7 Prețul de câștig
- 5.5 Modele de evaluare a costurilor software
  - 5.5.1 Modele de descompunere
    - 5.5.1.1 Model EE (estimarea efortului)
    - 5.5.1.2 Modelele LOC și FP
  - 5.5.2 Modele parametrice
    - 5.5.2.1 Modelul COCOMO 81
    - 5.5.2.2 Modelul COCOMO II
    - 5.5.2.3 PRET S Model
    - 5.5.2.4 Modelul SEER SEM
- 5.6 Productivitatea activității software

#### **6. Ghid de estimare a proiectelor**

- 6.1 Istoricul proiectului

#### **7. Planul proiectului**

- 7.1 Caracteristicile unui plan bun
- 7.2 Redactarea planului de proiect
- 7.3 O schiță a planului de proiect
  - 7.3.1 Prezentare generală
  - 7.3.2 Planul de etape
  - 7.3.3 Planul de organizare
  - 7.3.4 Plan de testare
  - 7.3.5 Modificarea planului de control
  - 7.3.6 Plan de documentare
  - 7.3.7 Plan de instruire
  - 7.3.8 Planul de revizuire și raportare

- 7.3.9 Plan de instalare și exploatare
- 7.3.10 Resurse și plan de livrare
- 7.3.11 Indexul planului

## **8. Criterii de acceptare**

## **9. Tehnologia WBS**

- 9.1 Ce este WBS
- 9.2 Scopurile WBS
- 9.3 Abordări ale dezvoltării WBS
  - 9.3.1 Utilizarea ghidurilor
  - 9.3.2 Abordarea analogiei
  - 9.3.3 Abordarea de sus în jos
  - 9.3.4 Abordarea de jos în sus
- 9.4 Câteva principii de bază pentru a crea un WBS bun
- 9.5 Cum se stabilește WBS
- 9.6 Pachetele de lucru
- 9.7 Concluzii

**Exercițiul #5**

**Exercițiul #6**

**Exercițiul #7**

## **Capitolul 5. FAZA DE PROIECTARE**

### **1 Proiectarea sistemului**

- 1.1 Specificația de proiectare
- 1.2 Designerii
- 1.3 Mediul de proiectare
- 1.4 Ghid de proiectare
- 1.5 Instrumente de proiectare
  - 1.5.1 Diagrame de flux
  - 1.5.2 HIPO
  - 1.5.3 Pseudocod
  - 1.5.4 Diagrame structurate
  - 1.5.5 Diagrame de flux de date
  - 1.5.6 Tabel de decizie
  - 1.5.7 UML
  - 1.5.8 Matricea de acoperire
  - 1.5.9 Hărți de stocare
  - 1.5.10 Limbaje de programare
  - 1.5.11 Model de simulare
- 1.6. Evaluarea calității designului

### **2. Planificarea proiectului în fază de proiectare**

- 2.1 Schimbarea controlului
- 2.2 Pregătirea pentru testare
  - 2.2.1 Definirea ierarhiei testelor
  - 2.2.2 Testarea integrării de sus în jos vs. de jos în sus
  - 2.2.3 Scrierea specificațiilor de testare
  - 2.2.4 Definirea procedurilor de testare
  - 2.2.5 Furnizarea timpului de calculator
  - 2.2.6 Trasarea rezultatelor testelor
- 2.3 Estimarea resurselor
- 2.4 Documentare
  - 2.4.1 Manual de programare
  - 2.4.2 Biblioteca de proiect
- 2.5 Instruire

### **3. Revizia fazei de proiectare**

- 3.1 Pregătire

- 3.1.1 Programarea persoanelor
- 3.1.2 Programarea sălilor de întâlnire
- 3.1.3 Pregătirea suporturilor de prezentare
- 3.1.4 Pregătirea materialelor pentru fișe

3.2 Ce trebuie acoperit

3.3 Rezultate

### **Exercițiul #8**

## **Capitolul 6. FAZA DE PROGRAMARE**

### **0 Introducere**

### **1 Tehnici de programare**

1.1 Programare structurată

- 1.1.1 Obiectivele programării structurate

1.2 Programare, proiectare și analiză orientată pe obiecte

- 1.2.1 Programare orientată pe obiecte

- 1.2.2 Proiectare orientată pe obiecte

- 1.2.3 Analiza orientată pe obiecte

### **2 Modalitati de organizare**

2.1 Organizare convențională

2.1.1 Grupul de analiză și proiectare

- 2.1.1.1 Controlul modificării

- 2.1.1.2 Controlul datelor

- 2.1.1.3 Proceduri structurate și inspecții

- 2.1.1.4 Modelarea prin simulare

- 2.1.1.5 Documentația utilizatorului

2.1.2 Grupul de programare

- 2.1.2.1 Proiectare detaliată

- 2.1.2.2 Codificare

- 2.1.2.3 Testul modulului

- 2.1.2.4 Documentație

- 2.1.2.5 Integrare: „De sus în jos”

- 2.1.2.6 Integrare: „De jos în sus”

- 2.1.2.7 Integrare: Specificația Testului

2.1.3 Grup de testare

2.1.4 Grupul de personal

- 2.1.4.1 Funcțiile personalului tehnic

- 2.1.4.2 Funcțiile personalului administrativ

2.1.5 Jocul numerelor

2.2 Organizarea echipei. Echipa de programatori șef

2.2.1 Cum funcționează

2.2.2 Organizarea proiectului folosind abordarea echipei de programatori șef

### **3 Modificați de control**

3.1 Documente de referință

3.2 Proceduri de control

### **4 Instrumente de programare**

4.1 Specificații scrise

4.2 Directori de testare

4.3 Simulatoare de mediu

4.4 Mediu de programare specializat

4.5 Ajutoare automate de documentare

4.6 Monitoare software și hardware

4.7 Biblioteca de proiect

- 4.7.1 Biblioteca generală

- 4.7.2 Biblioteca de sprijin pentru dezvoltare

### **5 Atribuțiile managerului**

- 5.1 Conducere tehnică
- 5.2 Planificare și control
- 5.3 Comunicarea
- 5.4 Asigurarea condițiilor și instrumentelor de muncă
- 5.5 Atribuirea lucrării
  - 5.5.1 Atribuirea persoanelor
  - 5.5.2 Atribuirea domeniilor
  - 5.5.3 Obiectivele sarcinii de lucru
- 5.6 Programul de lucru
  - 5.6.1 Alocarea orelor normale de lucru
  - 5.6.2 Program de lucru suplimentar
  - 5.6.3 Tehnica Flexy-Time
  - 5.6.4 Tehnica dead-line
- 5.7 Adăugarea mai multor persoane
- 5.8 Raportarea stării tehnice
  - 5.8.1 Rapoarte scrise
  - 5.8.2 Recenziile orale
- 5.9 Raportarea situației financiare
- 5.10 Instruire
  - 5.10.1 Instruirea personalului tehnic
  - 5.10.2 Formarea managerilor
- 5.11 Evaluare și consiliere
- 5.12 Întreținerea Sanității
- 5.13 Niveluri de management

**Exercițiu #10**

**Capitolul 7. FAZA DE TESTARE A SISTEMULUI**

**1 Testarea sistemului**

- 1.1 Specificații de testare a sistemului
- 1.2 Testerii
- 1.3 Condiții de testare
- 1.4 Efectuarea Testelor

**2 Instruirea clientilor**

- 2.1 Utilizarea sistemului
- 2.2 Întreținerea sistemului

**Exercițiu #11**

**Capitolul 8. FAZA DE ACCEPTANȚĂ**

**1 Testarea de acceptanță**

- 1.1 Specificația testului de acceptare
- 1.2 Criterii de acceptare
- 1.3 Execuție

**2 Documentație**

**Exercițiu #12**

**Capitolul 9. FAZA DE INSTALARE SI OPERARE**

**1 Testarea site-ului**

**2 Conversia**

- 2.1 Funcționare în paralel
- 2.2 Înlocuire imediată
- 2.3 Decupare

**3 Întreținere și reglare**

**4 Evaluarea proiectului**

**Exercițiu #13**

## **Capitolul 10. CONSIDERAȚII SPECIALE**

### **1 Proiecte mari**

- 1.1 Fazele
- 1.2 Organizare
- 1.3 Controlul clientului
- 1.4 Managementul configurației
- 1.5 Lansări multiple

### **2 Proiecte mici**

### **3 Propunerea de proiect**

- 3.1 Ghid pentru redactarea unei propuneri

### **Exercițiu #14**

# **MANAGEMENTUL PROIECTELOR SOFTWARE**

## **Capitolul 1. INTRODUCERE**

### **1. Obiective. Definiții**

- 1.1 Activitate de producție SW
- 1.2 Cerințe esențiale pentru un proiect SW de succes
- 1.3 Definiții: Proces de producție, Proiect SW

### **2. Necesitatea Managementului Proiectelor SW**

- 2.1 Rațiuni pentru organizarea dezvoltării unui proiect SW
- 2.2 Descompunerea unui proiect în activități constitutive
- 2.3 Roata calității (Roata lui DEMING). Cercul fazelor unui ciclu de dezvoltare

### **3. Procese, activități și sarcini într-un proiect software**

- 3.1 Standardul ISO/CEI 12207:1995
- 3.2 Definiții: Procese, Activități, Sarcini
- 3.3 Tipuri de procese
- 3.4 Procese primare: Achiziție, Aprovizionare, Dezvoltare, Utilizare, Întreținere
- 3.5 Procese suport: documentare, Managementul configurației software-ului, Asigurarea calității, Testare, Validare, Analiză comună, Audit, Rezolvarea problemelor
- 3.6 Procese organizaționale: Management, Infrastructură, Training, Îmbunătățire

### **4. Procesul de Dezvoltare**

- 4.1 Activitățile procesului de dezvoltare: (1) Inițializarea procesului, (2) Analiza cerințelor software și de sistem, (3) Proiectare arhitectură de sistem, (4) Proiectare detaliată a software-ului, (5) Codare, (6) Testarea codului scris, (7) Integrare de sistem, (8) Test de integrare, (9) Instalare software, (10) Suport pentru sistemul de validare

### **5. Cicluri de viață într-un proces de dezvoltare a unui proiect SW**

- 5.1 Ciclul de viață cascadă
- 5.2 Ciclul de viață „V”.
- 5.3 Ciclul de viață Prototipare
- 5.4 Ciclul de viață evolutiv
- 5.5 Ciclul de viață în spirală (Boehm)
- 5.6 Dezvoltarea formală
- 5.7 IEEE/EIA 12207 „Standardul pentru Tehnologia Informației - Procese ale ciclului de viață al software-ului”
  - 5.7.1 Ierarhia software standard
  - 5.7.2 Procesul de dezvoltare software IEEE/EIA 12207

### **6. Procesul de Management**

- 6.1 Activități: (1) Inițierea și stabilirea domeniului de aplicare, (2) Planificare, (3) Execuție și control, (4) Analiză și evaluare, (5) Finalizare

### **7. Leadership (Conducere) vs. management**

- 7.1 Definirea termenilor generali
- 7.2 Definirea termenilor specifici

## **Exercitiul 1**

# Capitolul 1. INTRODUCERE

## 1. Obiective. Definiții

### 1.1 Activitatea de producție software

- Are un caracter inovativ
- Necesită o cantitate mare de creativitate
- Tendința inițială a fost aceea de a dezvolta SW într-o manieră artizanală. Rezultate negative:
  - Integrarea dificilă a programatorilor în echipe și organizații care dezvoltă SW
  - Complexitatea ridicată a dezvoltării SW, în absența unui șablon organizational real, poate determina o risipă de resurse
- **Particularități ale producție SW:**
  - (1) Necesită Eficiență și Flexibilitate legate de:
    - Domeniul de aplicație standard
    - Platforma de dezvoltare standard
    - Procesul de dezvoltare standard
    - Mediu de dezvoltare standard
  - (2) Induce impact neprevăzut
    - Modificarea cerințelor de către client
    - Schimbarea platformei de dezvoltare de către management
    - Fluctuația personalului (de exemplu, schimbarea angajaților)
    - Modificarea datelor de livrare de către conducere
    - Întârzieri cauzate de furnizori interni și externi
  - (3) Natura imaterială a unui produs SW ()
    - Caracteristici cu greu experimentate de client (relația costurilor)
    - Rezultate intermedii greu de măsurat (conform rezultatului final)
    - Greu de evaluat
    - Dificil de introdus în contabilitate. A apărut pentru software termenul "Activ noncorporal"
    - Introduce riscuri legate de setul de caracteristici, costuri, durată
  - (4) Necesită un anumit nivel de inovare
    - Utilizează cele mai noi tehnologii
    - Realizează noi funcții de la zero
    - Introduce riscuri legate de calitate scăzută și de comportament în timpul rulării
  - (5) Lipsa maturității procesului de dezvoltare
    - Lipsește un Proces de dezvoltare standardizat și stabilit.
    - Necunoașterea practicilor de inginerie SW
    - Introduce riscuri legate de calitate scăzută și de comportament în timpul rulării

- **Modelului de Maturitate al Capabilității (CMMI)**

- CMMI este un model de maturitate a procesului care urmărește să ajute organizațiile să își îmbunătățească performanța.
- CMMI descrie o cale de îmbunătățire prin evoluție a proceselor imature către un proces matur, mai bine organizat.
- Majoritatea organizațiilor și corporațiilor de software mari au adoptat CMMI.
- Nivelurile CMMI se referă la un set de pași pe care fiecare organizație îi poate obține pentru a-și crește valoarea pe piață.
- CMMI conține cinci niveluri de maturitate ilustrate în figura 1.

## CMMI Staged Maturity Levels

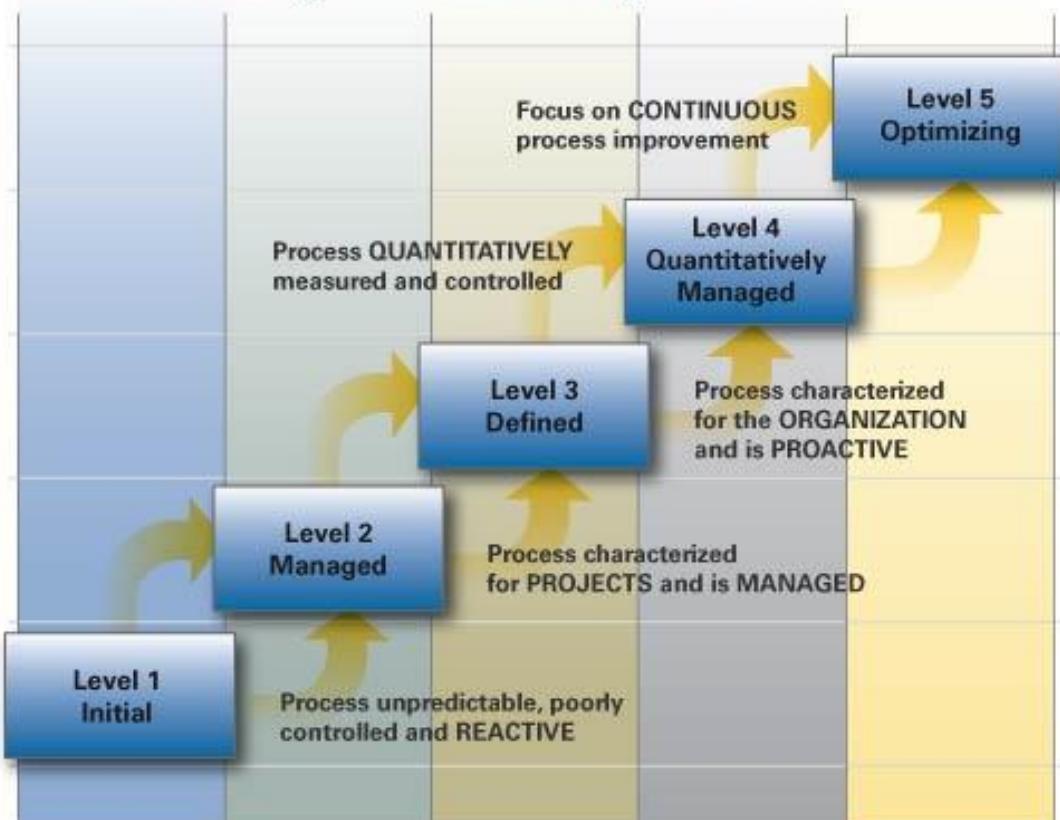


Fig. 1. Caracteristicile nivelurilor de maturitate

- **Standardul ISO 9000**

- ISO 9000 este o familie de standarde pentru sistemele de management al calității.
- Descrie elementele fundamentale ale sistemelor de management al calității și definește termenii aferenți.
- Promovează angajamentul managementului superior față de calitate, orientarea către client, adevararea resurselor, competența angajaților, managementul proceselor, planificarea calității, proiectarea produselor, revizuirea comenzielor primite, achiziția, monitorizarea și măsurarea proceselor și produselor

## 1.2 Cerințe esențiale pentru un proiect SW de succes

- (1) Acoperă exact nevoile clientului
- (2) Se finalizează la timp
- (3) Costurile de dezvoltare nu depășesc bugetul alocat
  - Pentru a îndeplini aceste cerințe este necesar un **proces organizațional** riguros.

## 1.3 Definiții

- Definiția clasică a procesului de producție:
  - **Definiție:** **Proces de producție** – Un termen care descrie procesul de producție a bunurilor materiale. De obicei, presupune un efort susținut de a replica un prototip dezvoltat într-o fază anterioară.
- Pentru producția SW termenul uzual este **Project SW**
  - **Definiție1: Project SW** – Un ansamblu de activități organizate, legate de dezvoltarea de programe, părți de programe sau sisteme de programe, cu scopul principal de a obține caracteristici bine definite pentru produsele dezvoltate.
  - **Definiția 2: Project SW** - Un proiect este o propunere, care se caracterizează prin unicitatea condițiilor sale, de exemplu:
    - Tintă/Obiectiv,
    - Limite în termeni de timp/resurse financiare/personal/altele,
    - Diferențiere față de alte propunerii,
    - Organizare specifică proiectului.

(German Industrial Norm, #69 901)

## 2. Necesitatea managementului proiectelor software

### 2.1 Rațiuni pentru organizarea dezvoltării unui proiect SW

- Estimarea inițială (când este gata, cât costă)
- Factorizarea activităților (determinată de complexitatea proiectului care la rândul său este determinată de complexitatea cerințelor funcționale)
- Execuția proiectului (pregătire, planificare, supraveghere, control)
- Managementul configurației (integritatea proiectului)
- Controlul programului planificat (timp, buget)
- Controlul calității produsului (\*) (Managementul calității)  
(\*) **Definiție:** calitatea este măsura modului în care produsul SW satisface cerințele impuse de utilizatori, clienți, cumpărători sau beneficiari.

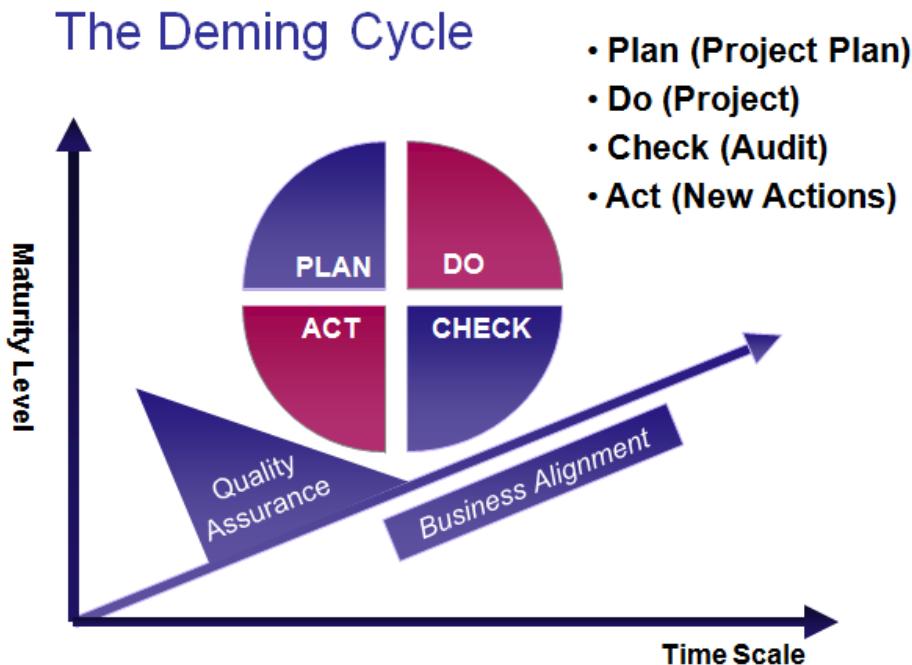
### 2.2 Descompunerea unui proiect în activități constitutive

- **Managementul de proiect** este de fapt un **proces de dezintegrare** care presupune în același timp un **proces organizațional** care gestionează **interdependențele** dintre părțile sau activitățile rezultante.
  - Interdependențele pot fi legate de **timp** sau de **resursele umane** și resursele **materiale** necesare desfasurării activitatilor.
- De obicei, dezvoltarea unui produs SW se realizează printr-un **proces de dezvoltare** care constă în mai multe **faze**.

- Aceste faze sunt prezentate într-o manieră intuitivă în roata calității a lui Deming.

### 2.3 Roata calității (Roata lui Deming). Cercul fazelor unui ciclu de dezvoltare

- Roata Calității (Roata lui Deming) apare în fig. 2.3.a și constă în 4 faze.



**Fig.2.3.a. Roata lui Deming**

- Faza **Planifică** presupune:
  - Definirea produsului
  - Estimarea riguroasă a resurselor
  - Stabilirea interdependentelor între activități și/sau costuri
  - Stabilirea condițiilor fazelor finale (estimarea timpului și criteriile de ieșire a fazelor)
  - Stabilirea operațiunilor în cadrul acțiunilor și a actorilor implicați (estimarea costurilor și a resurselor umane)
- (2) Faza **Execută** – dezvoltarea eficientă a produsului. De obicei, este singura fază acceptată de programatori
- (3) Faza **Verifică**
  - Estimarea calității
  - Măsurători ale produsului dezvoltat
  - Determinarea costurilor
  - Stabilirea dacă au fost respectate termenele programate
- (4) Faza **Acționează**
  - Analiza informațiilor obținute în faza anterioară
  - Corectarea și rezolvarea deficiențelor de definire sau a problemelor procesului de dezvoltare
- **Observație:**
  - Trei din cele patru faze sunt de fapt legate de **management**

- Fazele de management au ca efect:
  - Creșterea efortului de dezvoltare (costuri, timp, resurse)
  - Inducerea de efecte favorabile prin creșterea fiabilității procesului de dezvoltare

### 3. Procese, Activități și Sarcini (Taskuri) într-un proiect software

#### 3.1 Standardul ISO/IEC 12207: 1995

- Multitudinea și complexitatea problemelor legate de dezvoltarea unui produs SW a presupus necesitatea unei abordări sistematice și a standardizării.
- Rezultatul a fost **Standardul ISO/IEC 12207:1995** având ca scop principal stabilirea pentru industria SW:
  - Un cadru comun de dezvoltare
  - O terminologie bine definită.

#### 3.2 Definiții

- **Definiții:** În conformitate cu acest standard, un **PROIECT SW** constă în:
- (1) **Procese** – un ansamblu de resurse și activități interdependente orientate către un scop bine definit.
- (2) **Activități** – sunt părți ale unui proces constând în tipuri de acțiuni prin care resursele procesului sunt utilizate în scopul proiectului.
- (3) **Sarcini (task-uri)** – sunt componente ale activităților constând într-o sau un ansamblu de acțiuni.
  - O sarcină poate fi legată de o persoană sau un grup de persoane având responsabilitatea îndeplinirii lor
  - Pentru orice sarcină trebuie stabilită sau estimată
    - O alocare de resurse
    - Un orizont de timp
    - Un cost

#### 3.3 Tipuri de procese

- (1) **Procese primare (P)**
- (2) **Procese sistem (S)**
- (3) **Procese organizaționale (O)**

#### 3.4 Procese primare

- **Procesele primare** sunt procesele care deservesc principalele părți (actorii) unui proiect SW: *achizitor, furnizor, dezvoltator, operator (utilizator) și întreținător al produsului*.
- **Standardul ISO/CEI 12207:1995** definește **5 procese primare**:
  - (1) **Procesul de achiziție** – definește activitățile prin care o organizație achiziționează un sistem, un produs sau un serviciu SW
  - (2) **Procesul de aprovizionare** – definește activitățile prin care o organizație furnizează un sistem, un produs sau un serviciu SW
  - (3) **Procesul de dezvoltare** – constă în activități prin care o organizație definește și elaborează un sistem, un produs sau un serviciu SW
  - (4) **Procesul de utilizare** – definește activitățile prin care o organizație utilizează un sistem, un produs sau un serviciu SW

- (5) **Proces de întreținere** – definește activitățile prin care o organizație furnizează servicii de întreținere pentru un sistem, un produs sau un serviciu SW

### 3.5 Procese suport

- **Procesele suport** sunt procese care sprijină alte procese. Ele contribuie la succesul și calitatea unui proiect SW.
- **Standardul ISO/CEI 12207:1995** definește **8 Procese suport**:
  - (1) **Procesul de documentare** – include activitățile privind definirea și înregistrarea tuturor informațiilor rezultate din procesul de dezvoltare a SW.
    - Asta presupune documentația utilizatorului, precum și documentele legate de procesul de dezvoltare: planuri, rapoarte, specificații, standarde interne, documente asociate, proceduri interne.
  - (2) **Procesul de management al configurației SW (SCM)** – constă în proceduri administrative și tehnice care realizează:
    - Identificarea, definirea și stabilirea elementelor de configurare SW (componente, module, unități, fișiere, structuri de date)
    - Controlează depozitarea, manipularea și livrarea componentelor SW
    - Stabilirea versiunilor de produs
    - Stabilirea stării componentelor (funcționalități, disfuncționalități, erori)
    - Controlează modificările la trecerea de la o versiune la alta (Control Versions Management)
  - (3) **Procesul de asigurare a calității (QA)** – definește ansamblul activităților care asigură într-o manieră obiectivă că
    - Produsul SW realizat îndeplinește cerințele specificate
    - Procesele implicate respectă un set de planuri și proceduri stabilite
  - (4) **Procesul de testare** (\*) – definește ansamblul activităților având ca scop verificarea produselor rezultate din desfășurarea activităților, care îndeplinesc cerințele și condițiile impuse.
    - Verificarea are grade diferite de adâncime în funcție de activitatea al carei produs este testat
  - (5) **Procesul de validare** (\*) – definește ansamblul activităților care verifică dacă un produs SW aflat într-o fază finală, satisfac cerințele de utilizare planificate (acoperă nevoile utilizatorului rezultate în urma procesului de analiză)
  - (6) **Procesul de analiză comună** (\*) – este procesul de analiză/evaluare a stării unui proces sau a unui produs.
    - Este un proces periodic care implică părțile implicate în proiect (de obicei dezvoltatorul, beneficiarul și cumpărătorul sau furnizorul)
    - Se concentrează fie pe analiza cerințelor produsului SW, fie pe măsurarea „pulsului” proiectului
  - (7) **Procesul de audit** (\*) – cuprinde activitățile orientate spre certificarea conformității cu normele, cerințele, graficele și declarațiile contractului pentru un produs sau un proces SW.
    - În principiu, aceste activități sunt similare cu cele realizate prin procese de testare, validare sau analiză, cu următoarele diferențe:
      - (1) Se realizează pe parcursul desfășurării activității sau sarcinii, și nu la final, ca în cazul procesului de testare sau de validare.
      - (2) Partea de audit nu are responsabilități directe în produsele și procesele implicate, element care diferențiază procesul de audit de cel comun de analiză.

- (8) **Procesul de rezolvare a problemelor** (\*) – include activități privind analiza și rezolvarea problemelor (neconformități, erori funcționale, situații neașteptate)
- **Obs.** Procesele marcate cu (\*) (testare, validare, analiză comună, audit, rezolvare a problemelor) pot fi utilizate ca tehnici pentru procesul de asigurare a calității

### 3.6 Procese organizaționale

- **Procesele organizaționale** sunt procese legate de management, infrastructură, instruire și îmbunătățire
- **Standardul ISO/CEI 12207:1995** definește **4 procese organizaționale**:
  - (1) **Procesul de management** – definește activitățile de bază legate de managementul oricărui proces
  - (2) **Procesul de infrastructură** – constă în ansamblul activităților privind stabilirea, realizarea și menținerea infrastructurii oricărui proces.
    - Prin infrastructură înțelegem hardware, software, instrumente, tehnici, standarde și facilități de dezvoltare, exploatare și întreținere
  - (3) **Procesul de formare** – precizează ansamblul activităților de formare și menținere a nivelului profesional al personalului.
    - Efortul principal este îndreptat spre îmbunătățirea cunoștințelor și creșterea calificării personalului.
  - (4) **Procesul de îmbunătățire** – constă în ansamblul activităților orientate spre definirea, evaluarea, măsurarea, controlul și îmbunătățirea oricărui proces.

## 4. Procesul de dezvoltare

- Apartine **Proceselor Primare**.
- Este partea principală a întregului proiect SW care implică cel mai mare sprijin din partea celorlalte procese.
- Constă într-un număr de **activități specifice**.
- Este condus și controlat de către **Procesul de Management** (O).

### 4.1 Activitățile procesului de dezvoltare

- (1) **Inițierea procesului** – presupunând:
  - (a) Selectarea și utilizarea unui **model de ciclu de viață** în conformitate cu dimensiunea, complexitatea și domeniul de aplicare al produsului SW care urmează să fie dezvoltat
  - (b) Elaborarea **Planului de Dezvoltare a Proiectului** pe baza specificațiilor **Procesului de Documentare** (S), constând în:
    - **Standarde, metode și instrumente specifice** utilizate în dezvoltare. De obicei, acestea sunt rezultate ale **Procesului de infrastructură** (P)
    - **Factorizarea acțiunilor** procesului în sarcini (taskuri):
      - Identificarea cunoștințelor și aptitudinilor necesare îndeplinirii sarcinilor
      - Stabilirea programării sarcinilor
      - Identificarea persoanelor responsabile cu realizarea fiecărei sarcini, pe baza estimării competențelor necesare.
    - Dacă **echipa trebuie instruită**, aceasta se realizează ca parte a **Procesului de Formare** (O)

- **Identificarea ieșirilor Procesului de Dezvoltare**, programarea și specificarea acestora, sau trimiterea **Procesului de Management al Configurației** (S) dacă este necesar
  - **Identificarea rezultatelor livrabile ale Procesului de Dezvoltare** (P) și specificarea caracteristicilor acestora
- (2) **Analiza SW și analiza cerințelor de sistem**
  - Rezultatul acestei activități este documentul numit **Specification (Problem Specification, SPEC)**.
  - Acest document este realizat conform **Procesului de documentare** (S) și include:
    - Caracteristici și capabilități ale sistemului și SW
    - Cerințe de securitate, ergonomie și de afaceri
    - Cerințe organizaționale
    - Cerințe de interfață cu utilizatorul, alte componente, alte sisteme SW existente
    - Cerințe de exploatare și întreținere
    - Cerințe privind documentația utilizator
  - Această activitate face parte din Procesul comun de analiză (S), deoarece de obicei nu se dezvoltă o singură soluție, ci **o clasă de soluții** care rezolvă multitudinea cerințelor problemei, din care trebuia selectată soluția optimă.
  - Definirea **Planului de Testare de Validare** a cărui elaborare este considerată parte a acestei activități
- (3) **Proiectarea arhitecturii sistemului** – constă în elaborarea unui set de documente referitoare la:
  - **Componentele HW** ale sistemului și modalitățile lor de interconectare
  - **Elemente de configurare SW** și alocarea lor la componentele HW
  - **Operațiuni manuale** permise de sistem
  - Elemente de configurare utilizator și **Interfețe SW**
  - **Arhitectura de nivel înalt a elementelor de configurare SW** (componentele acestora), interfețe între componente și structura generală a bazei de date a acestora (dacă este necesar)
  - Versiunea preliminară a **manualelor de utilizare și de administrare**
  - Versiunea preliminară a **Planului de testare de integrare**.
- (4) **Proiectare SW Detaliată** – constă în elaborarea unui set de documente care detaliază proiectarea de bază. Se referă la:
  - **Proiectarea la nivel de detaliu a fiecărei componente SW** identificate în faza de proiectare. Asta presupune:
    - Descompunerea componentelor în unități SW (nivelul de detaliu care ajunge la clase în abordarea OO),
    - Precizarea rolului, a interfeței, precum și ciclul de viață specific pentru fiecare unitate.
    - Proiectarea detaliată trebuie să permită codificarea directă a componentelor fără alte informații suplimentare.
  - Proiectarea detaliată a **structurii bazei de date**.
  - Realizarea **Planului de testare al unităților SW** desemnat pentru a testa unitățile SW
  - Actualizarea **Planului de Testare de Integrare**
- (5) **Codificare** – se referă la codificarea componentelor SW

- (6) **Testarea codului scris** – se mai numește și **Test de calificare SW**. Se realizează pe baza **Planului de testare al unităților SW**
  - Rezultatele testelor sunt documentate ca rapoarte de testare
  - Problemele întâlnite (bug-uri) sunt rezolvate în baza **Procesului de rezolvare a problemelor (S)**
- (7) **Integrarea sistemului** – presupune activități legate de integrarea elementelor SW cu elementele HW și cu celelalte sisteme existente
- (8) **Testarea de integrare** – cunoscut și sub denumirea de **Test de calificare a sistemului**
  - Presupune verificarea corectitudinii funcționalității sistemului ca întreg.
  - Se bazează pe **Planului de Testare de Integrare**.
  - Rezultatele testelor sunt documentate ca rapoarte de testare
  - Problemele întâlnite (bug-uri) sunt rezolvate pe baza **Procesului de rezolvare a problemelor (S)**
- (9) **Instalare SW** – presupune instalarea și configurarea produsului SW în mediul țintă. Sarcinile tipice ale acestei activități sunt:
  - Elaborarea **Planului de Instalare** care se referă la:
    - Precizarea acțiunilor și resurselor necesare
    - Împărțirea responsabilităților între dezvoltator și cumpărător (utilizator)
    - Stabilirea condițiilor pentru migrarea datelor în cazul în care se înlocuiește un sistem vechi existent
  - Instalarea eficientă a sistemului în conformitate cu **Planul de instalare**
  - Evenimentele și rezultatele procesului de instalare sunt înregistrate în documente specifice în conformitate cu **Procesul de documentare**
- (10) **Validarea Sistemului Suport** care este oferit de dezvoltator utilizatorului sistemului. Conține din:
  - Asistenta în executarea testelor de validare
  - Validarea conformității sistemului cu cerințele specificate
  - Rezultatele testelor sunt înregistrate în documente specifice în conformitate cu **Procesul de documentare (S)**
  - Orice problemă întâlnită este rezolvată în conformitate cu **Procesul de rezolvare a problemelor (S)**
  - Încheierea cu succes a acestei activități, de obicei presupune sfârșitul **Procesului de Dezvoltare (P)**

## 5. Cicluri de viață pentru procesul de dezvoltare a proiectelor software

- Indiferent de modul în care se realizează dezvoltarea software-ului, acesta trebuie să treacă prin anumiți **pași** sau **faze de dezvoltare**.
- După ce software-ul este dezvoltat, acesta trebuie să fie **suportat** (adică, întreținut).
  - Combinația de faze de dezvoltare a software-ului și fazele activităților de asistență este denumită **Ciclul de viață al software-ului (\*) SWLC sau SWPLC** (Ciclul de viață al proiectului software)
- **(\*)Definiție:** *Un Ciclu de viață SW este descrierea abstractă a procesului structurat și metodologic de dezvoltare și modificare, care arată de obicei etapele principale în*

*producerea și întreținerea software-ului executabil* – (John McDermid, „The Software Engineer’s Reference Book”)

- Ciclul de viață SW face parte din **Procesul de Dezvoltare SW** (P), de fapt este o activitate a procesului menționat
- Există o serie de tehnici de dezvoltare software pe care organizațiile le pot folosi, fiecare având un impact diferit asupra costurilor software
- În practică, logica organizării temporale a activităților în timpul **Procesului de Dezvoltare a Proiectului SW** (P) a impus dezvoltarea unor şabloane (modele) pentru ciclurile de viață ale proiectelor (PLC) aplicabile diferitelor tipuri de proiecte. În continuare sunt prezentate câteva din modelele reprezentative.

### 5.1 Ciclul de viață cascadă

- În figura 5.1.a. apare reprezentat un model generic de ciclu cascadă.

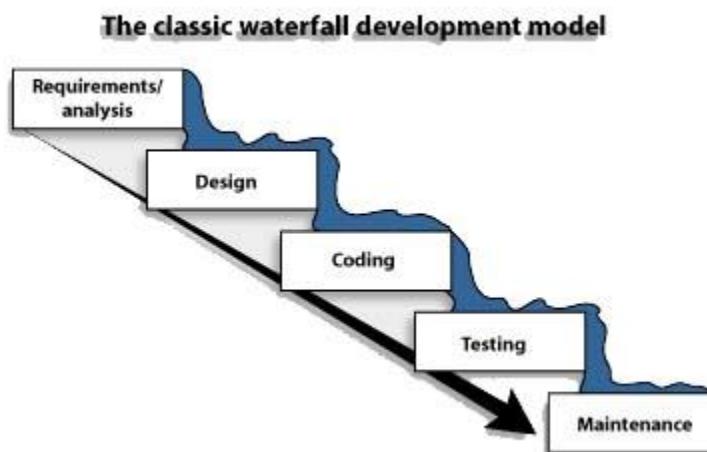


Fig. 5.1.a. Ciclul de viață Cascadă

- **Ciclul de viață CASCADĂ** – este modelul liniar clasic, cel mai vechi ciclu de viață. Aplicabil pentru:
  - Proiecte de complexitate redusă
  - Cerințe inițiale foarte bine definite

### 5.2 Ciclul de viață „V”

- **Ciclul de viață „V”** – este tot un ciclu liniar. Presupune:
  - Un set de cerințe inițiale foarte bine specificate
  - Un utilizator cu disponibilitate de a colabora și de a participa eficient la procesul de specificare a proiectului

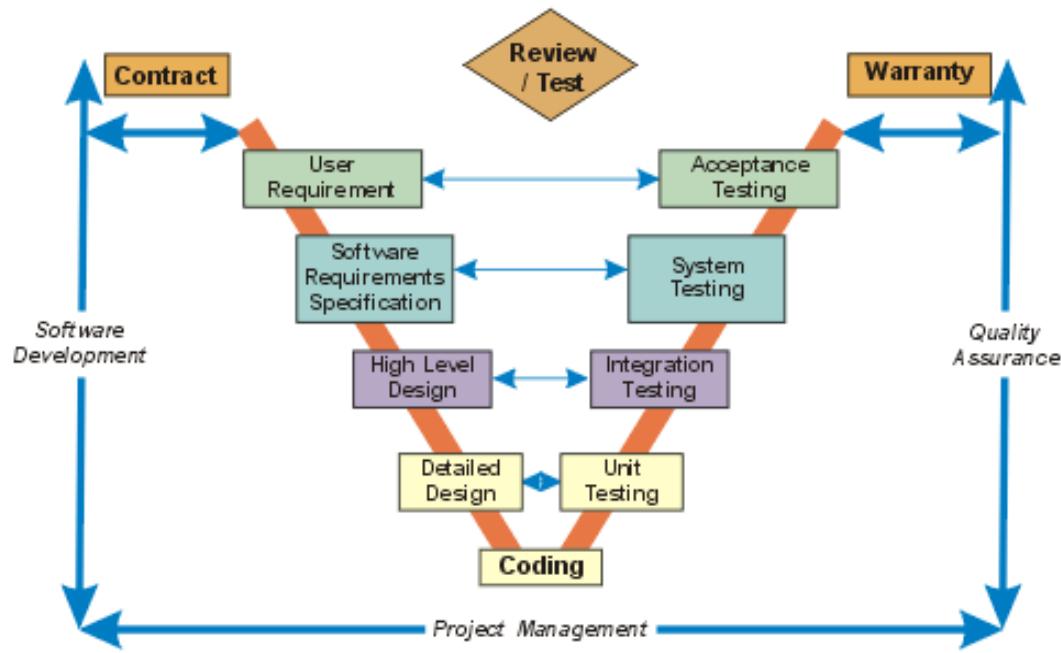
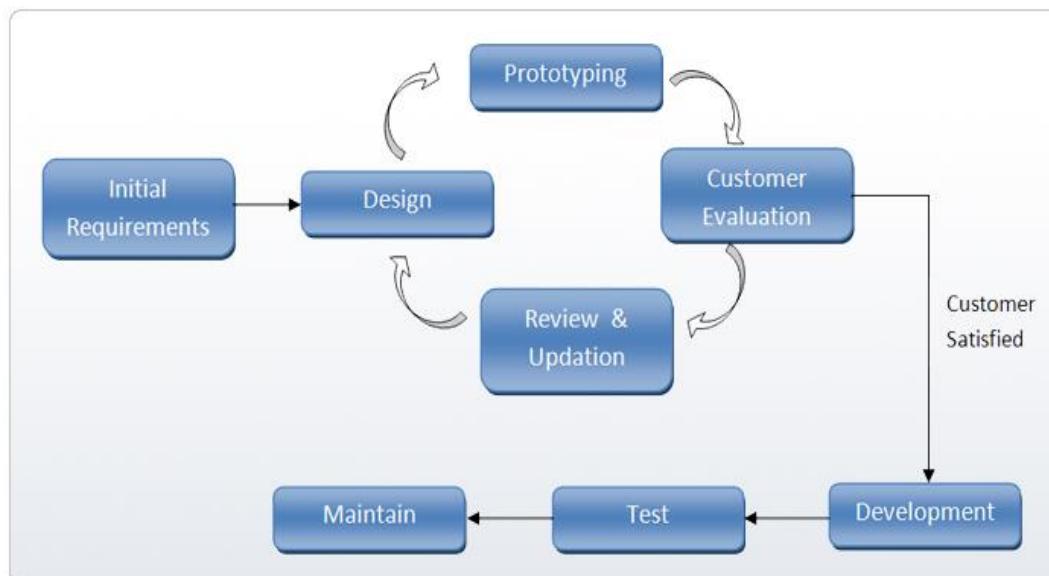


Fig.5.2.a. Ciclul de viață "V"

### 5.3 Ciclul de viață prototipizare

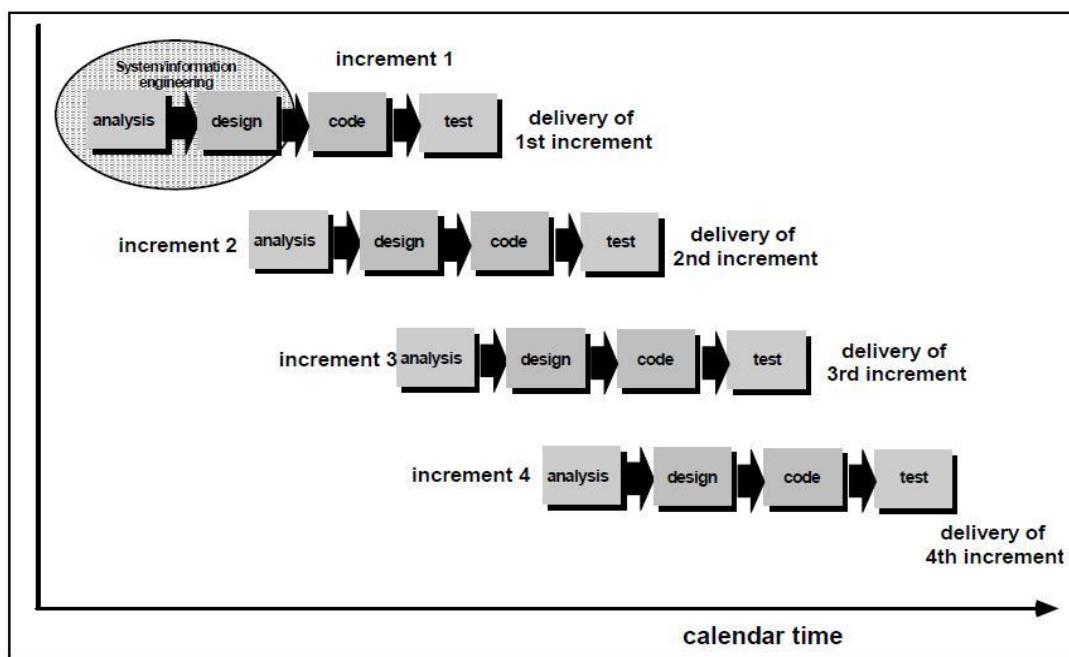
- Ciclul de viață prototipizare – este recomandat în cazul proiectelor la care clientul nu poate participa sau nu este interesat să realizeze o listă de cerințe bine definite
- Activitățile de analiză și de proiectare sunt iterative.
- Rezultatul este un ciclu de „**obținere-validare-corecție**” aplicabil prototipului de produs
- Necesită instrumente specifice de dezvoltare rapidă și eficientă a prototipurilor
- Poate fi folosit ca ciclu de dezvoltare preliminară, urmat de un alt tip de ciclu pentru dezvoltarea finală a produsului



**Fig.5.3.a.** Ciclul de viață prototipizare

#### 5.4. Ciclul de viață evolutiv

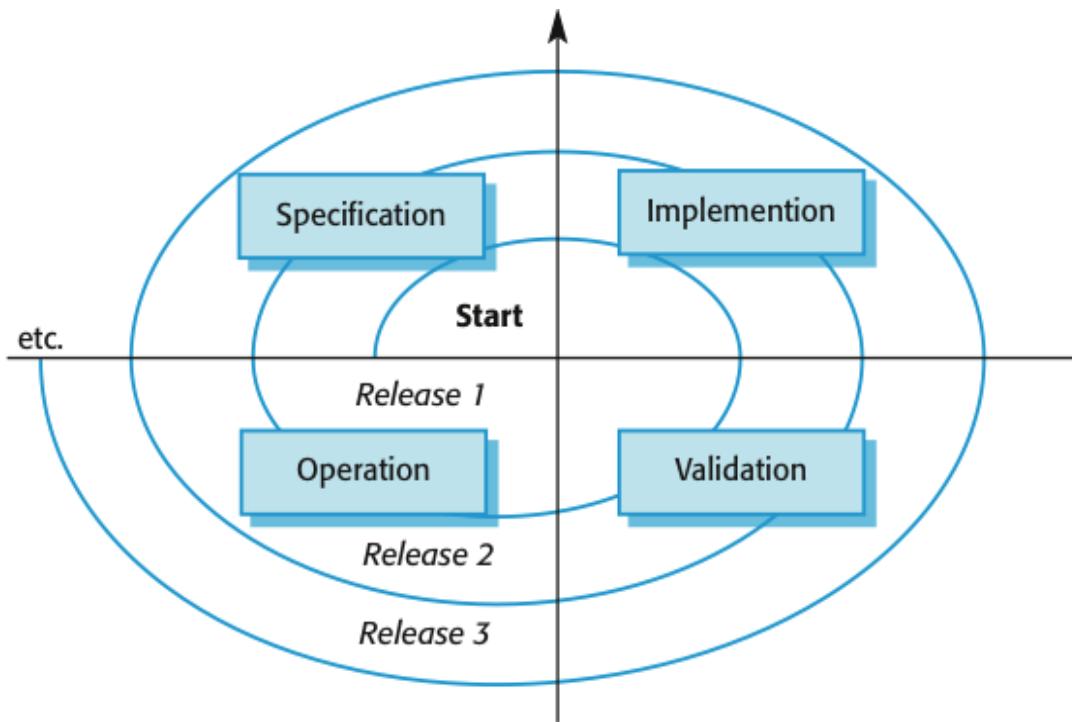
- Se bazează pe versiuni ale produsului
- Fiecare versiune este derivată rapid
- Acest proces se numește iterație
- Fiecare versiune este trimisă clientului care o analizează un feed-back de consumabile
- Fiecare iterăție se bazează pe un ciclu de viață al cascadei
- RUP (Rational Development Process) se bazează pe acest model



**Fig.5.4.a.** Ciclul de viață Evolutiv

#### 5.5 Ciclul de viață în spirală (Ciclul lui Boehm)

- Ciclul de viață în spirală (Ciclul lui Boehm) – este recomandat pentru proiecte:
  - Cu riscuri mari de dezvoltare
  - Cu o complexitate ridicată
  - Care necesită tehnologii foarte speciale
  - Pentru care nu se cunoaște cu precizie modalitatea de rezolvare a cerințelor clientilor
  - Sunt foarte scumpe
  - Ciclul Spiral poate fi considerat ca o repetare a ciclurilor liniare, fiecare nou ciclu adaugat spiralei, adăugând în același timp noi facilități produsului



**Fig. 5.3.a.** Ciclul de viață în spirală

## 5.6 Dezvoltarea formală

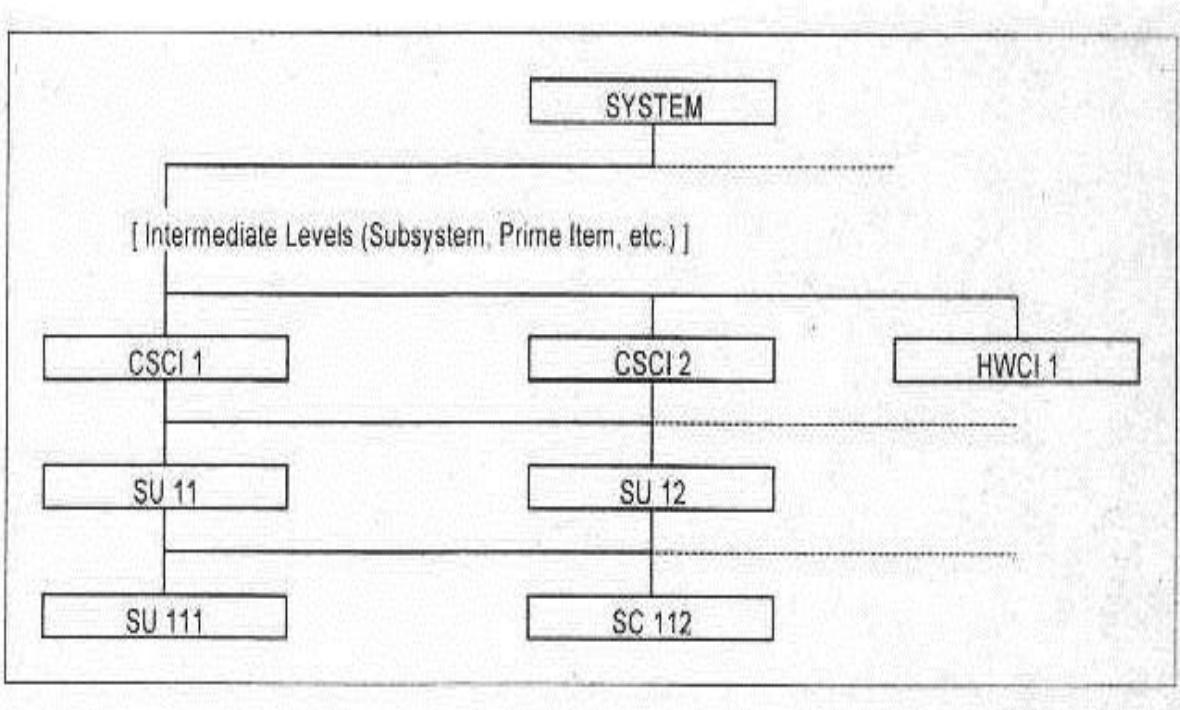
- Utilizează metode matematice
- Se bazează pe o specificație formală dezvoltată prin metoda rafinărilor succesive (Stepwise Rafinament)
- Abordarea formală presupune verificarea automată a corectitudinii specificației formale și generare automată de cod. Nu sunt necesare teste
- Utilă pentru sisteme non-interactive, dar necesită experți calificați

## 5.7 IEEE/EIA 12207 „Standardul pentru Tehnologia Informației - Procese ale ciclului de viață al software-ului”

- IEEE/EIA 12207 „Standardul pentru Tehnologia Informației - Procese ale ciclului de viață al software-ului” este un proces software generic utilizat ca și cadru pentru multe sisteme aflate în curs de dezvoltare sau suportate
- IEEE/EIA 12207 definește un **set de activități de dezvoltare recomandate și alternative de documentare** pentru sistemele intensive de software.
- Acest standard este **compatibil** cu o serie de metode diferite de dezvoltare software, inclusiv modelul cascadă.
- IEEE/EIA 12207 definește **o ierarhie software standard** și **o terminologie asociată** (utilizată în mod obișnuit pentru software-ul DOD complex și alte sisteme MIS complexe)

### 5.7.1 Ierarhia software standard

- În general, un **Sistem** (de exemplu, aeronava de luptă F-22) este mai întâi împărțit în diferite **Subsisteme** (de exemplu, avionică) și, uneori, **elemente principale** și **elemente critice** (de exemplu, radar de atac) (Fig.5.7.1.a.).



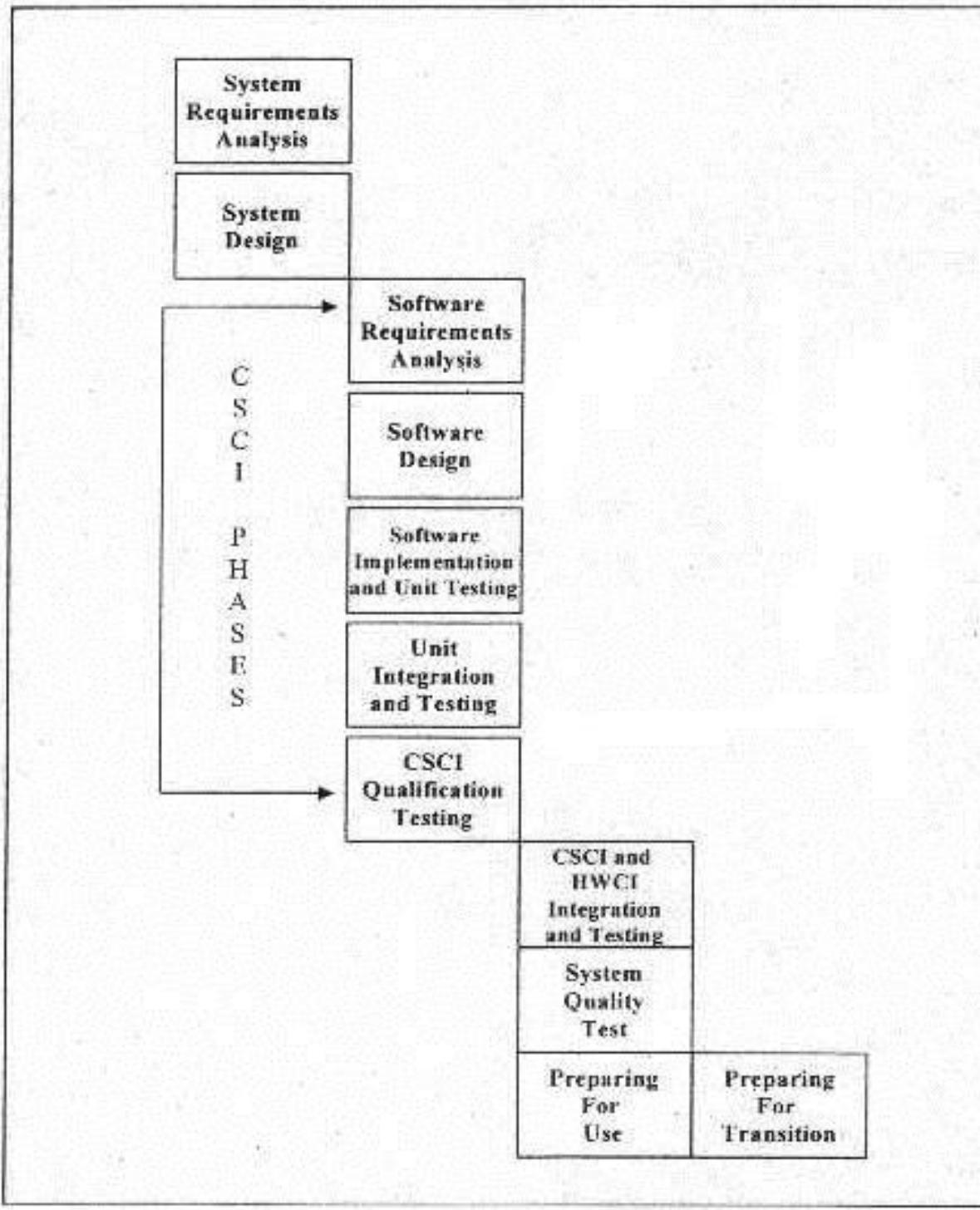
**Fig.5.7.1.a.** IEEE/EIA 12207: Ierarhia Software Standard

- Aceste **subsisteme** sau **elemente** sunt împărțite în continuare în:
  - (1) **Elemente de configurare hardware (HWCI)**
  - (2) **Elemente de configurare software pentru computer (CSCI)**
- Un **CSCI** (Computer SW Configuration Item) este definit ca o agregare de software care satisface o funcție finală comună.
  - Când **CSCI**-urile sunt mari (de exemplu, depășesc 100.000 de LOC), ele sunt din nou împărțite în niveluri mai gestionabile, numite **Unități software (SU)**.
  - **SU**-urile de cel mai coborât nivel conțin în general între 100 și 200 de linii de cod (LOC).
  - Structura și numărul de niveluri SU depind de natura și complexitatea CSCI în particular.
- Structura **ierarhiei software standard** este, de asemenea, un exemplu de **WBS (Work Breakdown Structure)** orientat spre produs.
  - Un **WBS** este o tehnică de management folosită pentru a subdiviza un sistem total în elementele sale componente.
  - **WBS**-urile sunt, în general, arbori genealogici orientați spre produs, compusi din hardware, software, servicii și alte sarcini de lucru.

- Un **WBS** definește produsul (produsele) care urmează să fie dezvoltat și leagă elementele de lucru între ele și produsul final.

### 5.7.2 Procesul de dezvoltare software IEEE/EIA 12207

- Procesul de dezvoltare software definit de standardul IEEE/EIA 12207 se bazează pe **Modelul fazelor**.
- Figura 5.7.2.a prezintă fazele ciclului de viață al software-ului asociate cu **Modelul Cascadă**.
- (1) În timpul *primelor două faze* (**Faza de analiză a cerințelor sistemului și Faza de proiectare a sistemului**), cerințele la nivel de sistem sunt împărțite în cerințe de nivel **CSCI** și **HWCI**.
- (2) Fiecare **CSCI** este apoi dezvoltat folosind un proces bazat pe **Ciclul de viață pentru CSCI** similar cu cel prezentat în figură (fazele CSCI) și care presupune:
  - (2.1) **Faza de analiză a cerințelor software**, cerințele specifice CSCI sunt definite în detaliu.
  - (2.2) **Faza de proiectare a software-ului**, cerințele software sunt rafinate în continuare la nivelul **SU** și împărțite în module în care sunt definite funcțiile, intrările, ieșirile și constrângerile.
  - (2.3) Odată ce software-ul este complet proiectat, acesta poate fi **codificat**.
    - Conform lui Cheadle, cerințele CSCI și fazele de proiectare pot reprezenta 60% din întregul efort de dezvoltare CSCI, în timp ce codificarea poate reprezenta doar 20%
  - (2.4) **Faza de implementare a software-ului și de testarea unităților SW**: scrierea codului sursă (de exemplu, instrucțiuni de limbaj C++) pentru fiecare SU și testarea fiecărui SU;
  - (2.5) **Integrarea și testarea**, agregarea de SU;
  - (2.6) **Faza de testare a calificării**: efectuarea de teste specifice pe CSCI pentru a asigura că toate cerințele sunt îndeplinite cu succes.
- (3) **Faza de integrare și testare CSCI și HWCI**: după ce CSCI-urile individuale sunt testate, aggregatele de HWCI și CSCI-uri sunt integrate și testate.
- (4) **Faza de testare a calității sistemului**: testarea de calificare este efectuată pe întregul sistem pentru a se asigura că cerințele la nivel de sistem sunt îndeplinite.
- (5) **Pregătirea pentru utilizare și pregătirea pentru fazele de tranziție**: după ce toate testele sunt finalizate, software-ul este transferat agenților de utilizare și suport.
- În timpul fiecărei etape de dezvoltare a software-ului, pot avea loc o serie de alte activități cheie, cum ar fi:
  - (1) **Management de proiect software**,
  - (2) **Gestionarea configurației software**,
  - (3) **Asigurarea calității software-ului**.



**Fig.5.7.2.a.** Standardul IEEE/EIA 12207: Fazele modelului de dezvoltare software cascadă

- Fiecare dintre activitățile desfășurate pentru fiecare disciplină, în fiecare fază, poate fi organizată într-o activitate **WBS** pentru fiecare **CSCI**.
  - Acest **WBS** poate fi utilizat pentru produsul WBS ca bază pentru raportarea managementului și urmărirea pentru CSCI.
- Fazele ciclului de viață al software-ului (adică, CSCI) prezentate **nu** trebuie să apară **secvențial**, așa cum ar putea sugera ilustrația.
- Multe practici moderne de dezvoltare pot duce la o ordine diferită a activităților sau chiar la o combinație (sau suprapunere) de activități.

- Astă explică apariția unei multitudini de metodologii alternative de dezvoltare software.
- Tipul de metodologie de dezvoltare software utilizată are în general un impact semnificativ asupra **costurilor totale** ale ciclului de viață al dezvoltării și întreținerii software-ului.

## 6. Procesul de management

- Aparține Proceselor Organizaționale
- Este simplu în esență: planifică și controlează
- De fapt poate deveni de o mare complexitate din cauza multiplelor direcții de activitate care trebuie tratate.
- Activitățile **Procesului de management** sunt:

### 6.1 Activități

- (1) **Inițierea și stabilirea domeniului de aplicație** – această activitate presupune:
  - (1.1) **Identificarea** procesului de tratat
  - (1.2) **Stabilirea cerințelor** procesului;
  - (1.3) **Stabilirea fezabilității** procesului (analiza existenței și adevării resurselor umane, materiale, tehnologice, de mediu și de timp în conformitate cu sfera proiectului)
  - (1.4) În funcție de necesități, este permisă **adaptarea cerințelor** procesului la **resursele** disponibile, cu acordul prealabil al tuturor părților implicate.
  - (1.5) **Stabilirea și analiza riscurilor** proiectului
- (2) **Planificare** – presupune realizarea **planurilor de execuție** a sarcinilor procesului său.
  - În principiu, aceste **planuri** se referă la:
    - (2.1) **Descrierea** activităților
    - (2.2) **Descrierea** sarcinilor (taskurilor) asociate
    - (2.3) **Identificarea** produselor ce urmează a fi livrate
  - Un **plan** include:
    - (a) **Restricții temporale** impuse de mediul procesului (termene limită de execuție);
    - (b) **Estimarea efortului**;
    - (c) **Identificarea** resurselor adevărate;
    - (d) **Atribuirea** sarcinilor și stabilirea **responsabilităților**;
    - (e) **Stabilirea** planificărilor și a **termenelor limită** pentru toate activitățile;
    - (f) **Identificarea** și evaluarea **riscurilor** asociate fiecărei sarcini și stabilirea măsurilor de contracarare adevărate;
    - (g) **Pregătirea** mediului și a **infrastructurii**;
    - (h) **Stabilirea** măsurilor de control al calității care trebuie utilizate în cadrul procesului;
    - (i) **Stabilirea** costurilor asociate executării procesului.
- (3) **Execuție și control** – presupune următoarele sarcini:
  - (3.1) **Inițierea implementării** planului

- (3.2) **Monitorizarea execuției procesului și raportarea progresului** către client, precum și către conducerea superioară
- (3.3) **Investigarea**, analiza și identificarea **soluției** pentru orice **problemă descoperită**.
- (3.4) **Supravegherea** aplicării **soluției** și **înregistrarea** remedierii problemei
- (3.5) Dacă rezolvarea problemei presupune **modificarea planului**, managerul este responsabil cu asigurarea că **impactul este determinat, controlat și monitorizat**.
- (3.6) **Monitorizarea nivelurilor de risc** asociate fiecărei sarcini și dispozitiv în timp real a măsurilor necesare
- (4) **Analiză și evaluare** – această activitate presupune:
  - (4.1) **Evaluarea** (testarea) **produselor SW** și a **sarcinilor (taskurilor)** finalizate pentru conformitatea cu cerințele
  - (4.2) **Analiza rezultatelor evaluării**
- (5) **Finalizare** – presupune că managerul de proiect:
  - (5.1) **Se asigură că procesul este încheiat** și că toate obiectivele sale au fost atinse (toate criteriile planificate au fost îndeplinite)
  - (5.2) **Verifică** completitatea **documentelor**, în principal înregistrările referitoare la produsele SW dezvoltate și la activitățile și sarcinile executate
  - (5.3) **Arhivează rezultatele și documentele** pe un suport adecvat și stabil

## 7. Leadership (conducere) și management

### 7.1 Definirea termenilor generali

- Dicționarul **Wordweb Thesaurus Dictionary/ Merriam-Webster Dictionary** precizează:
  - **Leadership (conducere)** înseamnă:
    - Activitatea de conducere
    - O poziție de lider al unui grup, organizație etc.
    - Corpul de oameni care conduc un grup
    - Statutul de lider
    - Puterea sau capacitatea de a conduce alți oameni
  - **Management** înseamnă:
    - Actul sau arta de a gestiona ceva: conducerea sau supravegherea a ceva (ca afacere)
    - Acțiunea sau abilitatea de a controla și de a lua decizii cu privire la o afacere, un departament, o echipă sportivă etc.
    - Utilizarea judicioasă a mijloacelor pentru realizarea unui scop
    - Organismul colectiv al celor care conduc sau conduc o întreprindere
    - Cei care se ocupă de conducerea unei afaceri
- **Peter Drucker și Warren Bennis**:
  - **Management** înseamnă a face **lucrurile corect** (așa cum trebuie).
  - **Leadership** înseamnă a face **lucrurile corecte** (cele care trebuie).
- **Stephen R.Covey**:
  - **Leadership**-ul se ocupă de linia de avangardă: Care sunt lucrurile de realizat?
  - **Managementul** se concentrează de jos în sus: Cum pot realiza cel mai bine anumite lucruri?

- **Managementul** este eficiență în urcarea pe scara succesului;
- **Conducerea** determină dacă scara se sprijină de peretele corect.
- **Conducerea puternică proactivă** trebuie să monitorizeze constant schimbările de mediu, în special obiceiurile, tendințele și motivele de cumpărare ale clientilor, și să ofere forță necesară pentru a organiza resursele în direcția corectă.
- **Managerii** identifică instrumentele, scriu manuale de politici și proceduri, organizează programe de formare, aduc tehnologii îmbunătățite, stabilesc programe de lucru și programe de compensare.
- Niciun succes în **management** nu poate compensa eșecul în **conducere**.

## 7.2 Definirea termenilor specifici

- **Managementul unui Proiect SW:** cuprinde toate *Activitățile și Sarcinile (Taskurile) de planificare și control* a Activităților Membrilor Personalului, astfel încât poate fi atinsă o țintă pe care personalul nu o poate atinge singur.  

(Balzert, H. Lehrbuch der Software-Technik, Spektrum, 1998)
- **Inginerie software:** Implementare orientată către țintă (de exemplu, în termeni de costuri, timp, calitate) și utilizarea sistematică a metodelor, conceptelor, notațiilor și instrumentelor pentru dezvoltarea și aplicarea sistemelor software complexe de tip inginer.  

(Balzert, H. Lehrbuch der Software-Technik, Spektrum, 1998)
- **Inginerie software:** Subiect tehnic, Studiază ca Disciplină de Inginerie
- **Management de proiect:** Organizarea non-tehnică a procesului de dezvoltare
  - Presupune interfețe adecvate pentru
    - Modelul de execuție a proiectului/modele de proces
    - Managementul calității
    - Managementul configurației
  - Include și alte sarcini (taskuri) aferente execuției unui proiect, cum ar fi
    - Pregătire (structură și personal)
    - Planificare
    - Supraveghere
    - Control
  - Dincolo scopul unui proiect individual, managementul de proiect include următoarele sarcini:
    - Finalizarea proiectului
    - Îmbunătățirea proceselor
    - Managementul resursei umane
- **Sarcina unui Manager de Proiect** este accea de a se asigura că **proiectul software** este realizat în conformitate cu **bugetul și timpul stabilit** și să livreze **software-ul** care contribuie la atingerea **țintelor economice stabilite**.

(Sommerville, I. *Software Engineering*, Pearson, 2001)

## Exercițiul #1

- 1) Descrieți principalele caracteristici ale activității de producție de software.
- 2) Care sunt rațiunile pentru a organiza dezvoltarea unui proiect software?

- 3) Ce este standardul ISO/CEI 12207:1995? Definiți următorii termeni: Procese, Activități, Sarcini (Taskuri).
- 4) Descrieți procesele primare, de sprijin și organizaționale.
- 5) Care sunt activitățile procesului de dezvoltare?
- 6) Ce fel de cicluri de viață pentru dezvoltarea proiectelor software cunoașteți?
- 7) Care sunt activitățile procesului de management?
- 8) Care sunt diferențele dintre termenii leadership (conducere) și management.
- 9) Definiți următoarele concepte: Inginerie software și Management de proiect software? Care este diferența dintre ele?

## **Capitolul 2. TEHNOLOGII PENTRU DEZVOLTAREA PRODUSELOR SOFTWARE**

- 1. Tehnologia MicroSoft pentru dezvoltarea produselor SW**
  - 1.1 Cadrul general
    - 1.1.1 Prezentare MS
    - 1.1.2 Filosofia Microsoft
    - 1.1.3 Obiectivele MS
  - 1.2 MS Paradigma: Sincronizări frecvente și stabilizări periodice
    - 1.2.1 Paradigma de bază a abordării MS
    - 1.2.2 Tendințe în industria SW
    - 1.2.3 Abordarea de dezvoltare clasică în cascadă (secvențială).
    - 1.2.4 Facilități dorite pentru procesul de dezvoltare
  - 1.3 Strategii și principii
    - 1.3.1 Prezentare generală a abordării dezvoltării sincronizate și stabilizate
    - 1.3.2 Definirea produsului și organizarea procesului de dezvoltare. Prima strategie MS
    - 1.3.3 Dezvoltarea și livrarea produselor. A doua strategie pentru SM
  - 1.4 Tehnologia MS
    - 1.4.1 Obiectiv
    - 1.4.2 Reguli
    - 1.4.3 Modul de lucru
    - 1.4.4 Particularități ale echipei MS
  - 1.5 Concluzii
    - 1.5.1 Inovații MS
    - 1.5.2 Strategii competitive ale statelor membre
    - 1.5.3 Punctele slabe ale abordării de dezvoltare Microsoft.
    - 1.5.4 Avantajele tehnologiei MS

### **2.Tehnologia ORACLE pentru dezvoltarea produselor SW**

- 2.1 Probleme actuale ale industriei IT
  - 2.1.1 Probleme în industria IT
  - 2.1.2 Cauze evidențiate de experiența ORACLE
  - 2.1.3 Concluzie
- 2.2 ORACLE set de metodologii de dezvoltare
  - 2.2.1 Idee fundamentală
  - 2.2.2 ORACLE set de metodologii de dezvoltare
- 2.3 Responsabilitățile managerului de proiect
  - 2.3.1 Responsabilități
  - 2.3.2 Sarcini majore
- 2.4 Etapele proiectului ORACLE
  - 2.4.1 Planificare
  - 2.4.2 Execuție
  - 2.4.3 Finalizare
- 2.5 Procesele proiectului ORACLE
  - 2.5.1 Definiția procesului. Tipuri de procese
  - 2.5.2 Procesele în tehnologia ORACLE
  - 2.5.3 Observații generale
- 2.6 Concluzii

### **3. Procesul de dezvoltare Rational**

- 3.1 Calea Rational
- 3.2 Ciclul de viață general al software-ului
  - 3.2.1 Cele două perspective
  - 3.2.2 Cicluri și faze

- 3.2.3 Iterații
- 3.2.4. Reconcilierea celor două perspective
- 3.2.5 Discriminatori
- 3.2.6 Efort și program
- 3.3 Fazele procesului rațional
  - 3.3.1 Faza de început
  - 3.3.2 Faza de elaborare
  - 3.3.3 Faza de construcție
  - 3.3.4 Faza de tranzitie
  - 3.3.5 Cicluri de evoluție
- 3.4 Activități în Procesul Rațional
- 3.5 Artefacte ale ciclului de viață
  - 3.5.1 Artefacte de management
  - 3.5.2 Artefacte tehnice
  - 3.5.3 Cerințe
- 3.6 Exemple de proces rațional
  - 3.6.1 Proces rațional pentru dezvoltarea software contractuală mare
  - 3.6.2 Proces rațional pentru un produs software comercial mic
- 3.7 Concluzie
- 3.8 Glosar
- 3.9 Acronime

## **Exercițiu #2**

## Capitolul 2. TEHNOLOGII PENTRU DEZVOLTAREA PRODUSELOR SOFTWARE

### 1 Tehnologia MicroSoft pentru dezvoltarea produselor software

#### 1.1 Cadrul general

##### 1.1.1 Prezentarea Microsoft

- Cel mai mare producător mondial de software pentru PC
- Probabil că a abordat mai multe proiecte software pentru PC decât orice altă companie din industrie.
- Complexitatea unora dintre produsele sale, rivalizează cu cea a multor sisteme pentru calculatoare mainframe și sisteme de telecomunicații.

##### 1.1.2 Filosofia Microsoft

- 1) Să-și cultive rădăcinile ca o companie antreprenorială extrem de flexibilă
- 2) Să nu adopte prea multe dintre practicile de inginerie software structurată promovate în mod obișnuit de astfel de organizații precum:
  - Institutul de Inginerie Software (SEI)
  - Organizația Internațională de Standardizare (ISO)
- 3) Pentru a „extinde” un stil de dezvoltare a produsului format din echipe mici (unii ar putea spune hacker)

##### 1.1.3 Obiectivele MS

- 1) Să determine mai multe echipe paralele mici (cu câte trei până la opt dezvoltatori fiecare) sau programatori individuali să lucreze împreună ca o singură echipă relativ mare pentru a construi produse mari relativ rapid
- 2) Să permită programatorilor individuali și echipei libertatea de a-și evoluă design-urile și de a funcționa aproape autonom.
- 3) Echipele paralele dezvoltă treptat funcții și produse finite, introducând ocazional noi concepte și tehnologii.
- 4) Deoarece dezvoltatorii sunt liberi să inoveze pe măsură ce avansează, ei trebuie să-și sincronizeze modificările frecvent, astfel încât componentele produsului să funcționeze împreună.
- 5) **Microsoft:** utilizează diverse tehnici și le combină într-o abordare generală care echilibrează flexibilitatea și structura în dezvoltarea de produse software.
- 6) Abordarea generală se bazează pe o paradigmă fundamentală.

### 1.2 Paradigma MicroSoft: Sincronizări frecvente și stabilizări periodice

#### 1.2.1 Paradigma de bază a abordării MS: „sincronizare și stabilizare”

- **Ideea** pe care se bazează paradigma:
  - **Sincronizarea continuă** a ceea dezvoltă oamenii ca indivizi și ca membri ai echipelor paralele.
  - **Stabilizarea periodică** a produsului trătat, pe măsură ce proiectul se derulează și nu odată la sfârșitul proiectului.
- **Termeni utilizati:**
  - 1) Dezvoltatorii Microsoft se referă la tehniciile lor în diferite maniere utilizând termeni ca „milestone” (“jalon”), „daily build” („constructie de

*zi”), „nightly build” („construcție de noapte”) sau „zero defect” („zero defecte”).*

- 2) Termenul **build** se referă la acțiunea de a pune împreună piese parțial finalizate sau finite ale unui produs software în timpul procesului de dezvoltare, pentru a vedea ce funcții funcționează și ce probleme există. Acest lucru se realizează de obicei prin recompilarea completă a codului sursă și prin executarea de teste de regresie automate.

- **Cerințele impuse metodologiei de dezvoltare:**

- 1) Să dezvolte produse noi, foarte complexe
- 2) Să folosească echipe cât mai mici
- 3) Să stimuleze inventivitatea, creativitatea și inovația în timpul dezvoltării produsului.
- 4) Necesitatea de a crea componente care sunt interdependente. Astfel de componente sunt dificil de definit cu acuratețe în primele etape ale ciclului de dezvoltare
- 5) Încurajarea modalităților care structurează și coordonează ceea ce dezvoltă membrii individuali
- 6) Încurajarea dezvoltării incrementale
- 7) Tendința de a acorda dezvoltatorilor suficientă flexibilitate pentru a fi creativi și pentru a dezvolta detaliile produsului în etape.
- 8) Abordarea dezvoltării trebuie să beneficieze, de asemenea, de un mecanism care să permită dezvoltatorilor să testeze produsul cu clientii și să își rafineze design-urile în timpul procesului de dezvoltare.

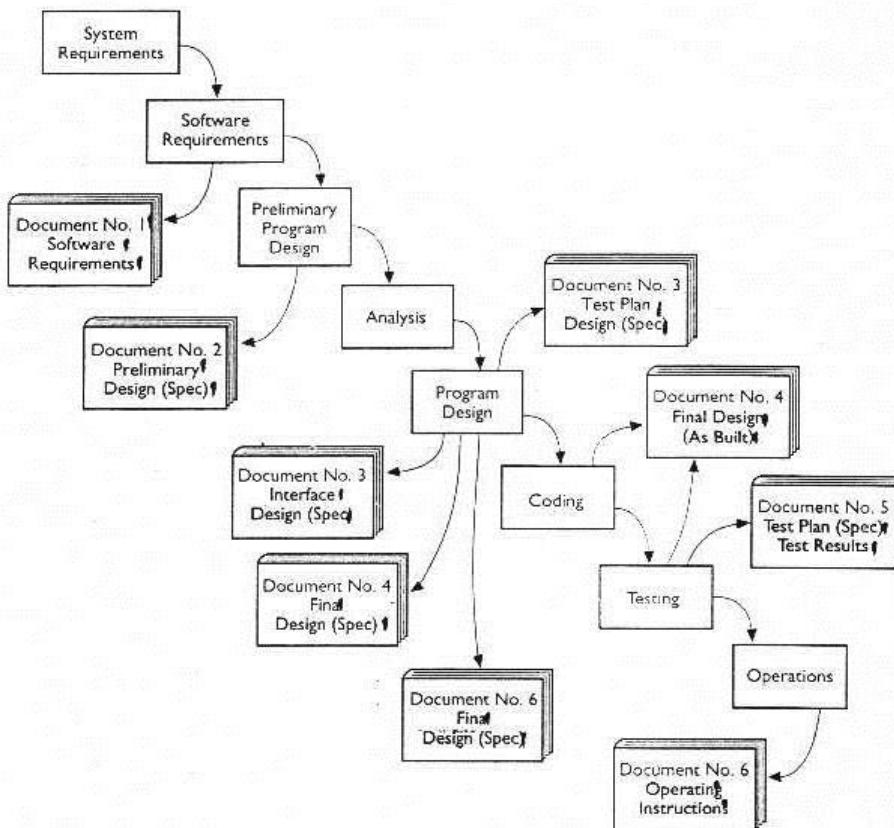
### 1.2.2 Tendințe în industria software actuală

- Multe companii folosesc acum noi abordări:
  - **Prototipare**
  - **Cicluri multiple de activități concurente** de proiectare, construcție și testare pentru a controla iterările,
  - **Schimbări incrementale** în dezvoltarea produsului.
- În comunitatea software, cercetătorii și managerii vorbesc despre:
  - **Îmbunătățirea iterativă**,
  - **Modelul în spirală** pentru iterare între fazele dezvoltării proiectului,
  - **Dezvoltarea concurrentă** a mai multor faze și activități.
- Cu toate acestea, multe companii au întârziat să adopte oficial aceste recomandări.
- **Ideea de bază** împărtășită de aceste abordări este aceea că:
  - Nevoile utilizatorilor pentru multe tipuri de software sunt greu de înțeles.
  - Schimbările în tehnologiile hardware și software sunt atât de continue și rapide.
  - Nu este înțelept să încerci să proiectezi un sistem software complet în avans.
- În schimb, proiectele ar putea avea nevoie să se repete în timp ce gestionează concomitent multe cicluri de proiectare, construcție și testare pentru a avansa către finalizarea unui produs.

### 1.2.3 Abordarea de dezvoltare clasică în cascadă (secvențială)

- O elaborare mai detaliată a ciclului de viață cascadă apare în fig.1.2.3.a.
- **Filosofia** ciclului de dezvoltare cascadă se bazează pe

- (1) Proiectele urmăresc „înghețarea” specificației de produsului;
- (2) Se creează un design;
- (3) Se construiesc componente;
- (4) Se asamblează componente la sfârșitul proiectului într-o fază mare de integrare și testare.
- Această abordare a dezvoltării software a fost comună în anii 1970 și 1980
- De asemenea, rămâne un model de bază pentru planificarea proiectelor în multe industrii.



**Fig.1.2.3.a.** Modelul procesului de dezvoltare cascadă

- **Dezavantajele** ciclului cascadă:
  - Specificațiile și design-urile nu pot fi modificate;
  - Nu este posibil să se obțină feedback de la clienți;
  - Nu este posibilă testarea continuă a componentelor pe măsură ce produsele evoluează.

#### 1.2.4 Facilități dorite pentru procesul de dezvoltare

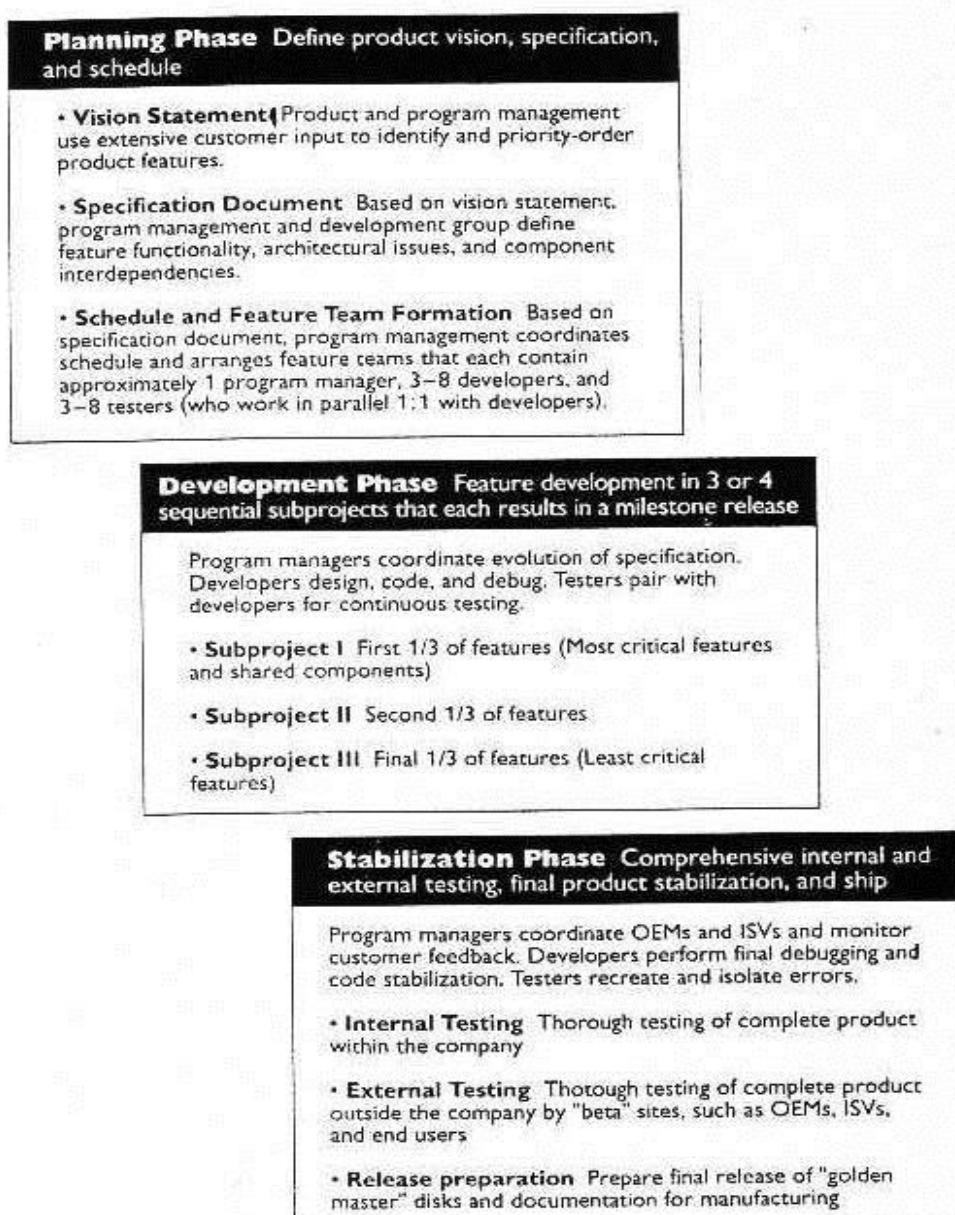
- (1) Iterație între proiectare, componente de construcție și testare.
- (2) Suprapunerea fazelor de dezvoltare.
- (3) Interacțiuni multiple cu clienții în timpul dezvoltării.
- (4) Facilități pentru expedierea versiunilor preliminare ale produselor.
- (5) Adăugarea progresivă de caracteristici sau funcționalități de-a lungul timpului în diferite versiuni de produse.
- (6) Integrarea frecventă a pieselor produsului pentru a determina ce funcționează și ce nu, fără a aștepta până la finalul proiectului.

## 1.3 STRATEGII ȘI PRINCIPII

- Pentru a implementa paradaigma sa fundamentală „*sincronizare și stabilizare*”, MS a dezvoltat:
  - (1) O abordare specifică de dezvoltare – **Metoda de dezvoltare sincronizează și stabilizează**
  - (2) Două strategii vizând:
    - (2.1) **Definirea produsului și organizarea procesului de dezvoltare**
    - (2.2) **Dezvoltarea și livrarea produselor**
- Fiecare strategie constă din seturi de principii care sunt esențiale pentru realizarea stilului de sincronizare și stabilizare a dezvoltării produsului.

### 1.3.1 Prezentare generală a Metodei de dezvoltare sincronizează și stabilizează

- Procesul de dezvoltare a produsului se bazează pe **modelul fazelor** și constă în 3 faze (fig.1.3.1.a).



### **Fig.1.3.1.a. Prezentare generală a Metodei de dezvoltare sincronizează și stabilizează**

#### **(1) Faza de planificare**

(1.1) Procesul de dezvoltare a produsului începe prin crearea unui document denumit „**Viziune**” (Vision statement).

- Documentul **Viziune**
  - Definește obiectivele noului produs
  - Ordenează activitățile utilizator care trebuie să fie suportate de caracteristicile produsului
- Managerii de produs (specialiști în marketing) se ocupă de acest document în timp ce consultă managerii de program specializați în redactarea specificațiilor funcționale ale produsului
- Gestionarea produselor și a programelor utilizează o contribuție extinsă a clientilor pentru a identifica și a ordona și prioritiza caracteristicilor produsului

(1.2) Pe baza **Viziunii**, grupul de management și dezvoltare a programului elaborează **Documentul de specificare** (Specificația)

- Documentul de specificare definește pentru fiecare caracteristică:
  - (1) Descrierea funcționalității caracteristicii
  - (2) Problemele arhitecturale specifice
  - (3) Interdependențele cu alte componente
- Evidențiază caracteristicile produsului la un nivel de detaliu suficient pentru a organiza planificările și alocarea personalului
- Documentul de specificații initiale nu încearcă să acopere toate detaliile fiecărei caracteristici sau să blocheze proiectul în setul original de caracteristici
- În timpul dezvoltării produsului, membrii echipei revizuiesc setul de caracteristici și detaliile caracteristicilor pe măsură ce învață mai multe despre ceea ce ar trebui să fie în produs.
- Experiența la Microsoft sugerează că setul de caracteristici dintr-un document cu specificații se poate modifica cu 30% sau mai mult pe parcursul dezvoltării

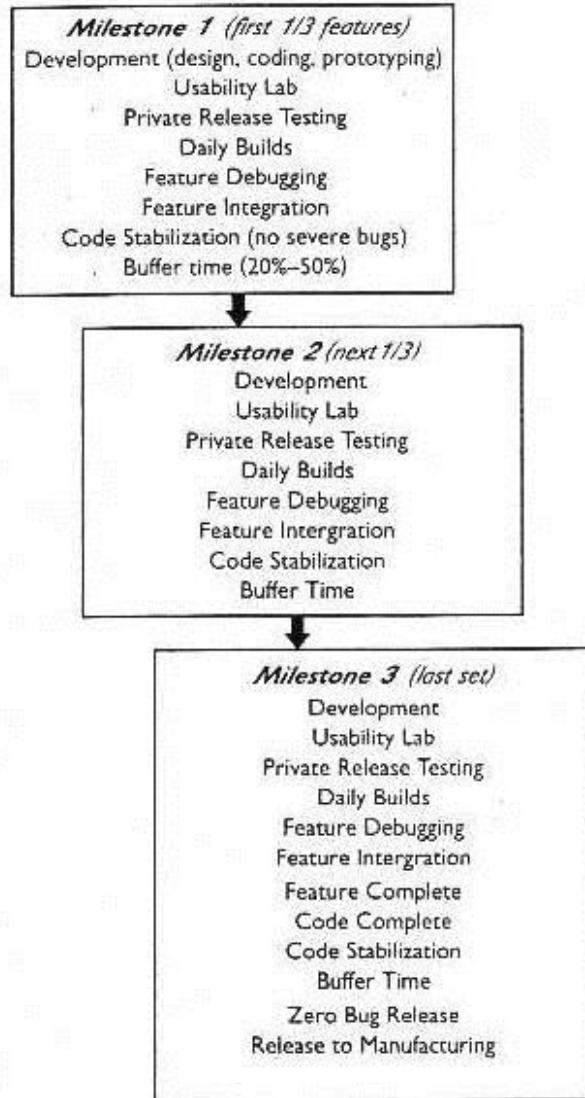
(1.3) Planificarea și formarea echipei de caracteristici

- Pe baza documentului de specificare, managementul programului coordonează planificările și organizează echipele de caracteristici.
- Fiecare echipă conține aproximativ:
  - 1 manager de program
  - 3—8 dezvoltatori
  - 3—8 testeri care lucrează în paralel 1:1 cu dezvoltatorii

#### **(2) Faza de dezvoltare**

- Managerii de proiect împart apoi produsul și proiectul în părți care pot fi atribuite caracteristicilor și echipelor de caracteristici mici

- Programul proiectului este împărțit în trei sau patru puncte de referință (subproiecte succesive) reprezentând puncte de finalizare pentru porțiuni majore ale produsului (fig.1.3.1.b).
  - **Subproiectul I** (Milestone 1) - Prima 1/3 din caracteristici (Cele mai critice caracteristici și componentele partajate)
  - **Subproiectul II** (Milestone 2) - A doua 1/3 din caracteristici
  - **Subproiectul III** (Milestone 3) - Ultima 1/3 din caracteristici (Cel mai puțin critice)



**Fig. 1.3.1.b.** Repere (milestone-uri) ale metodei sincronizează și stabilizează.

- Managerii de program coordonează evoluția specificațiilor
- Dezvoltatorii proiectează, codifică și depanează
- Testerii se asociază cu dezvoltatorii pentru testare continuă.
- Toate echipele de caracteristici trec prin un ciclu complet de dezvoltare, integrare de caracteristici, testare și remediere a problemelor în fiecare subproiect de reper
- Pe parcursul întregului proiect, echipele de caracteristici își sincronizează munca prin construirea produsului și prin găsirea și remedierea erorilor zilnic și săptămânal.

- La sfârșitul unui sub-proiect de reper, dezvoltatorii remediază aproape toate erorile detectate în produsul aflat în evoluție. Aceste corecții de eroare:
  - Stabilizează produsul
  - Permite echipei să înțeleagă clar care părți ale produsului au fost finalizate.
- Echipa de dezvoltare poate trece apoi la următorul reper și eventual, la data livrării.

### (3) Faza de stabilizare

- Presupune o serie de activități specifice cum ar fi:
- **Testare internă cuprinzătoare** – constă în testarea produsului complet în cadrul companiei
- **Testare externă** – constă în testarea produsului complet în afara companiei de către site-uri „beta”, cum ar fi **Original Equipment Manufacturer's (OEM)**, **Institut of Software Validation (ISV)** și utilizatorii finali.
- Managerii de program coordonează observațiile OEM și ISV și monitorizează feedback-ul clienților
- Dezvoltatorii efectuează **depanarea finală și stabilizarea codului**
- Testerii recreează și izolează erorile
- Stabilizarea **produsului final**
- Pregătirea lansării - Pregătirea **versiunii finale** a discurilor „golden master” și documentația de dezvoltare
- Expedierea produsului.

#### 1.3.2 Prima strategie MS: Definirea produsului și organizarea procesului de dezvoltare

- Pentru a defini produsele și a organiza procesul de dezvoltare, principalele grupuri de produse Microsoft urmează prima strategie care poate fi descrisă ca „concentrează creativitatea prin evoluția caracteristicilor și „repararea” resurselor”.
- Echipele implementează această strategie prin **cinci principii specifice**:
  - (1) **Divizarea proiectelor mari în mai multe cicluri de dezvoltare** (subproiecte), cu repere specifice și cu un tampon de timp alocat (20%—50% din timpul total al proiectului). Microsoft rezolvă aceste probleme prin:
    - Structurarea proiectelor în subproiecte secvențiale care conțin caracteristici ordonate în ordinea priorităților.
    - Stabilirea unui tampon de timp pentru fiecare subproiect, care oferă oamenilor timp să răspundă la schimbări și la dificultăți sau întârzieri neașteptate.
  - (2) **Utilizarea unei viziuni asupra produsului secondată de o creionarea specificației caracteristicilor** pentru a ghida proiectele, mai degrabă decât elaborarea de proiecte detaliate și specificații complete ale produsului înainte de codificare
    - Echipele realizează că nu pot determina în avans tot ce trebuie să facă dezvoltatorii pentru a construi un produs bun.
    - Această abordare lasă dezvoltatorilor și managerilor de programe spațiu pentru a inova sau a se adapta la oportunități și amenințări competitive modificate sau neprevăzute.

- În special pentru aplicații produse:
  - Echipete de dezvoltare încearcă să vină cu caracteristici care să se mapeze direct la activitățile pe care le fac clienții obișnuiați
  - Echipetele trebuie să efectueze observații și testari continue cu utilizatorii în timpul dezvoltării.

(3) Selecția și prioritizarea caracteristicilor se bazează pe activitățile și datele utilizatorilor.

(4) Dezvoltarea unei arhitecturi modulare și orizontale, care să reflecte structura produsului în structura proiectului.

- Majoritatea modelelor de produse au **arhitecturi modulare** care permit echipele să adauge sau să combine incremental funcții într-un mod simplu și previzibil
- Managerii le permit **membrilor echipei** să-și stabilească **propriile planificări**, dar numai după ce dezvoltatorii au analizat sarcinile în detaliu (de exemplu, bucăți de jumătate de zi până la trei zile) și li s-a cerut să se angajeze personal în respectarea planificărilor stabilite de ei.
- Managerii „**fixează**” **resursele proiectului** prin **limitarea numărului de oameni** pe care îi alocă fiecărui proiect.
- Managerii încearcă, de asemenea, să limiteze **timpul petrecut pentru proiecte**, (în special pentru aplicații precum Office și produsele multimedia), astfel încât echipele să poată șterge funcții dacă rămân prea în urmă.
- Cu toate acestea, reducerea funcțiilor pentru a economisi timpul de programare nu este întotdeauna posibilă cu proiecte de sisteme de operare în care:
  - Fiabilitatea este mai importantă decât caracteristicile produsului
  - Multe caracteristici sunt strâns legate de celelalte și nu pot fi anulate cu ușurință.

(5) Control prin angajamente individuale pentru taskurile mici și resursele „fixe” ale proiectului.

### 1.3.3 A doua strategie MS: Dezvoltarea și livrarea produselor

- Pentru a gestiona procesul de dezvoltare și livrare a produselor, Microsoft urmează a două strategie care poate fi descrisă ca „**totul se realizează în paralel cu sincronizări frecvente**”.
- Echipele implementează această strategie urmând un alt **set de cinci principii**:
  - (1) Echipele lucrează în paralel, dar „**sincronizarea**” și **depanarea** se realizează zilnic.
  - (2) Există întotdeauna un produs pe care poate fi livrat cu versiuni pentru fiecare **platformă** și **piață** majoră, avute în vedere
  - (3) Se vorbiți o „**limbă comună**” pe **un singur site de dezvoltare**.
  - (4) Produsul se **testează continuu** pe măsură ce este construit
  - (5) Se utilizează datele metrice specifice pentru a stabili finalizarea etapei de referință și lansarea produsului.

- **Obs:** Aceste principii aduc o disciplină considerabilă procesului de dezvoltare, fără a fi nevoie ca fiecare dezvoltator să fie controlat în fiecare moment al zilei.

## 1.4 Tehnologia Microsoft

### 1.4.1 Obiective

- Microsoft încearcă să permită mai multor **echipe mici și indivizilor** suficientă libertate pentru **a lucra în paralel**, dar totuși să funcționeze ca **o singură echipă mare**, astfel încât să poată construi **produse la scară largă** relativ **rapid și ieftin**.

### 1.4.2 Reguli

- Echipele aderă la **câteva reguli rigide** care impun un grad ridicat de coordonare și comunicare.
  - (1) Indiferent de ziua în care dezvoltatorul decide să „**încarce**” („check in”) sau **să-și introducă bucătile de cod** în baza de date de produse, trebuie să facă acest lucru **până la o anumită oră** (să zicem, 14:00 sau 17:00). Această regulă permite echipei de proiect:
    - Să asambleze componentele disponibile
    - Să recompileze complet codul sursă al produsului și să creeze un nou „build” al produsului care se dezvoltă până la sfârșitul zilei sau până a doua zi dimineață
    - Să înceapă imediat testarea și depanarea.
    - Această regulă este analogă cu a le spune copiilor că pot face tot ce vor ei toată ziua, dar trebuie să fie în pat la ora 21:00.
  - (2) În cazul în care dezvoltatorii constată că în urma încărcării **codul „distrugă”** („breaks”) build-ul împiedicând finalizarea recompilării, **ei trebuie să remedieze defectul imediat**.
    - Această regulă seamănă cu faimosul sistem de producție Toyota, în care muncitorii din fabrică sunt încurajați să oprească liniile de producție ori de câte ori observă un defect la mașina pe care o montează.

### 1.4.3 Modul de lucru

- **Procesul de construire zilnic** al Microsoft are mai mulți **pași**.
  - (1) În primul rând, pentru a dezvolta o caracteristică pentru un produs, dezvoltatorii **pot descărca** (check-out) copii private ale fișierelor codului sursă dintr-o **versiune centrală principală** (master) a codului sursă
  - (2) **Dezvoltatorii își implementează caracteristicile** prin modificarea **copiilor lor private** ale fișierelor de cod sursă
  - (3) Dezvoltatorii creează apoi **o versiune privată a produsului** (private build) care conține noua caracteristică încărcată și o testează
  - (4) Apoi dezvoltatorii **încarcă modificările** de la copiile lor private ale fișierelor codului sursă **în versiunea principală (master) a codului sursă** generând o **ramură** (branch)

(5) Procesul de check-in include și un **test de regresie automat** pentru a verifica dacă modificările aduse la fișierele codului sursă nu cauzează erori în altă parte a produsului.

(6) De obicei, dezvoltatorii își încarcă codul în copia principală de cel puțin două ori pe săptămână, dar îl pot încărca și zilnic.

(7) Indiferent de cât de des își încarcă dezvoltatorii individuali modificările aduse codului sursă, un **dezvoltator desemnat**, numit **project build master**, generează zilnic **o versiune completă a produsului** folosind versiunea principală a codului sursă.

(8) Generarea unei **versiuni** pentru un produs constă în executarea unei **secvențe automate de comenzi** numită „**script de compilare**” (build script).

- Această versiune zilnică creează o nouă **versiune internă a produsului** și include mai mulți pași de compilare a codul sursă.
- Procesul de construire traduce automat codul sursă pentru un produs într-unul sau mai multe fișiere executabile și poate crea diferite fișiere de bibliotecă, permitând utilizatorilor finali să personalizeze produsul.
- **Noua versiune internă a produsului** construită în fiecare zi este **versiunea zilnică** (daily build).

(9) **Versiunile zilnice** sunt generate pentru **fiecare platformă**, cum ar fi Windows și Macintosh, pentru **fiecare piață**, cum ar fi S.U.A și pentru **versiunile internaționale majore**.

(10) Echipele de produs **testează caracteristicile** pe măsură ce le construiesc, din mai multe perspective, inclusiv aducând clienți „de pe stradă” pentru a încerca prototipuri într-un laborator de utilizare Microsoft.

#### 1.4.4 Particularități ale echipelor Microsoft

- Aproape toate echipele Microsoft lucrează într-un **un singur site fizic**
- Utilizează **un limbaj comun de dezvoltare** (în principal C și C++),
- Utilizează **un stil comun de codare**
- Utilizează **instrumente de dezvoltare standardizate**.
  - Un **site comun** și **un limbaj și instrumente comune** ajută echipele să **comunice**, să **dezbată** idei de design și să **rezolve** probleme față în față.
- Echipele de proiect folosesc, de asemenea, **un set mic de metrii cantitative** pentru a ghida deciziile, cum ar fi când să avanseze într-un proiect și când să livreze un produs pe piață.
  - De exemplu, managerii urmăresc cu rigurozitate progresul versiunilor zilnice monitorizând câte **erori** (bug-uri) sunt recent deschise, rezolvate (cum ar fi prin eliminarea dupliilor sau amânarea remedierii), remediate și active.

### 1.5 Concluzii

#### 1.5.1 Inovații MS

- MS încurajează unele echipe să **experimenteze** și să facă o mulțime de **schimbări** fără prea multă planificare în avans.

- Proiectele rămân în general sub **control**, deoarece echipele de programatori și testerii se **sincronizează** frecvent și **stabilizează periodic** modificările.
- Microsoft seamănă cu companiile din multe industrii care fac dezvoltare **incrementală** sau **iterativă** de produse, precum și **inginerie concurentă**.
- MS a **adaptat**, de asemenea, **practicile de inginerie software** introduse anterior de alte companii (cum ar fi diferite **tehnici de testare**)
- MS a **reinventat roata** în multe ocazii (cum ar fi concluzia că acumularea datelor istorice de metrică este utilă pentru analiza tendințelor erorilor și stabilirea programelor realiste ale proiectelor)
- MS a introdus o **abordare structurată** asemănătoare hackerilor pentru dezvoltarea de produse software care funcționează destul de bine atât pentru produsele la scară mică, cât și pentru cele mari (Excel, Office, Publisher, Windows 95,98,2000, Windows NT, Windows XP, Windows Vista, Word, Works și multe altele)

Synch-and-Stabilize	Sequential Development
Product development and testing done in parallel	Separate phases done in sequence
Vision statement and evolving specification	Complete "frozen" specification and detailed design before building the product
Features prioritized and built in 3 or 4 milestone subprojects	Trying to build all pieces of a product simultaneously
Frequent synchronizations (daily builds) and intermediate stabilizations (milestones)	One late and large integration and system test phase at the project's end
"Fixed" release and ship dates and multiple release cycles	Aiming for feature and product "perfection" in each project cycle
Customer feedback continuous in the development process	Feedback primarily after development as inputs for future projects
Product and process design so large teams work like small teams	Working primarily as a large group of individuals in a separate functional department

**Fig. 1.5.1.a.** Sincronizează-și-stabilizează vs. dezvoltare secvențială

- Microsoft este un exemplu fascinant al modului în care **cultura și strategia competitivă** pot conduce **dezvoltarea produselor** și procesul de inovare.
- Cultura Microsoft se concentrează în jurul **programatorilor de PC**, cu feroare antiburocrati cărora **nu le plac** prea multe regulile, structurile sau planificările.

- Principiile din spatele filozofiei de sincronizare și stabilizare adaugă o aparență de ordine lumii în mișcare rapidă, adesea haotică, a dezvoltării de software pentru PC.
- În acest domeniu, nu există gloanțe de argint care să rezolve probleme majore cu o singură soluție simplă. Mai degrabă, există:
  - (1) Abordări, instrumente și tehnici specifice;
  - (2) Câteva reguli rigide;
  - (3) Persoane cu înaltă calificare a căror cultură se aliniază acestei abordări.
- Sincronizarea și stabilizarea introduse de Microsoft cu siguranță reprezintă un uriaș pas înainte în raport de stilurile mai vechi, mai tradiționale, secvențiale și rigide de dezvoltare a produsului (Fig.1.5.1.a.)

### 1.5.2 Strategii competitive ale Microsoft

- Identificarea rapidă a piețelor de masă
- Introducerea produselor care sunt „destul de bune” (în loc să se aștepte până când ceva este „perfect”)
- Îmbunătățirea acestor produse prin evoluția progresivă a caracteristicilor acestora
- Vânderea mai multor versiuni de produse și upgrade-uri clienților din întreaga lume.

### 1.5.3 Punctele slabe ale abordării de dezvoltare Microsoft

- Microsoft nu acordă suficientă atenție arhitecturii produselor, respectiv mecanismelor de prevenire a defectelor.
- Implicarea insuficientă în practicile de inginerie convenționale, cum ar fi proiectarea formală și revizuirea codului.
- Zonele noi afectate de produsele dezvoltate ridică noi provocări pentru metodele de dezvoltare ale MS, care nu sunt întotdeauna rezolvate cu succes.
  - De exemplu, unele domenii noi, cum ar fi video la cerere, au multe componente strâns legate, cu constrângerile timp real, care necesită modele matematice precise privind momentul în care datele video/audio/utilizator pot fi livrate în mod fiabil și la timp.
- Multe produse existente și noi au un număr extrem de mare sau chiar infinit de condiții sau scenarii utilizator potențiale de testat în funcție de hardware-ul și aplicațiile pe care le folosește fiecare client.
  - Aceste noi produse pot beneficia de unele schimbări incrementale în procesul de dezvoltare.
  - De asemenea, necesită mai multă planificare în avans și proiectare arhitecturală a produsului decât o face Microsoft de obicei pentru a minimiza problemele de dezvoltare, testare și operare.

#### 1.5.4 Avantajele tehnologiei MS

- Descompune produsele mari în bucăți mai mici, mai ușor de gestionat (un set de caracteristici ale produsului ordonate funcție de prioritate pe care echipele mici de caracteristici le pot crea în câteva luni).
- Permite proiectelor să deruleze sistematic chiar și atunci când nu poate determina un design de produs complet și stabil de la începutul proiectului.
- Permite echipelor mari să lucreze ca echipe mici prin:
  - Împărțirea lucrării în bucăți
  - Se lucrează în paralel, dar se sincronizează continuu
  - Stabilizare incrementală
  - Găsirea și remedierea continuă a problemelor.
- Facilitează concurența în ceea ce privește contribuțiile clientilor, caracteristicile produsului și timpii scurti de dezvoltare prin:
  - Furnizarea unui mecanism pentru încorporarea intrărilor clientilor,
  - Stabilirea priorităților,
  - Finalizarea cu prioritate a celor mai importante părți
  - Modificarea sau anularea caracteristicilor mai puțin importante.
- Permite unei echipe de produs să fie foarte receptivă la evenimentele de pe piață prin:
  - Având „întotdeauna” un produs gata de expediat,
  - Având o evaluare precisă a caracteristicilor care sunt finalizate,
  - Păstrarea flexibilității proces-produs și stimularea oportunismului pe tot parcursul procesului de dezvoltare.
- Ideile și exemplele Microsoft oferă lecții utile pentru organizații și manageri din multe industrii.
- Abordarea de sincronizare și stabilizare utilizată la Microsoft este potrivită în special piețelor cu ritm rapid, caracterizate prin:
  - Sisteme de produse complexe
  - Cicluri de viață scurte
  - Concurență bazată pe evoluția caracteristicilor produsului și a standardelor tehnice de facto.
- În coordonarea muncii unei echipe mari, multe componente interdependente care se schimbă continuu necesită un nivel constant și ridicat de comunicare și coordonare.
- Este foarte dificil să se asigure un înalt nivel de comunicare și coordonare, oferind în același timp designerilor, inginerilor și oamenilor de marketing libertatea de a fi creativi.
  - Atingerea acestui echilibru este probabil dilema centrală cu care se confruntă managerii de dezvoltare de produse în software-ul pentru PC, precum și în multe alte industrii.

## **2 Tehnologia ORACLE pentru dezvoltarea produselor software**

### **2.1 Probleme actuale ale industriei IT**

#### **2.1.1 Probleme în industria IT**

- Industria IT nu are o reputație foarte bună. În producția SW (2020):
- **Costurile estimate** au fost depășite în medie cu 189%;
- **Termenele asumate** au fost depășite în medie cu 222%;
- În medie, sunt livrate doar 61% din **caracteristicile inițiale convenite**;
- Doar 12% din proiectele SW sunt **livrate la timp și nu depășesc bugetul stabilit**;
- 25% din proiecte **sunt anulate** înainte de a fi finalizate;
- 75% dintre proiectele mari au fost considerate de utilizatorii lor drept **eșecuri operaționale**.

#### **2.1.2 Cauze evidențiate de experiența ORACLE**

- Lipsa unei implicații reale a managementului superior al companiei;
- Întârzierea deciziilor și chiar lipsa deciziei;
- Planuri nerealiste;
- Manager de proiect neexperimentat sau neangajat 100% din punct de vedere al timpului și efortului;
- Neidentificarea risurilor și lipsa planurilor de acțiune adecvate;
- Monitorizare financiară slabă;
- Lipsa personalului calificat corespunzător;
- Lipsa unui plan de bună calitate.

#### **2.1.3 Concluzie**

- În cele mai multe cazuri, cauzele eșecurilor în industria SW sunt **deficiențele de management**

### **2.2 Setul ORACLE de metodologii de dezvoltare**

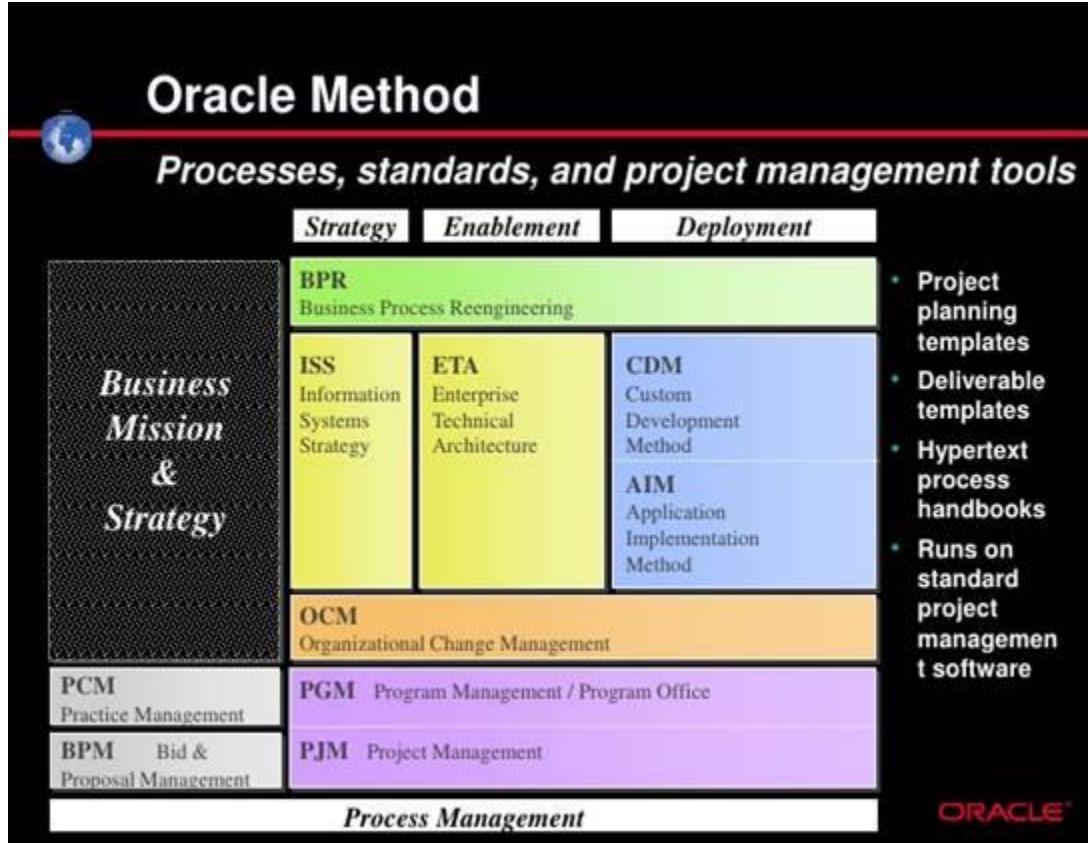
- Tradiția și experiența ORACLE acumulate în procesul de dezvoltare a proiectelor de mare anvergură au impus necesitatea elaborării unei metodologii de dezvoltare.

#### **2.2.1 Idee fundamentală**

- **Idea de bază:** construirea unui cadru de parteneriat între client și dezvoltator
- **Obiectiv:** un set de metodologii de dezvoltare orientate către mai multe domenii funcționale sau de afaceri dedicate liderilor de proiect

#### **2.2.2 ORACLE Set de metodologii de dezvoltare**

Setul de metodologii ORACLE conține următoarele componente (Fig.2.2.2.a)



**Fig. 2.2.2.a.** Setul de metodologii ORACLE

- **BPR - Business Processes Re-engineering** este principala metodă care aliniază o organizație cu strategia sa pentru a-și pune în practică viziunea și pentru a obține beneficii
- **Soluțiile tehnologice** acoperă domenii precum strategia tehnologică, arhitectura tehnică, întreținerea software-ului și administrarea datelor. Printre soluțiile tehnologice joacă un rol important:
  - **ISS** – Strategia Sistemelor Informaționale
  - **ETA** – Arhitectura tehnică a întreprinderii
- Pentru **dezvoltarea aplicațiilor** sunt utilizate **două metodologii** distincte în funcție de natura clientului:
  - **CDM** – Custom Development Method care structurează dezvoltarea proiectelor pe baza aplicațiilor construite cu ajutorul instrumentelor ORACLE (Oracle Designer, Oracle Programmer, Oracle Developer)
  - **AIM** – Metoda de implementare a aplicațiilor centrată pe realizarea proiectelor pornind de la Aplicații ORACLE
- **Pregătirea organizației** se referă la managementul proiectelor și la managementul schimbării organizației
  - **OCM** – Managementul schimbării organizaționale pregătește organizația să accepte schimbarea asociată cu realizarea proiectelor SW.
  - Succesul realizării unor proiecte valoroase SW este în legătură directă cu capacitatea organizației de a se adapta la situații noi și de a accepta schimbarea

- **Management** - există diferite tipuri de management specific:

- **PCM** – Practice Management
- **BPM** – Bid & Proposal Management
- **PGM** – Program Management
- **PJM** – Project Management

## 2.3 Responsabilitățile managerului de proiect

### 2.3.1 Responsabilitățile managerului de proiect

- (1) Reprezinta în același timp clientului și managementul consultantului;
- (2) Este considerat responsabil pentru satisfacerea cerințelor ambelor părți implicate în ceea ce privește așteptările proiectului;
- (3) Trebuie să înțeleagă obiectivele de afaceri și să conceapă o viziune clară pentru a le îmbunătăți;
- (4) Trebuie să fie implicat timpuriu în relația cu clientul, de preferință înainte de semnarea contractului;
- (5) Este responsabil în fața furnizorilor de resurse din organizația de consultantă, pentru utilizarea eficientă a resurselor;
- (6) Este responsabil în fața clientului pentru atingerea obiectivelor asumate în cadrul proiectului.

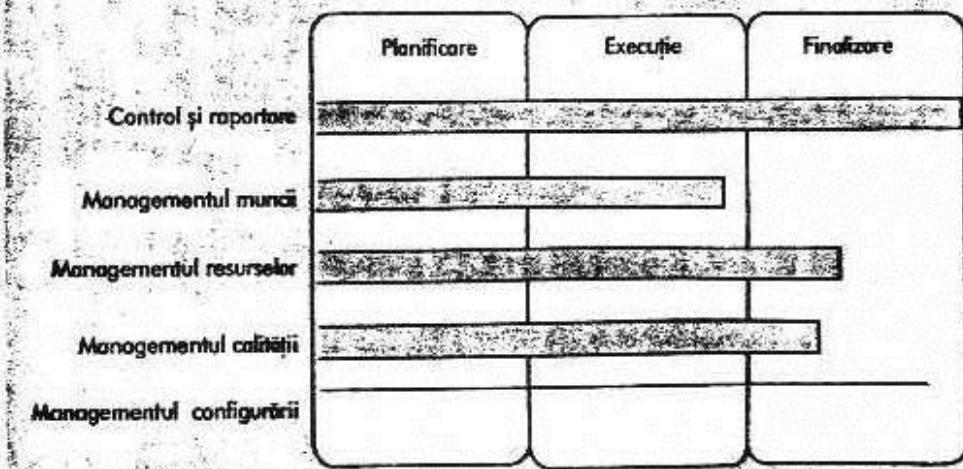
### 2.3.2 Sarcinile principale ale managerului de proiect:

- (1) Să ia decizii corecte sub presiunea riscurilor, incertitudinii și a unui volum mare de informații potențiale relevante.
- (2) Să ducă la înndeplinire sarcinile proiectului de regulă în circumstanțe nefavorabile cum ar fi:
  - Personal diferit și eterogen;
  - De obicei, autoritate scăzută asupra oamenilor;
  - Control limitat asupra personalului.

## 2.4 Fazele ciclului de viață ORACLE

- ORACLE utilizează trei faze în procesul de dezvoltare:
  - (1) Planificare
  - (2) Execuție
  - (3) Finalizare

## Fazele și procesele unui proiect



**Fig. 2.4.a. Fazele si procesele unui proiect ORACLE**

### 2.4.1 Faza planificare

- Este prima fază de dezvoltare. Consta in:
  - (1) Actualizarea **amplitudinii** proiectului pentru a reflecta modificările impuse de negocierea și semnarea contractului;
  - (2) Elaborarea unui **plan detaliat pentru faza de execuție**;
  - (3) Obținerea **angajamentului** clientului privind responsabilitățile și resursele pe care le va furniza;
  - (4) Obținerea **avizelor** pentru trecerea în faza de execuție de la conducerea clientilor și de la conducerea consultanței;
  - (5) Identificarea eventualelor **modificări de infrastructură** necesare pentru susținerea fazei de execuție;
  - (6) Alocarea și pregătirea **resurselor umane** pentru dezvoltarea proiectelor.

### 2.4.2 Faza execuție

- (1) Direcționarea și adaptarea **efortului** în funcție de amprenta proiectului, calitatea impusă și costurile furnizate
- (2) Anticiparea **riscurilor** posibile și pregătirea măsurilor pentru prevenirea sau corectarea acestora
- (3) Rezolvarea cu eficiență a **problemelor curente** care apar
- (4) Asigurarea **integrității și coerentiei** livrabilelor în conformitate cu obiectivele proiectului

### 2.4.3 Faza finalizare

- (1) Obținerea **acceptului clientului** pentru produsul livrat
- (2) Finalizarea cu succes a **relației contractuale**

- (3) Transferarea documentației și a mediului de producție către client
- (4) De-alocarea resurselor umane și fizice utilizate
- (5) Documentarea și arhivarea rezultatelor

## 2.5 Procesele unui proiect ORACLE

### 2.5.1 Definiția procesului. Tipuri de procese

- **Procesul:** - este un ansamblu de activități conexe, cu grade ridicate de interdependentă, având ca rezultat unul sau mai multe livrabile critice pentru proiect
- În ORACLE sunt definite următoarele tipuri de procese (fig.2.4.a.):
  - (1) Control și Raportare
  - (2) Managementul Muncii
  - (3) Managementul resurselor
  - (4) Managementul calității
  - (5) Managementul configurației

### 2.5.2 Procesele în tehnologia ORACLE

(1) **Proces de control și raportare** – se ocupă de amplitudinea proiectului precum și de relațiile cu clientul. În acest scop se utilizează:

- Proceduri de control al modificărilor (schimbărilor)
- Proceduri de gestionare a riscurilor
- Proceduri de gestionare a problemelor apărute
- Proceduri de raportare privind evoluția proiectului

(2) **Managementul muncii** – stabilește un **plan de lucru** consistent (aprobat de client) care detaliază:

- Amplitudinea proiectului
- Obiectivele
- Modul de abordare a proiectului
- **Planul de lucru** este utilizat de către manager pentru:
  - Monitorizare
  - Control
  - Raportare
  - Replanificarea execuției proiectului

(3) **Managementul resurselor** – este responsabil pentru furnizarea resurselor fizice și umane pentru proiect.

- Este strâns interconectat cu Procesul de management al muncii

(4) **Managementul calității** – stabilește un **Plan de calitate** care definește standarde și proceduri pentru asigurarea consistenței și coeranței auditurilor de calitate, pentru livrabile și pentru membrii echipei de proiect

**Obs.** Activitatea de testare nu face parte din Procesul de management al calității, deși sunt interdependente, ci este un proces separat inclus în CDM sau AIM

(5) **Managementul configurației** – se referă la identificarea, controlul și monitorizarea oricărui articol produs în timpul dezvoltării proiectului.

- În cadrul acestui proces, **livrabilele** sunt asamblate pas cu pas într-un produs unitar.
- Acest proces trebuie să asigure faptul că numai modificările aprobate de client și de consultanța managementului vor afecta produsul final.

### 2.5.3 Observații generale

- Procesele acționează în **diferite faze ale proiectului**
- **Managerul de proiect** este persoana care stabilește **nivelul de responsabilitate și autoritate** pentru fiecare membru al echipei de dezvoltare
- Managerul de proiect stabilește **setul de reguli și standarde** care trebuie urmate în timpul procesului de dezvoltare

### 2.6 Concluzii

- ORACLE folosește **metode practice și clare**;
- Pune accent pe **construirea unui mediu de parteneriat între client și dezvoltator** ca fundament al succesului;
- Folosește **echipe experimentate** care dezvoltă proiecte având tot timpul în vedere, **obiectivele proiectului și beneficiile organizației** dezvoltatoare.

### 3 Procesul de dezvoltare RATIONAL

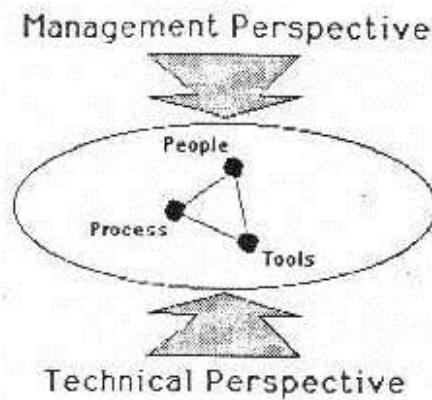
#### 3.1 Calea Rational

- **Procesul de dezvoltare Rational** este:
  - (1) Iterativ și incremental;
  - (2) Orientat pe obiecte;
  - (3) Gestionat și controlat;
  - (4) Înalt automatizat.
- Este suficient de **generic** pentru a fi adaptat la o mare varietate de produse software și proiecte, atât ca dimensiune, cât și ca domeniu de aplicare.
- Este **centrat** în jurul a trei poli:
  - (1) Oameni;
  - (2) Proces;
  - (3) Instrumente și metode.

#### 3.2 Ciclul de viață general al software-ului

##### 3.2.1 Două perspective de abordare

- Procesul Rațional poate fi abordat din **două perspective** diferite și integrate (Fig. 3.2.1.a):
  - (1) O **perspectivă managerială**, care se ocupă de aspectele financiare, strategice, comerciale și umane.
  - (2) O **perspectivă tehnică**, care se ocupă de aspecte legate de calitate, inginerie și metodă de proiectare.



**Fig. 3.2.1.a.** Procesul Rational. Perspectiva managerială și perspectiva tehnică

##### 3.2.2 Perspectiva managerială: Cicluri și faze

- Din perspectiva managerială, **ciclul de viață al software-ului** este organizat pe **4 faze principale** (Fig. 3.2.2.a):
  - (1) **Faza Inițiere**:
    - Se pornește cu o idee bună;

- Specificarea viziunii produsului final;
- Specificarea cazului său de afaceri (business case);
- Definirea scopului proiectului.

(2) **Faza Elaborare:**

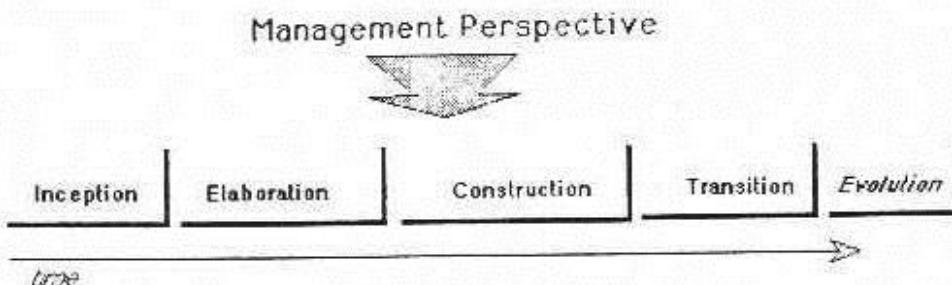
- Planificarea activităților necesare;
- Planificarea resurselor necesare;
- Specificarea caracteristicilor;
- Proiectarea arhitecturii.

(3) **Faza Construcție:**

- Construirea produsului;
- Evoluția viziunii, arhitecturii și planurilor până când produsul (viziunea finalizată) este gata de transfer către comunitatea de utilizatori.

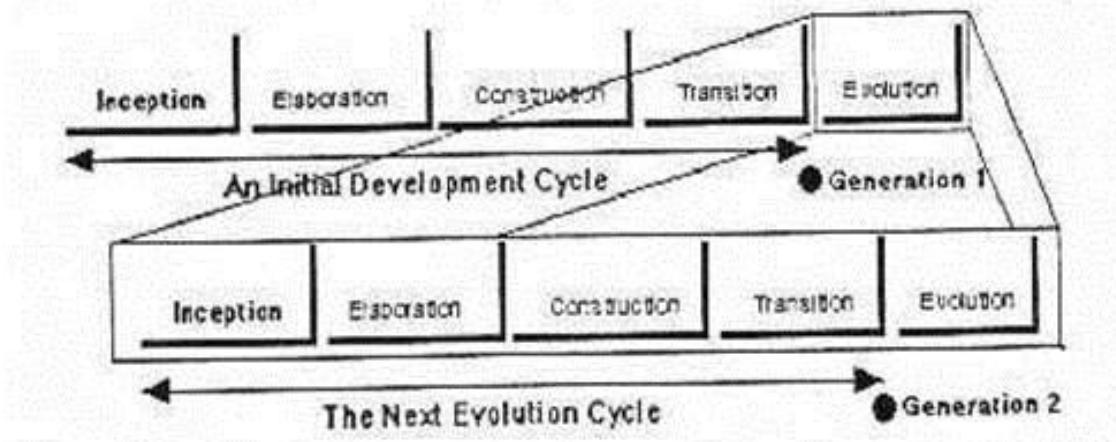
(4) **Faza Tranzitie:**

- Efectuarea tranzitiei de la produs la comunitatea de utilizatori;
- Manufacturarea, livrarea, instruirea, susținerea, întreținerea produsului până când utilizatorii sunt mulțumiți.



**Fig.3.2.2.a.** Perspectiva managerială

- Trecerea prin cele 4 faze se numește **ciclu de dezvoltare** și produce o **generație software** (Fig. 3.2.2.b).
- Un produs existent poate evolu către **următoarea generație** prin repetarea acelorași secvențe de faze, cu un accent diferit însă asupra diferitelor faze.
  - Această perioadă se numește **ciclu evolutiv**.
- Pe măsură ce produsul trece în cele din urmă prin mai multe cicluri, sunt produse **noi generații**.
- **Ciclurile evolutive** pot fi declanșate de:
  - *Îmbunătățiri* sugerate de utilizator;
  - *Schimbări în contextul utilizatorilor*;
  - *Schimbări în tehnologia de bază*;
  - *Reacția la competiție* etc.

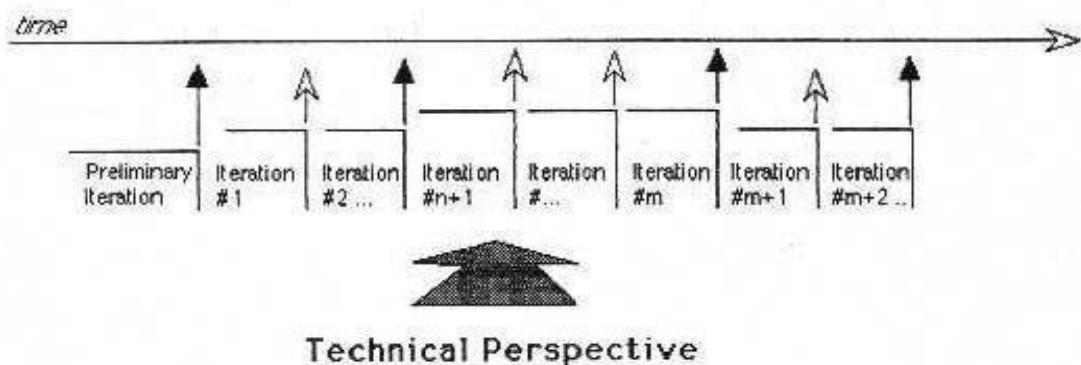


**Fig. 3.2.2.b. Generații și cicluri evolutive**

- În practica curentă ciclurile se pot **suprapune** ușor:
  - Faza de inițiere și elaborare poate începe în timpul fazei de tranziție a ciclului anterior.

### 3.2.3 Perspectiva tehnică: iterații

- Din punct de vedere tehnic, **dezvoltarea software-ului** este văzută ca o **sucesiune de iterații**, prin care software-ul în curs de dezvoltare evoluează progresiv. (Fig. 3.2.3.a).

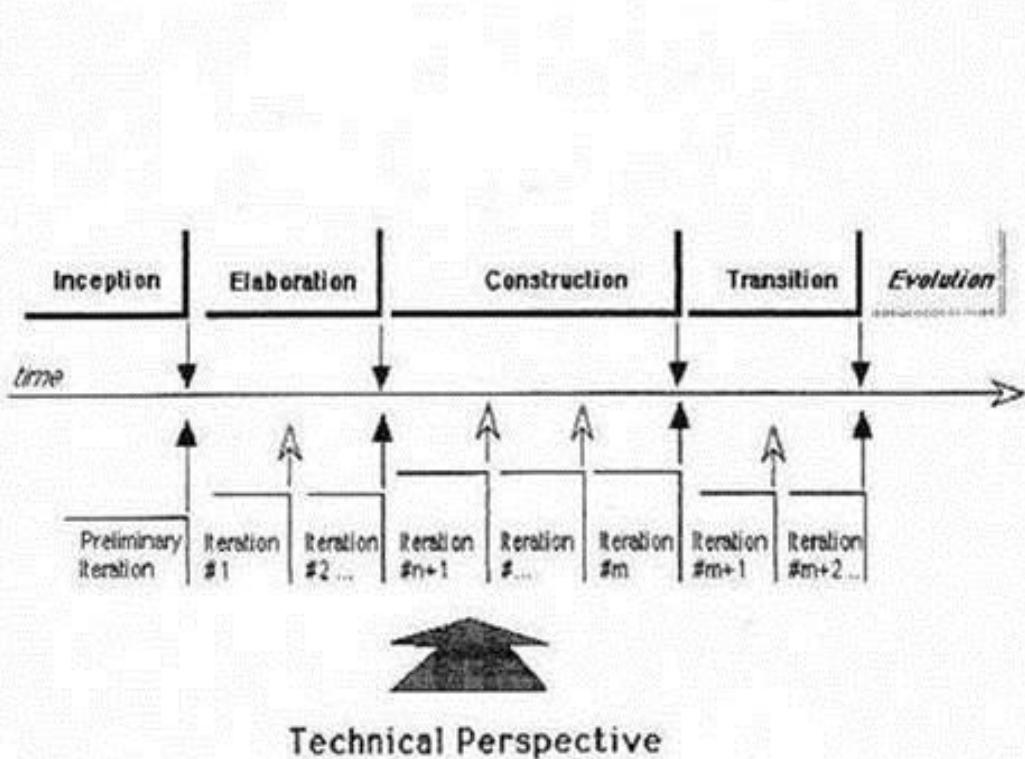


**Fig.3.2.3.a. Perspectiva tehnică**

- Fiecare **iterație** se încheie prin **lansarea unui produs executabil** (release) (care poate fi un subset al viziunii complete, dar util din perspectiva ingineriei sau a utilizatorului)
- Fiecare **release** este însoțit de **artefakte suport**: *planuri, descrierea release-ului, documentația utilizator*, etc.
- O **iterație** constă din activitățile de **planificare, analiză, proiectare, implementare și testare** în diferite proporții, în funcție de locul în care se află iterăția în ciclul de dezvoltare.

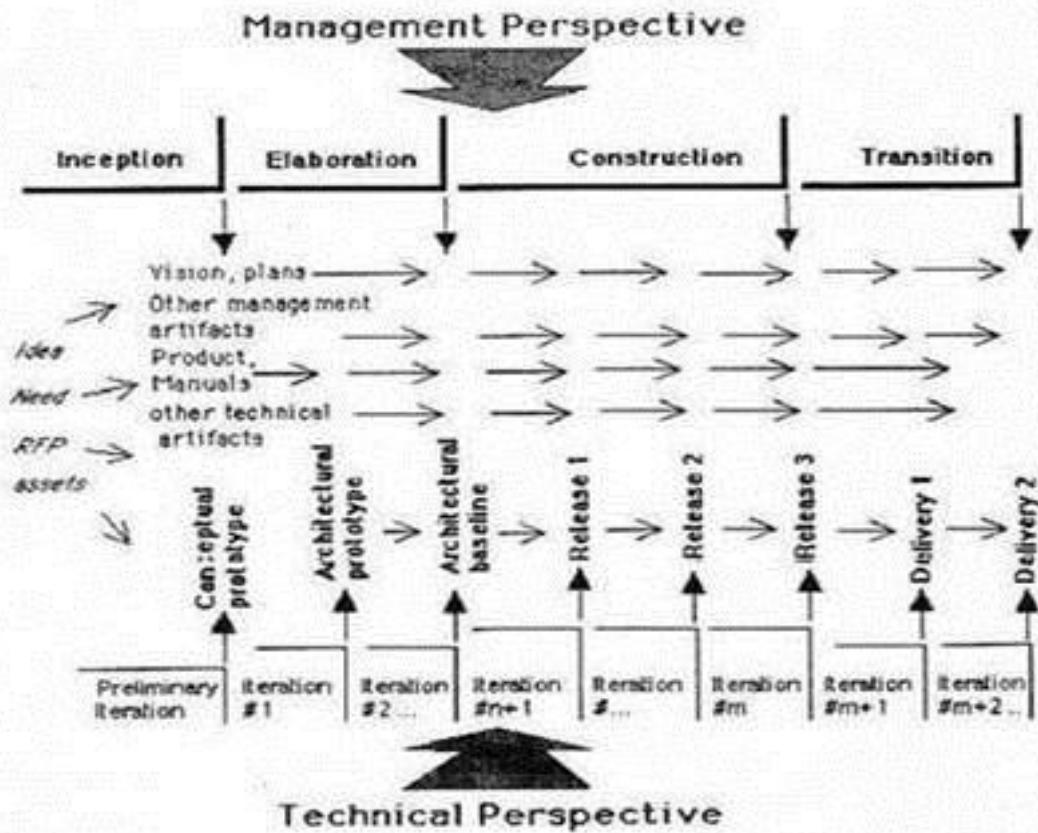
### 3.2.4. Reconcilierea celor două perspective

- Perspectiva managementului și perspectiva tehnică se reconciliază prin aliniere.
- În particular, **sfârșitul fazelor** este **sincronizat** cu sfârșitul **iterațiilor**.
  - Cu alte cuvinte, fiecare **fază** este împărțită în una sau mai multe **iterații**. (Fig. 3.2.4.a).
- Cele două perspective – managerială și tehnică – se **sincronizează** de fapt pe câteva **borne** (milestone-uri) bine identificate.



**Fig.3.2.4.a.** Reconcilierea celor două perspective

- Ambele perspective contribuie la un **set comun de produse și artefacte** care evoluează în timp
- Unele artefacte sunt mai mult sub controlul **părții tehnice**, altele mai mult sub controlul **părții de management** (Fig. 3.2.4.b.)



**Fig.3.2.4.b.** Articalele ciclului de viață al proiectului

- **Elementele tangibile** care constituie **bornele** (milestone-uri), mult mai mult decât simple date dintr-un calendar sunt:
  - (1) **Disponibilitatea artefactelor**;
  - (2) Satisfacerea **criteriilor de evaluare** stabilite pentru **produs** și **artefakte**.
- La fel ca și ciclurile, **iterațiile** se pot suprapune ușor:
  - De exemplu, activitățile de planificare sau de arhitectură ale iterăției N pot fi începute spre sfârșitul iterăției N-1;
  - În unele cazuri, unele iterății pot avea loc în paralel.

### 3.2.5. Discriminatorii

- Procesul de dezvoltare este influențat de următorii **factori**:
  - (1) Accentul și importanța diferențelor **faze**;
  - (2) **Criteriile de intrare și ieșire**;
  - (3) **Artefactele** implicate de-a lungul unui ciclu de dezvoltare;
  - (4) **Numărul și lungimea iterăților**.

- Toți acești **factori** pot varia în funcție de patru caracteristici majore ale proiectului, numite **discriminatori de proces**.
- În ordinea descrescătoare a impactului, cei mai importanți discriminatori de proces sunt:

(1) **Contextul de afaceri.** Există trei tipuri de context de afaceri:

- (a) **Lucrare pe bază de contract**, în care dezvoltatorul produce software-ul conform specificațiilor clientului dat și numai pentru acest client
- (b) **Dezvoltare speculativă sau comercială**, în cazul în care dezvoltatorul produce software pentru a fi introdus pe piață
- (c) **Proiect intern**, în care clientul și dezvoltatorul sunt în aceeași organizație

(2) **Dimensiunea efortului** (size) de dezvoltare software definită de unele **metriki** specifice:

- Instrucțiuni sursă livrate;
- Puncte funcționale, etc.;
- Număr de persoane-lună;
- Cost.

(3) **Gradul de noutate** – care este gradul de „precedență” al efortului software în raport cu organizația dezvoltatoare, respectiv, dacă dezvoltarea se află la primul ciclu de dezvoltare sau la un ciclu ulterior. Acest discriminant include:

- Maturitatea organizației și a procesului;
- Activele sale;
- Portofoliul său de abilități actuale;
- Capabilitățile de asamblare și formarea a unei echipe, de achiziționare de instrumente sau de alte resurse.

(4) **Tipul de aplicație** – se referă la domeniul țintă care poate fi:

- MIS – (Management Information System);
- Comandă și control;
- Încorporat în timp real;
- Dezvoltare de software pentru instrumente ambientale, etc.

• **Constrângerile specifice** pe care domeniul le-ar putea impune dezvoltării se referă la:

- Siguranță;
- Performanță;
- Internaționalizare;
- Constrângere de memorie, de timp, de resurse, etc.,.

• Acest capitol își propune:

(1) Mai întâi descrie **procesul generic**, adică partea procesului care se aplică tuturor tipurilor de dezvoltări software, într-o gamă largă și generală a acestor **discriminatori**.

(2) Apoi descrie câteva *exemple specifice* ale procesului pentru unele valori ale discriminanților.

### 3.2.6 Efort și planificare

- **Fazele de dezvoltare** a proiectului SW nu sunt identice în termeni de planificare și efort.
- **Planificarea și efortul** poate varia considerabil în funcție de factorii discriminatori ai proiectului,
- Pentru un **ciclu tipic de dezvoltare inițială** pentru **un proiect de dimensiune medie** se pot avansa următoarele procente (Fig.3.2.6.a):

	Incepție	Elaborare	Construcție	Tranziție
Efort	5%	20%	65%	10%
Planificare	10%	30%	50%	10%

**Fig.3.2.6.a.** Efort și planificare pentru un proiect de dimensiune medie

- Pentru un **ciclu evolutiv**, fazele de inițiere și elaborare pot fi reduse considerabil.
- De asemenea, folosind anumite instrumente și tehnici, cum ar fi constructorii de aplicații, faza de construcție poate fi mult mai mică decât faza de inițiere și elaborare împreună.

## 3.3 Fazele procesului rațional

### 3.3.1 Faza de incepție

- **Obiectivele fazei de incepție** sunt:
  - (1) Să construiască o **viziune originală** asupra unui **produs potențial** pe care să o transforme într-un **proiect efectiv**.
  - (2) Să stabilească cazul de afacerie (business case) pentru un produs nou sau o actualizare majoră pentru unul existent.
  - (3) Să specifică scopul al proiectului.
- **Rezultatele (ieșirile) fazei de incepție**:
  - (1) Pentru **dezvoltarea unui nou produs**, principalul rezultat al acestei faze este o **decizie de a trece sau nu la faza următoare** ("go-no go" decision) și de a investi timp și bani pentru a analiza în detaliu dacă ceea ce urmează a fi construit, poate să fie construit și cum poate fi construit.

(2) Pentru **evoluția unui produs existent**, aceasta poate fi o **fază simplă și scurtă**, bazată pe solicitările utilizatorilor sau ale clienților, pe rapoartele de probleme și pe noile progrese tehnologice.

(3) Pentru **o dezvoltare contractuală**, decizia de a continua se bazează pe experiența domeniului specific și pe competitivitatea organizației de dezvoltare în acest domeniu sau piață.

- În acest caz, faza de inițiere poate fi încheiată printr-o **decizie de a licita**, sau prin **oferta în sine**.
- Ideea se poate baza pe un prototip de cercetare existent, a cărui arhitectură poate fi sau nu potrivită pentru software-ul final.

- **Criterii de intrare** în **faza de incepție**:

- (1) O **viziune originală**;
- (2) Un **sistem moștenit**;
- (3) O **cerere de propunere** (Request For Proposal);
- (4) **Generația anterioară și o listă de îmbunătățiri**;
- (5) Unele **active** (software, know-how, active financiare);
- (6) Un **prototip conceptual** sau o **machetă**.

- **Criteriile de ieșire** din **faza de incepție**:

- (1) Un **caz de afaceri inițial** (initial business case) care conține cel puțin:
  - O **formulare clară a viziunii produsului** -- cerințele de bază – exprimate în termeni de funcționalitate, domeniul de aplicare, performanță, capacitate, bază tehnologică;
  - **Criteriile de succes** (de exemplu proiecția veniturilor);
  - O **evaluare inițială a riscului**;
  - O **estimare a resurselor** necesare pentru finalizarea fazei de elaborare.
- Optional, la sfârșitul fazei de incepție, putem avea:
  - (2) Un **model inițial de analiză a domeniului** (~10%-20% complet), care identifică principalele cazuri de utilizare cheie și este suficient pentru a ghida eforturile arhitecturale;
  - (3) Un **prototip arhitectural inițial**, care în această etapă poate fi un prototip provizoriu (de aruncat).

### **3.3.2 Faza de elaborare**

- **Scopul principal** al **fazei de elaborare** se referă la:

- (1) Analiza mai amănuntită a domeniul problemei;
- (2) Definirea și stabilizarea arhitecturii;
- (3) Abordarea elementelor cu cel mai mare risc ale proiectului;
- (4) Elaborarea unui plan cuprinzător care să arate cum vor fi realizate următoarele 2 faze. Planul conține:
  - O **viziune de bază a produsului** (adică un set inițial de cerințe) bazată pe un **model de analiză**;

- Criterii de evaluare pentru cel puțin prima iteratăie de construcție
  - O arhitectură software de bază;
  - Resursele necesare dezvoltării și implementării produsului, în special în ceea ce privește oamenii și instrumentele;
  - O planificare;
  - O rezoluție a riscurilor suficientă pentru a face o estimare „de înaltă fidelitate” a costului, programului și calității fazei de construcție.
- **Rezultatele (ieșirile) fazei de elaborare:**
- (1) Un prototip arhitectural executabil care este construit în una sau mai multe iterării în funcție de domeniul, dimensiunea, riscul, noutatea proiectului, care adreseză:
    - Cel puțin principalele cazuri de utilizare cheie identificate în faza de inițiere;
    - Riscurile tehnice de top ale proiectului.
  - (2) Un prototip evolutiv al producției de cod de calitate, care devine baza arhitecturală.
    - Nu exclude dezvoltarea unuia sau mai multor prototipuri exploratorii, (de aruncat), pentru a atenua riscurile specifice vizând:
      - Rafinarea cerințelor;
      - Fezabilitate;
      - Studii de interfață umană;
      - Demonstrații pentru investitori;
      - Etc.,.
    - Acest prototip devine în final baza arhitecturală.
  - (3) La sfârșitul acestei faze, există din nou un punct de decizie „go-no go” pentru a investi efectiv și a demara construcția produsului sau pentru a licita pentru dezvoltarea completă a contractului.
  - (4) Planurile realizate trebuie să fie suficient de detaliate, iar riscurile suficient de atenuate pentru a putea determina cu acuratețe costul și calendarul de finalizare a dezvoltării.
- **Criteriile de intrare în faza de elaborare:**
- (1) Produsele și artefactele descrise în criteriile de ieșire din faza de inițiere;
  - (2) Proiectul a fost aprobat de către autoritatea de management și finanțare a proiectului;
  - (3) Au fost alocate resursele necesare fazei de elaborare.
- **Criteriile de ieșire din faza de elaborare:**
- (1) Un plan detaliat de dezvoltare a software-ului, care conține:
    - O evaluare actualizată a riscurilor;
    - Un plan de management;

- Un **plan de personal**;
- Un **plan de fază** care arată numărul și conținutul iterăției pentru următoarea fază;
- Un **plan de iterăție**, care detaliază următoarea iterăție;
- **Mediul de dezvoltare** și alte instrumente necesare;
- Un **plan de testare**.

- (2) O **viziune de bază**, sub forma unui **set de criterii de evaluare** pentru produsul final.
- (3) **Criterii de evaluare obiective, măsurabile** pentru evaluarea rezultatelor iterăției/iterățiilor inițiale ale fazei de construcție.
- (4) Un **model de analiză a domeniului** (80% complet), suficient pentru a putea numi arhitectura corespunzătoare „completa”.
- (5) O **descriere a arhitecturii software** (declarând constrângeri și limitări).
- (6) O **arhitectură de bază executabilă**.

### 3.3.3 Faza de construcție

- **Obiectivele fazei de construcție:**

- (1) Primul **release al produsului final**.
- (2) **Faza de construcție** este împărțită în mai multe **iterății** -- dezvoltând **arhitectura de bază** și evoluând în pași sau trepte către **produsul final**.
- (3) La fiecare iterăție, **diferitele artefakte** pregătite în timpul fazei de elaborare (vezi mai sus) sunt **extinse și revizuite**, dar în cele din urmă se **stabilizează** pe măsură ce sistemul evoluează în corectitudine și completitudine.
- (4) În această fază sunt produse **noi artefakte** alături de software-ul însuși.
- **Documentație**, atât internă, cât și pentru utilizatorii finali;
  - **Paturi de testare și suite de testare**;
  - **Desfășurare colaterală** (Deployment collateral) pentru a sprijini următoarea fază (de exemplu garanții de marketing).

- **Pentru fiecare iterăție** a **fazei de construcție**:

- **Criterii de intrare** pentru **iterăție**:

- (1) **Produsul și artefactele** iterăției anterioare.
- (2) **Planul de iterăție** care trebuie să precizeze obiectivele specifice ale iterăției:
- **Capacități suplimentare** în curs de dezvoltare, respectiv care cazuri de utilizare sau scenarii vor fi acoperite;
  - **Riscurile** care sunt atenuate în timpul acestei iterății;
  - **Defectele** ce urmează a fi remediate în timpul iterăției.

- **Criterii de ieșire** pentru **iterăție**:

- (1) Aceleași produse și **artefacte actualizate**, plus:

- (2) Un document de descriere a release-ului, care surprinde rezultatele unei iterații;
  - (3) Cazuri de testare și rezultate ale testelor efectuate asupra produselor;
  - (4) Un plan de iterație, care detaliază următoarea iterație;
  - (5) Criterii de evaluare obiective și măsurabile pentru evaluarea rezultatelor următoarei iterații.
- Criterii de ieșire pentru faza de construcție:
  - Spre sfârșitul fazei de construcție, trebuie produse următoarele artefacte și trebuie stabilite criterii suplimentare de ieșire pentru ultima iterație a fazei:

- (1) Un plan de desfășurare, care specifică, după caz:
  - Ambalare;
  - Preț;
  - Lansare;
  - Suport;
  - Instruire;
  - Strategia de tranziție (de exemplu, un plan de upgrade de la un sistem existent);
  - Producție (de exemplu, realizarea de dischete și manuale).
- (2) Documentația utilizatorului.

### 3.3.4 Faza de tranziție

- Obiectivele fazei de tranziție:
  - (1) Plasarea produsului realizat în mâinile utilizatorilor săi finali.
    - Implică probleme de marketing, ambalare, instalare, configurare, susținere a comunității de utilizatori, efectuarea de corecții etc.
  - (2) Din punct de vedere tehnic, iterațiile continuă cu una sau mai multe release-uri (versiuni) sau livrări:
    - Lansări 'Beta';
    - Versiuni de disponibilitate generală;
    - Remedieri de erori;
    - Release-uri îmbunătățite.
  - (3) Faza este finalizată atunci când comunitatea de utilizatori este mulțumită de produs: acceptarea formală, de exemplu, într-un cadru contractual, sau când toate activitățile pe acest produs sunt încheiate.
  - (4) Este punctul în care unele dintre activele acumulate pot fi reutilizate în următorul ciclu sau prin alte proiecte.
- Criterii de intrare în faza de tranziție:
  - (1) Produsul și artefactele iterației anterioare.
  - (2) Un produs software suficient de matur pentru a fi pus în mâinile utilizatorilor săi.

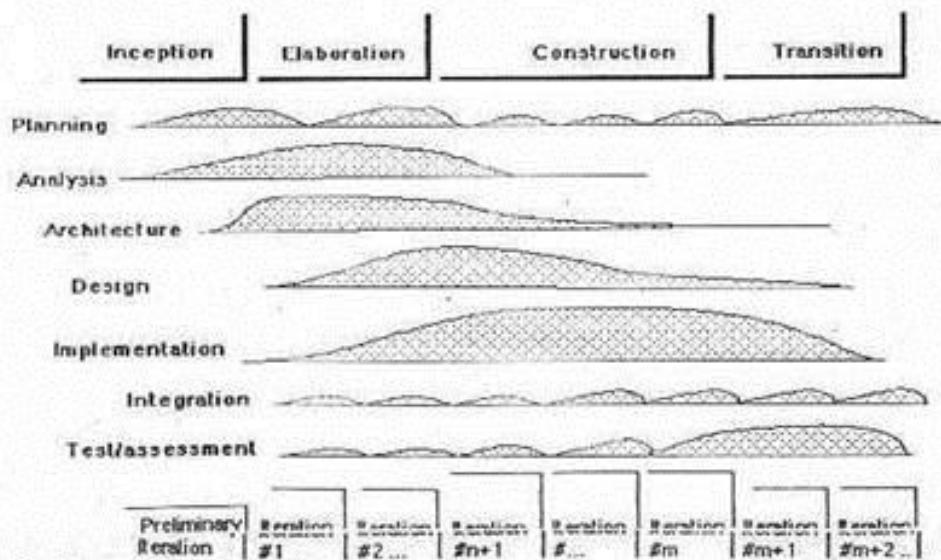
- **Criterii de ieșire** din faza de tranziție:
  - (1) O actualizare a unora dintre documentele anterioare (după caz).
  - (2) Planul care este înlocuit cu o analiză „post-mortem” a performanței proiectului în raport cu criteriile sale de succes inițiale și cu cele revizuite.
  - (3) Un scurt inventar al noilor active ale organizației ca rezultat al acestui ciclu de dezvoltare.

### 3.3.5 Cicluri de evoluție

- Pentru o **evoluție substanțială**, întregul proces poate fi aplicat **recursiv**, începând din nou din faza de inițiere, pentru **un nou ciclu**.
- Deoarece există deja un produs, **faza de incepție** poate fi redusă considerabil, în comparație cu un ciclu de dezvoltare inițial.
- **Faza de elaborare** poate fi, de asemenea, limitată și concentrată mai mult pe aspectele de planificare decât pe evoluția analizei sau a arhitecturii.
- Altfel spus: **ciclurile se pot suprapune ușor**.
- Pot avea loc evoluții minore în extinderea **fazei de tranziție**, adăugând una sau mai multe iterații.
- Alternativ, **faza de tranziție** poate fi încheiată printr-un **proces de sfârșit de viață**, adică produsul nu mai evoluează, dar trebuie întreprinse unele acțiuni specifice pentru a-l termina sau retrage.

### 3.4 Activități în Procesul Rațional

- Denumirile fazelor **Procesului Rațional** sunt diferite de termenii care descriu **activitățile** de dezvoltare specifice: analiză, proiectare, testare etc.
- O astfel de **activitate** nu se limitează doar la o **anumită fază**.
- **Activitățile** rămân **independente** raportat la termenii folosiți de alți autori, la standarde sau jargoane specifice domeniului.
- Ca atare, aceste **activități** au loc, dar în **grade diferite** în **fiecare fază și iterație** a ciclului de dezvoltare (Fig. 3.4.a.).
- **Natura exactă și conținutul** iterațiilor evoluează în timp, deși toate sunt structurate în același mod.
- **Începutul unei activități** nu este legat de **sfârșitul alteia**, de exemplu, proiectarea nu începe când analiza se încheie.
- Diferitele **artefacte** asociate **activităților** sunt revizuite pe măsură ce problema sau cerințele sunt mai bine înțelese.
- Într-un proces iterativ, **activitățile** de **planificare, testare și integrare** sunt **răspândite în mod incremental** pe tot parcursul ciclului, în **fiecare iterație** și nu sunt grupate masiv la început, respectiv la sfârșit.
- **Activitățile** nu apar ca **etape** sau **faze separate** în proces.



	<b>Planning and management</b>	15%
	<b>Analysis/requirements</b>	10%
	<b>Design/integration</b>	15%
	<b>Implementation/functional tests</b>	30%
	<b>Measurement/assessment/acceptance test</b>	15%
	<b>Tools/environment/change management</b>	10%
	<b>Maintenance (fixes during development)</b>	5%

**Fig. 3.4.a.** Activități în procesul Rational

- Pentru un **ciclu de dezvoltare inițial tipic** pentru **un proiect de dimensiune medie** se estimează următoarele rapoarte pentru diferitele activități, deși aceste cifre pot varia considerabil funcție de factorii discriminatori ai proiectului (Fig.3.4.b.).

<b>Planificare și management</b>	15%
<b>Analiză/cerințe</b>	10%

Proiectare/integrare	15%
Implementare/teste functionale	30%
Măsurători/estimări/teste de acceptanță	15%
Unelte/mediu/managementul schimbării	10%
Mentenanță (remedieri în timpul dezvoltării)	5%

**Fig. 3.4.b.** Ponderea activităților pentru un proiect mediu în procesul Rational

### 3.5 Artefacte ale ciclului de viață

- Procesul de dezvoltare **Rational** nu este bazat pe documente
- Artefactul său principal trebuie să rămână, în orice moment, produsul software *în sine*.
- Documentația trebuie să rămână restrânsă și limitată la câteva documente care aduc valoare reală proiectului din punct de vedere managerial sau tehnic.
- **Rational** sugerează următorul set tipic de documente.

#### 3.5.1 Artefacte de management

- Artefactele de management nu sunt produsele. Ele sunt folosite pentru:
  - (a) Conducerea și monitorizarea **progresul proiectului**;
  - (b) Estimarea **riscurilor**;
  - (c) Ajustarea **resurselor**;
  - (d) Pentru a oferi **vizibilitate clientului** (într-un cadru contractual) sau **investitorilor**.
- **Rational** recomandă următoarele **8 artefacte de management**:
  - (1) Un **document de politică organizațională**, care este codificarea procesului organizației. El conține o *instanță a procesului generic de organizare*.
  - (2) Un **document Viziune**, care descrie:
    - (2.1) *Cerințele la nivel de sistem*;
    - (2.2) *Calitățile*;
    - (2.3) *Prioritățile*.
  - (3) Un **document caz de afaceri**, care descrie:

- (3.1) *Contextul finanțier*;
- (3.2) *Contractul*;
- (3.3) *Rentabilitatea proiectată a investiției* etc.

(4) Un **document Plan de dezvoltare**, care conține în special:

- (4.1) *Planul general de iterăție*;
- (4.2) *Planul pentru iterăția curentă*;
- (4.3) *Planul pentru iterăția viitoare*.

(5) Un **document referitor la criteriile de evaluare**, care conține:

- (5.1) *Cerințele*;
- (5.2) *Criterii de acceptare*;
- (5.3) *Alte obiective tehnice specifice, care evoluează de la o bornă (milestone) majoră la o altă bornă majoră*.
- (5.4) *Scopurile iterăției*;
- (5.5) *Nivelurile de acceptanță*.

(6) **Documente de descriere a versiunii** (release-ului) pentru fiecare versiune.

(7) **Un document de desfășurare** (deployment), care sintetizează informații suplimentare utile pentru:

- (7.1) *Tranzitie*;
- (7.2) *Instruire*;
- (7.3) *Instalare*;
- (7.4) *Vânzări*;
- (7.5) *Fabricare*;
- (7.6) *Punere în funcțiune (cut-over)*.

(8) **Documente de evaluare a stării**:

- (8.1) *Instantanee periodice ale stării proiectului cu indicatorii (metricele) de progres*;
- (8.2) *Personalul*;
- (8.3) *Cheltuielile*;
- (8.4) *Rezultate*;
- (8.5) *Riscuri critice*;
- (8.6) *Elemente de acțiune*;
- (8.7) *Analiza post-mortem*.

### **3.5.2 Artefacte tehnice**

- Artefactele tehnice ale procesului **Rational** sunt:
  - (1) **Bunurile livrate** (software executabil și manuale).
  - (2) **Planurile** care au fost utilizate pentru fabricarea bunurilor livrate (modele software, codul sursă și alte informații de inginerie utile pentru înțelegerea și evoluția produsului).

- **Rational** recomandă următoarele **3 artefakte tehnice**:
  - (1) **Manual de utilizare**, dezvoltat devreme în ciclului de viață.
  - (2) **Documentație software**, de preferință sub formă de
    - *Cod sursă auto-documentat*, susținut de instrumente adecvate pentru a-l menține
    - *Modele (cazuri de utilizare, diagrame de clasă, diagrame de proces, etc.)* captureate și menținute cu instrumentele CASE adecvate.
  - (3) Un **document de arhitectură software**, care descrie:
    - *Structura generală a software-ului;*
    - *Descompunerea sa în elemente majore: categorii de clase, clase, procese, subsisteme, definirea interfețelor critice și justificarea deciziilor cheie de proiectare.*
- **Artefactele** enumerate în **criteriile de intrare și ieșire** din secțiunea 3 pot fi mapate pe unul dintre aceste **11 documente**.
- În funcție de tipul de proiect, acest **set de documente tipic** poate fi **extins sau contractat**, unele documente pot fi comasate.
- **Documentele nu** trebuie să fie **documente pe hârtie**
  - Ele pot fi, de asemenea, *foi de calcul, fișiere text, baze de date, adnotări în codul sursă, documente hipertext* etc.
  - *Sursa de informații* corespunzătoare trebuie însă să fie clar identificată, ușor accesibilă și o parte din istoria acesteia trebuie păstrată.

### 3.5.3 Cerințe

- **Procesul Rational** nu este determinat numai de **cerințe** (requirement driven).
- **Cerințele** pentru produs **evolvează** pe parcursul unui ciclu și iau **diferite forme**:
  - (1) **Cazul de afaceri** prezintă principalele constrângeri, mai ales în ceea ce privește resursele care pot fi cheltuite.
  - (2) **Documentul de viziune** descrie doar **cerințele cheie ale sistemului** din **perspectiva unui utilizator**. Evolvează doar lent în timpul ciclului de dezvoltare.
  - (3) **Cerințele mai detaliate** sunt elaborate în **faza de elaborare**, sub formă de **cazuri de utilizare și scenarii**.
- **Cerințele** sunt **rafinate incremental** pe parcursul **fazei de construcție**, pe măsură ce produsul și nevoile utilizatorilor devin mai bine înțelese.
  - Aceste **cerințe mai detaliate** sunt cuprinse în documentul referitor **criteriile de evaluare**;
  - Ele direcționează **definirea conținutului** iteratiilor de construcție și tranzitie și sunt menționate în **planul de iteratie**.

### 3.6 Exemple de procese Rational

- Procesul **Rational** ia diferite aspecte funcție de valorile discriminantilor (Fig. 3.6.a.)

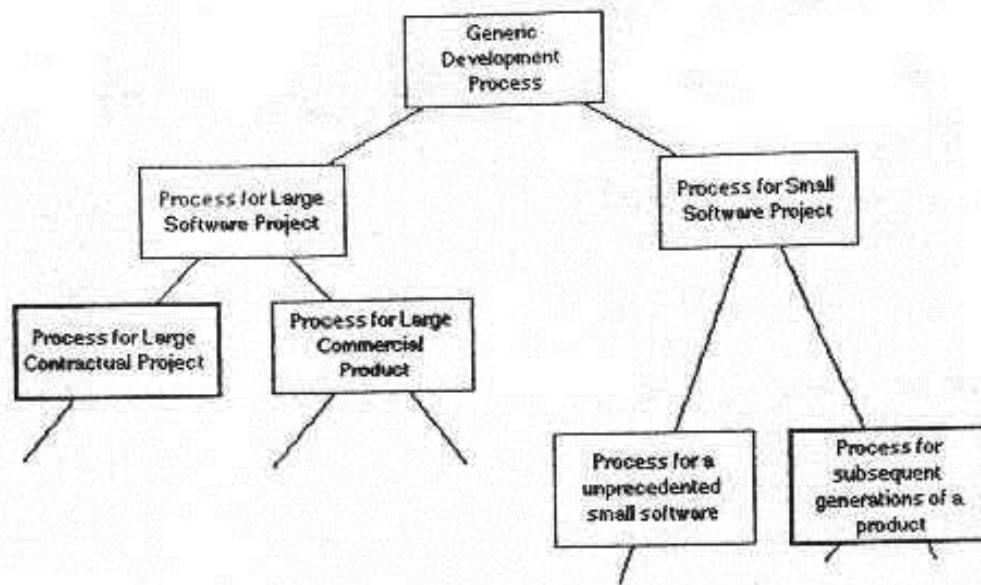
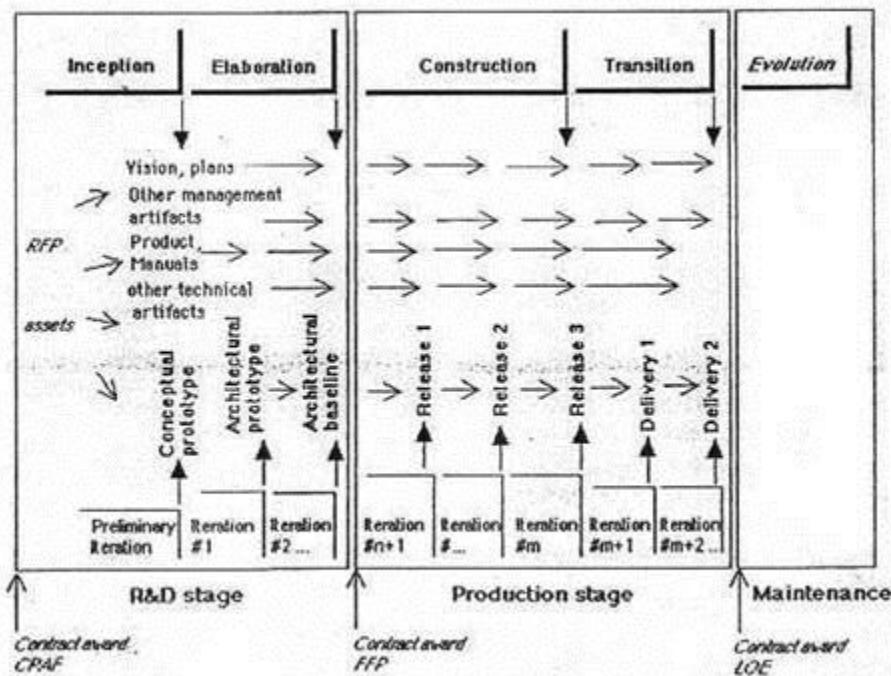


Fig. 3.6.a. Taxonomia proiectelor procesului de dezvoltare Rational

### 3.6.1 Proces rațional pentru dezvoltarea software pentru proiecte contractuale de mari dimensiuni

- Rational propune ca dezvoltarea de software mari dimensiuni să se desfășoare în 3 etape, asociate cu 3 tipuri diferite de contract. (Fig. 3.6.1.a)
  - O **etapă de cercetare și dezvoltare** (R&D stage), care cuprinde **faza de incepție și de elaborare**, de obicei licitată într-o manieră de partajare a riscului, de exemplu, ca un contract Cost Plus Taxa de atribuire (CPAF).
  - O **etapă de producție**, care cuprinde fazele de construcție și de tranziție, de obicei oferită ca un contract cu preț fix ferm (FFP).
  - O **etapă de întreținere**, dacă este necesară, corespunzătoare **fazei de evoluție**, de obicei oferită ca un contract de nivel de efort (LOE).



**Fig. 3.6.1.a.** Procesul Rational pentru dezvoltarea de software de mari dimensiuni

- Caracteristicile proiectelor software de mari dimensiuni:**
  - Se cere de obicei un nivel mai ridicat de vizibilitate din partea clientului asupra evoluției proiectului;
  - Sunt implicate un număr mare de persoane și organizații;
  - Este necesar mai mult **formalism** în proces;
  - Este necesar să se pună mai mult accent pe **artefactele scrise**, decât ar fi în cazul unui proiect mic, intern;

- Toate cele 11 documente descrise sunt absolut necesare în dezvoltare.

### 3.6.2 Procesul Rational pentru un produse software comercial de mici dimensiuni

- O dezvoltare software comercială de mici dimensiuni implică de regulă, un proces mai fluid, cu doar o cantitate mai redusă de formalism și cu un set mai limitat de documente, cum ar fi:
  - (1) O **viziune** asupra produsului;
  - (2) Un **plan de dezvoltare**, care să arate planificarea și resursele;
  - (3) **Documente de descriere a release-urilor**, care specifică pentru fiecare iteratăie scopul acesteia și actualizările realizate pe parcurs pentru a servi drept note ale release-ului la sfârșit;
  - (4) **Documentația utilizator**, după caz;
  - (5) **Arhitectura software, proiectarea software-ului, procesul și procedurile de dezvoltare** pot fi documentate de **codul în sine** sau de mediul de dezvoltare software.

### 3.7. Concluzii

- Procesul **Rational**
  - (1) Pune accent pe abordarea **zonelor cu risc ridicat** foarte timpuriu, prin dezvoltarea rapidă a unei **versiuni inițiale** a sistemului, care definește arhitectura acestuia.
  - (2) **Nu** presupune **un set fix de cerințe ferme la începutul proiectului**, în schimb permite **rafinarea cerințelor** pe măsură ce proiectul evoluează.
  - (3) Se așteaptă și se adaptează la schimbări.
  - (4) Procesul **nu** pune un accent puternic **nici pe documente**, nici pe „ceremonii” și se pretează la automatizarea multora dintre sarcinile obositoare asociate cu dezvoltarea de software.
  - (5) **Obiectivul principal** rămâne **produsul software în sine și calitatea acestuia**, măsurată prin gradul în care își satisfacă utilizatorii finali și își atinge în totalitate obiectivul de rentabilitate a investiției.
  - (6) Un proces derivat din procesul generic descris aici ar fi pe deplin conform cerințelor unui standard precum ISO 9000.

### 3.8 Glossar (lb. engleză)

<b>Artifact</b>	Any document or software other than the software product itself.
<b>Baseline</b>	A release that is subject to change management and configuration control.

<b>Construction</b>	The 3rd phase of the process, where the software is brought from an executable architectural baseline to the point where it is ready to make the transition to its user's community.
<b>Cycle</b>	One complete pass through the 4 phases: inception, elaboration, construction, and transition. The span of time between the beginning of the inception phase and the end of the transition phase.
<b>Elaboration</b>	The 2nd phase of the process where the product vision and its architecture are defined.
<b>Evolution</b>	The life of the software after its initial development cycle; any subsequent cycle, where the product evolves.
<b>Generation</b>	The result of one software development cycle.
<b>High fidelity</b>	Sufficient accuracy (in a cost estimate, quality estimate, or schedule estimate) that a development contractor would commit to a firm, fixed price achievement.
<b>Inception</b>	The first phase of the process, where the seed--idea, RFP, previous generation--is brought up to the point of being (at least internally) founded to enter into the elaboration phase.
<b>Iteration</b>	A distinct sequence of activities with a baselined plan and an evaluation criteria.
<b>Milestone</b>	An event held to formally initiate and/or conclude an iteration.
<b>Phase</b>	The span of time between 2 major milestones of the process where a well defined set of objectives are met, artifacts are completed, and decisions are made to move or not into the next phase.
<b>Product</b>	The software that is the result of the development, and some of the associated artifacts (documentation, release medium, training).

<b>Prototype</b>	A release which is not necessarily subjected to change management and configuration control.
<b>Release</b>	A subset of the end-product which is the object of evaluation at a major milestone (see: prototype, baseline).
<b>Risk</b>	An ongoing or upcoming concern which has a significant probability of adversely affecting the success of major milestones.
<b>Transition</b>	The 4th phase of the process where the software is turned into the hands of the user's community.
<b>Vision</b>	The user's view of the product to be developed.

### 3.9 Acronyms

<b>CPAF</b>	Cost Plus Award Fee
<b>FFP</b>	Firm Fixed Price
<b>LOE</b>	Level Of Effort
<b>OOT</b>	Object-Oriented Technology
<b>RFP</b>	Request for Proposal
<b>ROI</b>	Return On Investment

### Exercițiu #2

- 1) Ce este metoda de dezvoltare Sincronizare și stabilizare? Descrieți fazele acestei abordări.
- 2) Care sunt strategiile de dezvoltare utilizate de Microsoft? Descrieți câteva principii ale fiecărei strategii.
- 3) Descrieți regulile, modul de lucru și echipele în abordarea dezvoltării Sincronizare și stabilizare.

- 4) Descrieți filosofia Oracle de dezvoltare a proiectelor.
- 5) Care sunt responsabilitățile managerului în tehnologia Oracle Development?
- 6) Descrieți fazele proiectului Oracle și procesele proiectului.
- 7) Care sunt perspectivele Procesului Rational? Cum se reconciliază perspectivele?
- 8) Care sunt fazele Procesului Rational? Descrieți conținutul acestora (obiective, rezultate, criterii de intrare și ieșire).
- 9) Care sunt activitățile din Procesului Rational. Descrieți relațiile lor cu fazele Procesului Rational.
- 10) Descrieți artefactele ciclului de viață Procesul Rational.
- 11) Dați câteva exemple de proces rațional (de exemplu, software contractual de mari dimensiuni, software comercial de mici dimensiuni).

## **Capitolul 3. GESTIONAREA PROIECTELOR SOFTWARE**

### **1. Introducere**

- 1.1 Scurt istoric
- 1.2 Definiția managementului de proiect SW
- 1.3 Managementul afacerii
- 1.4 Terminologie
- 1.5 Reguli de bază pentru dezvoltarea unui proiect software

### **2. Contractul**

- 2.1 Cadrul general al unui contract
- 2.2 Tipuri de prețuri

### **3. Drepturile și responsabilitățile clienților**

- 3.1 Cine este Clientul
- 3.2 Parteneriatul Client – Dezvoltator
- 3.3 Drepturile clientului
- 3.4 Responsabilitățile clientului
- 3.5 Semnarea contractului

### **4. Dezvoltarea unui proiect**

- 4.1 Modalități de dezvoltare ale unui proiect
- 4.2 Dezvoltarea unui proiect ideal

### **5. Ciclul de viață al unui proiect**

- 5.1 Model de ciclu de viață al unui proiect software
- 5.1 Documente cheie
- 5.2 Testarea documentelor

### **Exercițiu #3**

## 1. Introducere

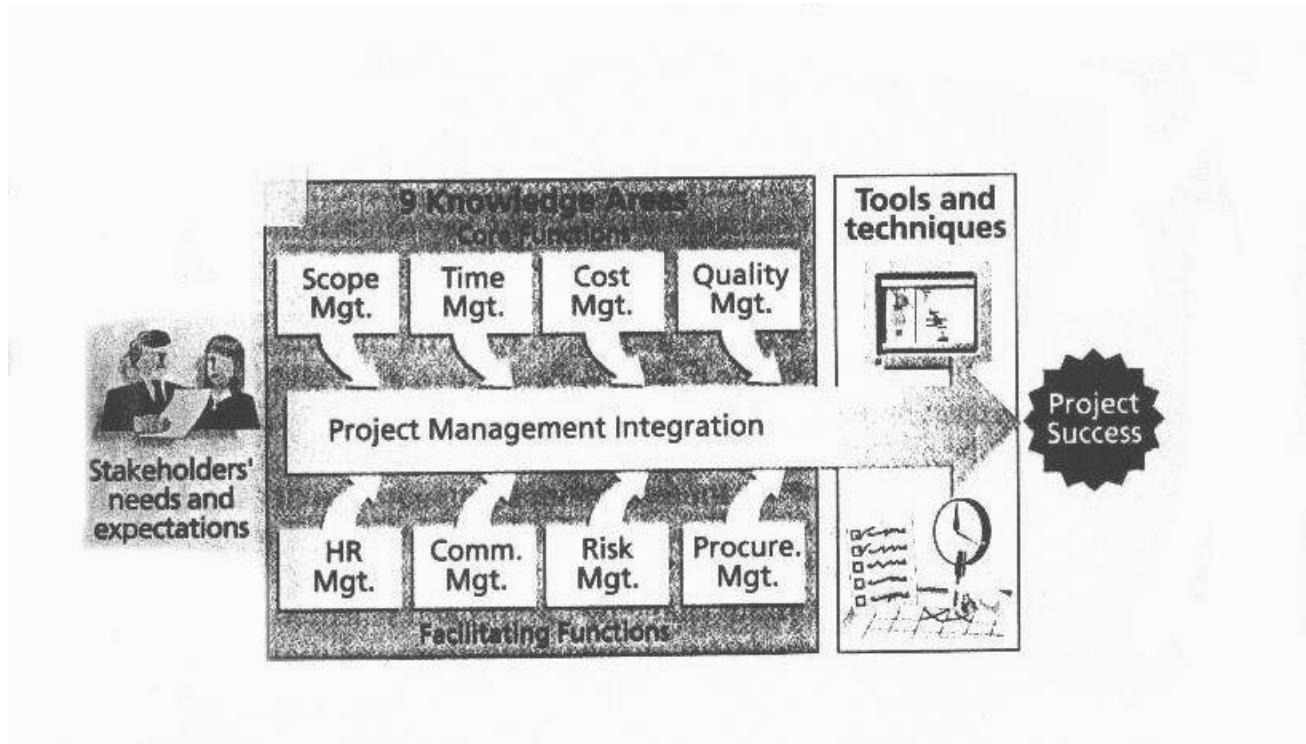
### 1.1 Scurt istoric

- De-a lungul timpului, s-au schimbat multe în domeniul calculatoarelor.
- Calculatoarele au devenit mai mici, mai puternice și din ce în ce mai ieftine.
- Au existat și alte modificări, chiar potențial mai semnificative în impactul pe care calculatoarele l-au avut asupra utilizatorilor, dar nu la fel de vizibile.
- În anii 1970 a existat o activitate febrilă în rândul informaticienilor și al programatorilor practicanți care căutați noi metode de a produce programe bine, la timp și mai eficiente.
- În anii 1980, căutarea continuă, dar multe proiecte sunt deja imersate în unele dintre noile metodologii, cum ar fi programarea structurată și dezvoltarea de sus în jos.
- În acea perioadă, programele pe care te puteai baza, realizate în limitele unui buget convenit și la timp, nu mai sunt o raritate, deși va dura ceva timp până această situație să devină una de zi cu zi.
- Managerii din anii 1980 se confruntă cu unele decizii fundamentale care trebuiau adoptate și care nu au fost atât de evidente în decenile precedente.
- Unele dintre aceste decizii merg la nucleul programării și afectează tot ceea ce face managerul din prima zi de proiect, inclusiv:
  - (1) Cum își organizează oamenii.
  - (2) Ce fel de talente trebuie să găsească.
  - (3) Cum ar trebui să fie planificat timpul de dezvoltare al proiectului.
  - (4) Cum definește un ciclul de dezvoltare pentru proiect.
- Toate acestea au conturat tot mai pregnant necesitatea definirii și implementării conceptului de management al unui proiect software.

### 1.2 Definiția managementului de proiect SW

- Una din multele definiții ale conceptului de management al unui proiect software este următoarea:
- **Definiție: Managementul de proiect** este „*aplicarea cunoștințelor, abilităților, instrumentelor și tehnicilor la activitățile proiectului pentru a satisface sau a depăși nevoile și așteptările părților interesate în proiect*“.
- [K. Schwalbe, „Managementul proiectelor de tehnologie a informației“, Thomson Learning, 2000]
- Această **definiție**:
  - (1) Subliniază utilizarea cunoștințelor și abilităților specifice.
  - (2) Subliniază importanța persoanelor implicate în managementul proiectelor.
- **Managerii** de proiect trebuie:

- (1) Să își dea silința să îndeplinească **obiectivele specifice ale proiectelor** vizând **scopul, timpul, costul și calitatea acestora**.
- (2) Să faciliteze întregul proces pentru a satisface **nevoile și așteptările** persoanelor implicate sau afectate de activitățile proiectului.
- În fig. 1.2.a se prezintă un exemplu de cadru care descrie managementul proiectului. Elementele cheie ale acestui cadru includ:
    - (1) **Părțile interesate** ale proiectului (stakeholders).
    - (2) **Domeniile de cunoștințe** privind managementul proiectelor.
    - (3) **Instrumente și tehnici de management** de proiect.



**Fig. 1.2.a.** Cadru pentru managementul proiectului

- (1) **Părțile interesate** (stakeholders) sunt persoanele implicate sau afectate de activitățile proiectului și includ:
- (1.1) Sponsorul proiectului;
  - (1.2) Echipa de proiect;
  - (1.3) Personalul de sprijin;
  - (1.4) Clienți, utilizatori, furnizori;
  - (1.5) Chiar și opoziții proiectului.
- Activitatea de succes a managerului de proiect presupune implicit dezvoltarea unor **relații cât mai bune** cu **părțile interesate ale proiectului** pentru a se asigura că nevoile și așteptările acestora sunt înțelese și îndeplinite.
- (2) **Zonele de cunoștințe** descriu **competențele cheie** pe care managerii de proiect trebuie să le dezvolte.
- În zona centrală a figurii 1.2.a sunt evidențiate cele **nouă domenii de cunoștințe** ale managementului de proiect. Este vorba despre:

- Patru **domenii de cunoștințe de bază** ale managementului de proiect includ **domeniul de aplicare al proiectului, timpul, costul și managementul calității**.
- Patru **domenii de cunoștințe ajutătoare (facilitatoare)** ale managementului de proiect includ **resursele umane, comunicațiile, riscurile și managementul achizițiilor**.
- **Domeniul de integrare a managementului proiectelor.**
- În continuare se prezintă sintetic conținutul acestora.
  - (2.1) Cele patru **domenii de cunoștințe de bază** conduc la obiective specifice ale proiectului. Mai jos sunt oferite scurte descrieri ale domeniilor de cunoștințe de bază:
    - (2.1.1) **Managementul scopului proiectului** implică definirea și gestionarea tuturor lucrărilor necesare pentru finalizarea cu succes a proiectului.
    - (2.1.2) **Managementul timpului proiectului** include estimarea cât timp va dura finalizarea lucrării, elaborarea unui program acceptabil al proiectului și asigurarea finalizării la timp a proiectului.
    - (2.1.3) **Managementul costurilor proiectului** constă în pregătirea și gestionarea bugetului proiectului.
    - (2.1.4) **Managementul calității proiectului** asigură că proiectul va satisface nevoile declarate sau implicate pentru care a fost întreprins.
  - (2.2) Cele patru **domenii de cunoștințe ajutătoare (facilitatoare)** ale managementului de proiect sunt **resursele umane, comunicațiile, riscurile și managementul achizițiilor**.
    - Acestea se numesc domenii ajutătoare (facilitatoare) deoarece sunt mijloacele prin care se realizează obiectivele proiectului.
    - Mai jos sunt oferite scurte descrieri ale fiecărui domeniu de cunoștințe ajutătoare:
      - (2.2.1) **Managementul resurselor umane** ale proiectului se preocupă de utilizarea eficientă a persoanelor implicate în proiect.
      - (2.2.2) **Managementul comunicațiilor** proiectului implică generarea, colectarea, diseminarea și stocarea informațiilor despre proiect.
      - (2.2.3) **Managementul riscului de proiect** include identificarea, analizarea și răspunsul la riscurile legate de proiect.
      - (2.2.4) **Managementul achizițiilor proiectului** implică achiziționarea sau procurarea de bunuri și servicii care sunt necesare pentru proiect din afara organizației executante.
  - (2.3) Domeniul **Integrarea managementului de proiect** este o funcție integratoare generală care afectează și este afectată de toate celelalte domenii de cunoștințe.
    - Managerii de proiect trebuie să aibă cunoștințe și abilități în toate aceste nouă domenii.
- (3) **Instrumentele și tehnicele de management de proiect** ajută managerii de proiect și echipele lor în realizarea **managementului domeniului, timpului, costurilor și calității**.

- Instrumentele suplimentare ajută managerii de proiect și echipele să efectueze managementul resurselor umane, comunicățiilor, riscurilor, achizițiilor și integrării.
- De exemplu, unele instrumente și tehnici populare de gestionare a timpului includ *diagrame Gantt*, *diagrame de rețea* și *analiza drumului critic*.
- Software-ul de management de proiect este un instrument care poate facilita procesele de management în toate domeniile de cunoaștere.

### 1.3 Managementul ca afacere

- Munca managerului este:
  - (1) Să **planifice** o activitate.
  - (2) Să **se asigure** că ea se realizează.
- Dar din momentul în care proiectul începe, el trebuie să se confrunte cu faptul că **oamenii tind să nu rezolve problemele** până când acestea devin **crize**.
  - În general, doar o criză pare demnă de o atenție reală.
  - Fie că este vorba despre un termen limită pentru o grevă, un impas diplomatic internațional, o nedreptate umană, nimic nu se rezolvă până când ceva debordează.
  - Într-o manieră similară, nici programele de calculator nu primesc o atenție serioasă până când termenul limită este foarte aproape sau clientul amenință să dea în judecată!
  - Există practic o **lege empirică**: „*O problemă trebuie să atingă proporții de criză înainte de a actiona pentru a o rezolva.*”
- Pentru proiectele software există un **scenariu comun**:
  - Timpul trece și problemele se dezvoltă.
  - Deși toată lumea o știe, o simte, starea reală însă este abordată cu mult optimism.
  - Sosește o dată importantă de livrare și nu este nimic de livrat. Panica usoara. Întâlniri. Când va fi livrat articolul, domnule manager? Luna viitoare. Nicio problemă! L-am pus pe Charlie Superprogrammer-ul la treabă.
  - Dar Charlie se dovedește a fi om; luna viitoare sosește și încă niciun produs. Un alt ciclu sau două și vine Vice-Președinte-Companiei în persoană. El va repară lucrurile cu ciocanul lui greu de manager superior. Dar nici asta nu merge.
  - În cele din urmă, se ia o decizie la nivel înalt de a opri, de a face bilanțul și de a veni cu un nou plan pentru finalizare. Un Plan nou? Ah, acolo este problema. Foarte probabil va fi primul plan.
- **Scopul** acestui curs este de a determina **managerul** de proiect software să **planifice** și apoi să-și **controlizeze proiectul** în conformitate cu acel **plan**.
  - (1) Aproape orice **plan** este mai bun decât **niciunul**.
  - (2) **Nu** există altă **alternativă**.

### 1.3 Terminologie

- Înținta cursului de față este **managerul unui proiect software de dimensiuni medii** care implică un număr mediu de programatori, manageri și altele.
  - Proiectele mai mari și mai mici vor fi discutate mai târziu.
- **Majoritatea informațiilor** care vor fi prezentate prezentate sunt **vitale, indiferent de dimensiunea proiectului**.
  - Ceea ce variază de la un job la altul nu este ce sarcini trebuie să îndeplinească managerul, ci de câți cai putere (ce forță de muncă) este nevoie pentru a le îndeplini.
- **Termenul manager** înseamnă lucruri diferite pentru diferite organizații. De obicei, este folosit pentru a identifica două categorii de persoane:
  - (1) **Persoana responsabilă** pentru **planificarea și direcționarea implementării unui job sau a unui proiect**
  - (2) **Persoana responsabilă** pentru **angajări, concedieri, ajustări de salarii și promovări**.
- **Managerul de nivel întâi** este cel care exercită în mod direct supravegherea oamenilor care realizează produsul;
- **Managerul de nivelul doi** supraveghează managerii de nivelul întâi și aşa mai departe.
- În multe organizații, termenul de **supervizor** înlocuiește cu cel de **manager de nivel întâi**. Uneori cele două titluri sunt sinonime, dar:
  - Cel mai adesea termenul **supervizor** are o **conotație tehnică**, de direcționare implicită a implementării, fără a avea responsabilități directă legate de angajare, concediere, salariu și promovări;
  - Dar **supervizorii** au o influență destul de mare în toate acele domenii.
- **Termenii program, proiect software și software** sunt **sinonimi**.
- **Programele operaționale** sunt cele scrise pentru a face job-ul pentru care există proiectul dvs.
  - De exemplu, calculul salariilor, dirijarea zborurilor spațiale, producerea de rapoarte de management, conducerea unui proces, etc.,
- **Programele suport (de sprijin)** sunt cele utilizate ca și ajutorare în realizarea programelor operaționale.
- Pentru a dezvolta un proiect SW, trebuie stabilite câteva reguli de bază.

### 1.4 Reguli de bază pentru dezvoltarea unui proiect software

- Citind literatura de specialitate, sau navigând pe internet, este imposibil să nu fii exasperat de multitudinea de definiții și și de utilizări diverse și adesea contradictorii ale unor termeni precum **software, modul, integrare, test de sistem** și altele.
- Este foarte important ca **toți termenii utilizați să fie definiți clar și fără ambiguități**.

- Este obligatoriu ca toată echipa care participă la dezvoltarea proiectului să vorbească o limbă comună, din punct de vedere managerial.
- În acest sens, cel mai bun lucru pe care îl putem face este să decidem pentru proiectul în dezvoltare următoarele elemente:
  - (1) Care sunt **definițiile** care trebuie utilizate.
  - (2) Care este **schema de management** care trebuie urmată.
- Nu trebuie să creeze probleme faptul că această abordare nu este conformă maniera „corectă” de a face lucrurile sugerată de alții.
  - Nu există o singură cale „corectă”, ci doar alternative dintre care trebuie să alegem.
- **Consistența** în cadrul unui proiect contribuie enorm la succesul acestuia.
- Există câteva **reguli de bază** de stabilit:
  - (1) Trebuie **definit** un **ciclu de dezvoltare al proiectului** la care trebuie raportate toate **planificările și activitățile** desfășurate în acest ciclu.
    - Când menționăm, să zicem, *faza de programare*, trebuie să fim siguri că aceasta înseamnă întotdeauna o **anumită secțiune precizată de timp**, care se poate fi indicată pe o **diagramă simplă** și pentru există întotdeauna anumite **activități asociate** cu acea fază. Nu contează, desigur, ce nume folosim; important este să fim consecvenți.
  - (2) Trebuie **definite** de o manieră **consistentă** **toate activitățile asociate procesului de dezvoltare**, cum ar fi spre exemplu nivelurile de testare.
    - Dacă există un set de definiții universal acceptat, se pot adopta cele care au sens pentru organizația dezvoltatoare. Odată adaptate ele trebuie să devină obligatorii.
  - (3) Trebuie **definit** un **sistem de documente** clar, consecvent și cât mai devreme.
    - Nu trebuie permis niciunui membru al echipei să opereze în afara sistemului respectiv.
    - Vor exista întotdeauna suficiente documente pentru orice proiect, ca atare trebuie evitată durerea de cap produsă de documentele aleatorii, care nu pot fi controlate și a căror autoritate este suspectă.
  - (4) În **concluzie** se recomandă unui **manager de proiect**:
    - (a) Să **definească** un **proces de dezvoltare consistent** pentru **proiectul** pe care îl conduce.
    - (b) Să **creadă** în el.
    - (c) Să îl **vândă** membrilor echipei.
    - (d) Să **impună** aplicarea sa.

## 2. Contractul

- **Contractul** este un **document absolut necesar**. Jumătate din poveștile de groază despre programare implică fie contracte proaste, fie nici un contract.

- Un **contract** este **un acord** între un **dezvoltator** și un **client** prin care se precizează jobul care trebuie realizat în limitele unor constrângeri specifice, pentru o sumă precizată bani.
  - Nu trebuie acționat niciodată în baza unor **acorduri verbale** sau a unor **înțelegeri ocazionale**, chiar dacă clientul se întâmplă să fie un prieten sau un cunoscut din aceeași organizație.
  - Într-o astfel de situație, în cadrul companiei, documentul poate fi numit „**scrisoare de înțelegere**” sau **ceva similar**, care sună mai prietenos decât „contract”, dar el în mod obligatoriu trebuie să existe
- **În orice caz**, este absolut necesară **o declarație formală scrisă** care să arate clar **ce dorește clientul** și **ce este de acord să furnizeze dezvoltatorul**.
  - Operarea fără un astfel de acord este o nebunie pentru ambele părți, aşa cum au aflat mulți manageri de proiecte software și tot atâtia clienti.

## 2.1 Cadrul general al unui contract

- De obicei, un contract trebuie să acopere următoarele elemente esențiale:

### (1) Scopul lucrării.

- **Care** este treaba de făcut?
- Dacă definiția jobului este prea vagă, pot fi încheiate două contracte:
  - Unul pentru a defini jobul.
  - Unul pentru a scrie programe.

### (2) Planificare și livrabile (rezultate).

- **Ce articole specifice** (programe, documente) urmează să fie livrate clientului?
- **Când** urmează să fie livrate?
- **Unde** vor fi livrate?
- **În ce formă** (dischete, CD-uri, drafturi sau documente curate)?
- **Câte exemplare?**

### (3) Oameni cheie.

- **Cine** este autorizat să *aprobe modificările*
- **Cine** este să *accepte* produsul finit?

### (4) Revizuirea planificării.

- **Când și cum** va primi clientul **recenzile și rapoartele despre progres**?
- Care este procedura care se aplică dacă clientului **dezaproba un raport**?

### (5) Procedurile de control a modificărilor.

- Care este mecanismul de rezolvare a modificărilor cerute de client, cu deosebire al acelora care sunt considerate de dezvoltator, modificări ale scopului stabilit inițial?

### (6) Constrângeri de testare.

- Unde și sub controlul cui poate fi obținut timp calculator sau timp de testare pe echipamentele clientului?
- În ce intervale de timp?
- Ce prioritate vor avea testerii dezvoltatorului?

(7) **Criterii de acceptare.**

- Care sunt criteriile cantitative specifice care vor fi utilizate pentru a aprecia dacă produsul finit este **acceptabil**?

(8) **Constrângeri suplimentare.**

- Există elemente care pot fi **specifice mediului de lucru** al dezvoltatorului?
- Dezvoltatorul utilizează **personal al clientului**? Dacă da, ce control are asupra acestora?
- Există probleme speciale de **securitate a datelor**?
- Este obligat clientul să furnizeze date de testare? Dacă da, ce tipuri de date, în ce formă, când și cât de curate?

(9) **Pretul**

- Care este **pretul** dezvoltatorului pentru a face treaba?
- Este **fix** sau **variabil**?
- Dacă este **variabil**, în ce **circumstanțe**?
- Toate aceste elemente și multe altele vor fi abordate mult sau mai puțin detaliat în contract.
- Ultimul articol, **pretul**, este tratat în multe moduri diferite, în funcție de tipul de contract convenit.
- Iată un scurt rezumat al tipurilor de contracte formale uzuale.

## 2.2 Tipuri de prețuri

(1) **Pret fix ferm (Firm Fixed Price) (FFP)**

- Pretul este **stabilit** și **nu** se poate schimba chiar dacă estimarea este proastă.
- Acesta este **cel mai riscant tip de contract** de utilizat într-o job de programare.
- Nu trebuie folosit niciodată fără **o specificație de lucru foarte clară**, care **nu** conține zone neclare și **nici** definiții aproximative.
- Multe dintre proiecte au suferit pierderi grave în cadrul unui astfel de contract.

(2) **Pret fix cu escaladare (Fixed Price with Escalation) (FP-E)**

- Pretul este **stabilit**.
- Se pot face **ajustări atât în sus**, cât și **în jos** în cazul în care se întâmplă anumite lucruri, de exemplu, modificări ale costului forței de muncă, inflație sau modificarea costurilor materialelor.

(3) **Pret fix stimulativ (Fixed Price Incentive) (FPI)**

- Este stabilit un pret **țintă**.

- Sunt stabilite câteva formule care permit:
  - (a) Un **procent mai mare de profit** dacă contractantul depășește obiectivele selectate, (cum ar fi costul).
  - (b) Un **procent mai mic al profitului** (chiar o pierdere) dacă contractantul ratează țintele selectate.

#### (4) Cost partajat (Cost Shared) (CS)

- Acest tip de contract rambursează contractantului o parte sau toate costurile sale, dar **nu permite profit sau comision**. Se poate utiliza:
  - (a) Pentru **activități de cercetare** cu **organizații nonprofit**.
  - (b) În **proiecte comune** (joint projects) între client și dezvoltator, în care se preconizează beneficii pentru contractant. De exemplu, lucrarea poate avea ca rezultat un produs pentru care antreprenorul va avea dreptul exclusiv de vânzare.

#### (5) Cost plus comision de stimulare (Cost Plus Incentive Fee) (CPIF)

- Dezvoltatorului îi se vor plăti **toate costurile** plus o **comision** sau onorariu (fee).
- **Comisionul** variază în funcție de:
  - (a) Cât de aproape este contractantul de **îndeplinirea costurilor țintă stabilită**.
  - (b) Cât de bine a soluționat problemele domeniilor specificate în contract.
- Criteriile care determină **comisionul** trebuie să fie **obiective și măsurabile**.

#### (6) Cost plus comision de merit (Cost Plus Award Fee) (CPAF).

- Este similar cu CPIF.
- Diferența constă în criteriile care sunt mai subiective și sunt canticări de un Comitet de evaluare.

#### (7) Cost plus comision fix (Cost Plus Fixed Fee) (CPFF)

- Contractorului îi se plătesc **costurile admisibile** și un **comision stabilit**.

#### (8) Timp și materiale (Time & Materials) (T&M)

- Dezvoltatorul este plătit pentru **orele de muncă efectiv lucrate** și pentru **costul materialelor utilizate**.

#### (9) Oră de muncă (Labor hour) (LH)

- Sunt platite doar **orele de muncă**, dar nimic altceva.

#### (10) Nivel de efort (Level of Effort) (LOF)

- Este similar cu tipul de contract ore de muncă.
- Este plătit **efortul**, dar nimic altceva.

- **Concluzii:**

- (1) Ultimele două tipuri de contract (Ore de muncă și Nivel de efort) sunt aproape **fără riscuri** pentru antreprenor. El oferă posibilitatea de a acționa așa cum cere clientul.

- (2) Celelalte tipuri de contracte implică **grade diferite de risc** pentru antreprenor și client.
- (3) Atunci când produsul livrabil poate fi bine definit în prealabil, dezvoltatorul poate propune un **preț fix** și un **onorariu ridicat**.
- (4) Atunci când produsul final este prost definit sau supus modificărilor, din punctul de vedere al contractantului, un **contract de tip cost** este cel mai adecvat. Profitul lui este mai mic, dar și riscul.

### **3. Drepturile și responsabilitățile clientilor**

- Succesul software-ului depinde de dezvoltarea unui **parteneriat de colaborare** între **dezvoltatorii** de software și **clientii acestora**.
  - De prea multe ori, însă, relația client-dezvoltator devine **tensionată** sau chiar **contradictorie**.
- Problemele apar parțial deoarece oamenii nu împărtășesc **o înțelegere clară a cerințelor și cine sunt cu adevărat clientii**.
- Pentru a clarifica aspectele cheie ale parteneriatului client-dezvoltator, Karl Wiegers [\*] propune două documente:
  - (1) **Lista de cerințe referitoare la drepturile clientilor software**.
  - (2) **Lista de cerințe referitoare la obligațiile clientilor software**.

- Deoarece este imposibil să fie identificată fiecare cerință la începutul unui proiect, practica frecvent utilizată – și uneori abuzată – de **înghețare prin semnătură** a cerințelor proiectului necesită o examinare suplimentară.
- 

#### **3.1 Cine este clientul?**

- Un **client** este orice persoană care obține beneficii directe sau indirecte dintr-un produs. Aceasta include:
  - (1) Persoane care solicită, plătesc, selectează, specifică sau utilizează un produs software sau
  - (2) Cei care sunt beneficiarii produsului software.

- Există diferite tipuri de clienti:

##### **(1) Clientii care inițiază sau finanțează un proiect software.**

- Furnizează conceptul de produs la nivel înalt și rațiunea de afaceri a proiectului.
- Aceste cerințe de afaceri descriu valoarea pe care utilizatorii, organizația dezvoltatoare sau alte părțile interesate (stakeholders) doresc să o primească de la sistem.

##### **(2) Clientii care vor folosi efectiv produsul.** Sunt de fapt **utilizatorii** care pot descrie:

- Cerințele pe care trebuie să le îndeplinească (cazurile de utilizare) cu produsul.
- Atributele de calitate dorite ale produsului.

- **Analistii** interacționează cu **clientii** pentru a colecta și documenta cerințele și pentru a deriva caracteristicile funcționale specifice ale software-ului pornind de la cerințele utilizatorului.
- Există diferite scenarii posibile:
  - **Clienții nu au timp** să participe la stabilirea cerințelor.
  - **Clienții se așteaptă** ca dezvoltatorii să-și dea seama de ce au nevoie utilizatorii fără prea multe discuții și documentare.
  - Grupurile de dezvoltare **exclud** uneori **clientii** din activitățile de determinare a cerințelor, apreciind că ei știu mai bine să facă acest lucru, că vor economisi timp sau că ar putea pierde controlul asupra proiectului prin implicarea altora.
  - Utilizarea **clientilor** pentru a **răspunde la întrebări** sau pentru a oferi **feedback selectat** după ce produsul este dezvoltat.
  - **Clienții proactivi** vor insista să fie parteneri în acțiune.

(3) Pentru **dezvoltarea de software comercial, clientul și utilizatorul** sunt adesea aceeași persoană.

- Chiar și pentru **software-ul comercial** ar trebui însă să fie implicați utilizatorii reali în procesul de colectare a cerințelor. Acest lucru se poate realiza prin grupuri focus sau apelând pe relațiile existente de testare beta.

(4) **Client surogat** cum ar fi departamentul de marketing care încearcă să determine ceea ce **clientii reali** ar găsi atrăgător.

### **3.2 Parteneriatul client-dezvoltator**

- **Software-ul de calitate** este produsul unui **design bine executat** bazat pe **cerințe precise**, care sunt la rândul lor rezultatul unei **comunicări și colaborări eficiente**, un **parteneriat real între dezvoltatori și clienți**.
- **Eforturile de colaborare** funcționează numai atunci când:
  - (1) Toate părțile implicate **știu** de ce au nevoie pentru a avea **succes**.
  - (2) Ei **înțeleg și respectă** ceea ce **au nevoie** colaboratorii lor pentru a **reuși**.
- Pe măsură ce presiunile proiectelor cresc, este ușor să uiți că toată lumea împărtășește un obiectiv comun:
- Pentru a construi un produs de succes care să ofere valoare pentru afaceri, satisfacție utilizatorilor și împlinirea dezvoltatorului.
- Karl Wiegers propune o **listă de drepturi ale clientului software** (Figura 3.2.a) [Wi99].
- În virtutea acestor drepturi există de fapt așteptări pe care clienții le pot plasa în interacțiunile lor cu analiștii și dezvoltatorii în timpul ingineriei cerințelor.
- Fiecare dintre aceste drepturi implică responsabilitatea corespunzătoare a dezvoltatorului de software.

## Drepturile clientului software

1. Să se aștepte ca analiștii să îi vorbească limba adică să fie familiarizat cu limbajul afacerii.
2. Să se aștepte ca analiștii să învețe despre afacerea sa și despre obiectivele sale referitoare la software.
3. Să se aștepte ca analiștii să structureze informațiile despre cerințele formulate într-o specificație a cerințelor software.
4. Să se aștepte ca dezvoltatorii să explice cerințele produselor în lucru.
5. Să se aștepte ca dezvoltatorii să îi trateze cu respect și să mențină o atitudine profesională.
6. Să se aștepte ca analiștii să le prezinte idei și alternative atât pentru cerințe cât și pentru implementare.
7. Să-i fie descrise acele caracteristicile care vor face produsul ușor și mai placut de utilizat.
8. Să primească oportunități și sugestii de a-și ajusta cerințele pentru a permite reutilizarea.
9. Să primească estimări cu bună-credință ale costurilor și compromisurilor atunci când solicită o modificare.
10. Să primească un sistem care să răspundă nevoilor funcționale și de calitate convenite cu dezvoltatorul.

**Figura 3.2.a.** Lista a drepturilor clienților software

- Karl Wiegers propune, de asemenea, 10 responsabilități pe care clientul le are față de dezvoltator în timpul procesului stabilire a cerințelor (Figura 3.2.b) [Wi99].
- Aceste drepturi și responsabilități se aplică și reprezentanților actuali ai utilizatorilor pentru dezvoltarea software-ului corporativ intern.
- Pentru dezvoltarea de produse pe piața de masă, acestea se aplică mai mult surogatelor clienților, cum ar fi departamentele de marketing.

## Responsabilitățile clientului software

1. Să educe analiștii cu privire la afacerea personală și să-și definească jargonul.
2. Să aloce timp pentru a stabili cerințele, pentru a le clarifica și pentru a le concretiza în mod iterativ.
3. Să fie specific și precis cu privire la cerințele sistemului.
4. Să ia decizii în timp util cu privire la cerințe atunci când se solicită acest lucru.
5. Să respecte evaluările dezvoltatorilor cu privire la cost și fezabilitate.

6. Să stabilească priorități pentru cerințele individuale, pentru caracteristicile sistemului sau pentru cazurile de utilizare.
7. Să examineze și să revizuiască cu atenție documentele și prototipurile.
8. Să comunice cu promptitudine modificările la cerințele produsului.
9. Să se conformeze procesului de control al modificărilor definit de organizația de dezvoltare.
10. Să respecte cerințele proceselor de inginerie pe care le folosește dezvoltatorul.

**Figura 3.2.b.** Lista responsabilităților clienților software

- La începutul proiectului, reprezentanții clientului și al dezvoltatorului trebuie să revizuiască aceste două liste și să ajungă la un consens.
- Dacă apar anumite probleme sau neclarități, acestea trebuie negociate pentru a se ajunge la o înțelegere clară cu privire la responsabilitățile reciproce.
- Această înțelegere poate reduce semnificativ divergențele de mai târziu, când una dintre părți se așteaptă la ceva ce cealaltă parte nu este dispusă sau capabilă să ofere.
- Aceste liste nu sunt axhaustive, astfel încât ele pot fi oricând modificate pentru să satisfacă nevoile specifice.

### 3.3 Drepturile clientului

- **Dreptul #1:** Să se aștepte ca analiștii să îi vorbească limba adică să fie familiarizat cu limbajul afacerii.
  - Discuțiile privind **cerințele** ar trebui să se concentreze pe **nevoile și afacerile clientului**, utilizând **vocabularul de afaceri al clientului** (pe care ar putea fi nevoie să îl transmită analiștilor).
  - Clientul nu ar trebui să fie nevoie să utilizeze **jargonul informatic**.
- **Dreptul #2:** Să se aștepte ca analiștii să îi vorbească limba adică să fie familiarizați cu limbajul afacerii.
  - Prin **interacțiunea** cu utilizatorii în timp ce stabilesc cerințele, analiștii pot înțelege mai bine sarcinile de afaceri ale clienților și modul în care produsul se potrivește cu necesitățile lor.
  - Acest lucru va ajuta dezvoltatorii să proiecteze software care **să răspundă** cu adevărat **nevoilor** **clienților**.
  - Trebuie luată în considerare **invitarea dezvoltatorilor sau analiștilor** pentru ca **să observe ce fac** **clienții** **la locul de muncă**.
  - Dacă noul sistem înlocuiește o aplicație existentă, dezvoltatorii ar trebui să folosească sistemul în calitate client, pentru a vedea cum funcționează, cum se încadrează în fluxul său de lucru și unde poate fi îmbunătățit.

- **Dreptul #3:** Să se aștepte ca analiștii să structureze informațiile despre cerințele formulate într-o specificație a cerințelor software.
  - Analistul va sorta informațiile furnizate de client, separând nevoile reale ale utilizatorilor de alte elemente, cum ar fi:
    - Cerințe și reguli de afaceri;
    - Cerințe funcționale;
    - Obiective de calitate;
    - Idei de soluții.
  - Analistul va scrie apoi o specificație structurată pentru cerințele software, care constituie un acord între dezvoltator și client cu privire la produsul propus.
  - Clientul trebuie să revizuiască aceste specificații pentru a se asigura că reprezintă exact și complet cerințele sale.
- **Dreptul #4:** Să se aștepte ca dezvoltatorii să explice cerințele produselor în lucru.
  - Analistul poate reprezenta cerințele folosind diferite diagrame care completează documentul de specificare scris.
  - Aceste reprezentări grafice ale cerințelor exprimă anumite aspecte ale comportamentului sistemului mai clar decât pot cuvintele.
  - Deși nu sunt familiare, diagramele nu sunt greu de înțeles.
  - Analistii ar trebui:
    - (1) Să explicce scopul fiecărei diagrame.
    - (2) Să descrie notațiile folosite.
    - (3) Să demonstreze cum pot fi determinate erorile.
- **Dreptul #5:** Să se aștepte ca dezvoltatorii să îl trateze cu respect și să mențină o atitudine profesională
  - Discuțiile despre cerințe pot fi frustrante dacă utilizatorii și dezvoltatorii nu se înțeleg.
  - Lucrul împreună poate deschide ochii fiecărui grup asupra problemelor cu care se confruntă celălalt.
  - Clienții care participă la dezvoltarea cerințelor au dreptul ca dezvoltatorii să îl trateze cu respect și să aprecieze timpul pe care îl investesc în succesul proiectului.
  - În mod similar, clientul trebuie să demonstreze respect pentru dezvoltatori în timp ce aceștia lucrează cu el în vederea atingerii obiectivului lor comun adică un proiect de succes.
- **Dreptul #6:** Să se aștepte ca analiștii să le prezinte idei și alternative atât pentru cerințe cât și pentru implementare.
  - Analistii ar trebui să exploreze modalități în care sistemele existente ale clienților nu se potrivesc bine cu procesele sale actuale de afaceri, pentru a se asigura că noul produs nu automatizează procesele ineficiente sau ineficiente.
  - Analistii care înțeleg temeinic domeniul aplicației pot sugera uneori îmbunătățiri în procesele de afaceri ale clienților.

- Un analist experimentat și creativ adaugă, de asemenea, valoare prin propunerea de capabilități valoroase pe care noul software le-ar putea oferi, capabilități pe care utilizatorii nici măcar nu și le-au imaginat.
- **Dreptul #7:** Să-i fie descrise acele caracteristicile care vor face produsul mai ușor și mai plăcut de utilizat.
  - Analistul ar trebui să informeze clientul despre caracteristicile software-ului care depășesc nevoile funcționale.
  - Aceste caracteristici numite și „attribute de calitate”, fac de regulă software-ul mai ușor sau mai plăcut de utilizat, permitând utilizatorului să-și îndeplinească sarcinile în mod precis și eficient.
  - De exemplu, clienții afirmă uneori că produsul trebuie să fie „foarte ușor de utilizat” sau „robust” sau „eficient”, dar acești termeni sunt de regulă, atât subiectivi cât și vagi.
  - Analistul ar trebui să explice și să documenteze caracteristicile specifice care înseamnă „prietenos”, „robust” sau „eficient” pentru utilizator.
- **Dreptul #8:** Să primească oportunități și sugestii de a-și ajusta cerințele pentru a permite reutilizarea.
  - Analistul poate cunoaște componentele software deja existente care sunt aproape de a răspunde unor nevoi descrise de client.
  - Într-un astfel de caz, analistul ar trebui să ofere clientului șansa de a-și modifica cerințele pentru a permite dezvoltatorilor să refolosească software-ul existent.
  - Ajustarea cerințelor clienților atunci când sunt disponibile oportunități sensibile de reutilizare, ar putea economisi timpul care de altfel ar fi necesar pentru a dezvolta exact ceea ce a fost specificat în cerințele originale.
- **Dreptul #9:** Să primească estimări de bună-credință ale costurilor și compromisurilor atunci când solicită o modificare.
  - Oamenii fac uneori alegeri diferite când știu că o alternativă este mai scumpă decât alta.
  - Estimarea impactului și costului unei propuneri de modificare de cerință sunt absolut necesare pentru putea lua decizii de afaceri bune cu privire la care dintre modificările solicitate să fie aprobate.
  - Dezvoltatorii ar trebui să prezinte cele mai bune estimări ale impactului, costurilor și compromisurilor care nu vor fi întotdeauna ceea ce își dorește clientul să audă.
  - Dezvoltatorii nu trebuie să mărească costul estimat al unei modificări propuse doar pentru că nu doresc să o implementeze.
- **Dreptul #10:** Să primească un sistem care să răspundă nevoilor funcționale și de calitate convenite cu dezvoltatorul.
- **Concluzie.** Proiectul software dorit este obținut numai dacă:
  - (1) **Clientul** comunică în mod clar **toate informațiile** care vor permite dezvoltatorilor să construiască produsul care îi satisfacă nevoile.
  - (2) **Dezvoltatorii** comunică **opțiunile și constrângerile**.

(3) **Clientul** trebuie să comunice orice fel de ipoteze sau de așteptări implicate pe care le-ar putea avea referitoare la produs, altfel, dezvoltatorii probabil nu le vor putea adresa spre a satisface așteptările clientilor.

### 3.3 Responsabilitățile clientului

- **Responsabilitatea #1:** Să educe analiștii cu privire la afacerea personală și să-și definească jargonul.
  - Analisii depind de client pentru a-i educa despre concepțele și terminologia lui de afaceri.
  - Intenția nu este de a transforma analisii în experți de domeniu, ci de a-i ajuta să înțeleagă problemele și obiectivele clientilor.
  - Clientul nu trebuie să se aștepte ca analisii să aibă cunoștințe pe care el și colegii săi le consideră de la sine înțelese.
- **Responsabilitatea #2:** Să aloce timp pentru a stabili cerințele, pentru a le clarifica și pentru a le concretiza în mod iterativ.
  - Clientul are responsabilitatea de a investi timp în ateliere, interviuri și alte activități de elicitație a cerințelor.
  - Uneori, analistul poate crede că înțelege un punct spus de client, pentru a realiza mai târziu că are nevoie de clarificări suplimentare.
  - Clientul trebuie să aibă răbdare cu această abordare iterativă pentru dezvoltarea și rafinarea cerințelor, deoarece este natura comunicării umane complexe și esențială pentru succesul software-ului.
- **Responsabilitatea #3:** Să fie specific și precis cu privire la cerințele sistemului.
  - Este foarte dificil să se redacteze cerințe clare și precise.
  - Clientul este uneori tentat să formuleze cerințe vagi, deoarece stabilirea detaliilor este plăcătoare și necesită timp.
  - La un moment dat în timpul dezvoltării, totuși, cineva trebuie să rezolve ambiguitățile și impreciziile.
  - Clientul este probabil cea mai bună persoană pentru a lua acele decizii. În caz contrar, se bazează pe abilitatea de a ghici corect a dezvoltatorilor.
  - Clientul trebuie să facă tot posibilul pentru a clarifica intenția fiecărei cerințe, astfel încât analistul să o poată exprima cu acuratețe în specificația cerințelor software.
  - Dacă clientul nu poate fi precis, trebuie să fie de acord cu un proces de generare a preciziei necesare, eventual prin prototipare.
- **Responsabilitatea #4:** Să ia decizii în timp util cu privire la cerințe atunci când se solicită acest lucru.
  - Analistul va cere clientului să facă multe alegeri și decizii.
  - Aceste decizii includ rezolvarea solicitărilor inconsecvente primite de la mai mulți utilizatori și realizarea de compromisuri între atributile de calitate conflictuale.
  - Clientii care sunt autorizați să ia astfel de decizii trebuie să facă acest lucru prompt atunci când sunt solicitați.

- De multe ori, dezvoltatorii nu pot continua până când clientul nu ia decizia, astfel încât timpul petrecut în aşteptarea unui răspuns poate întârzi progresul.
- Dacă deciziile clientului nu sunt la îndemâna, dezvoltatorii ar putea lua deciziile în locul lui pentru a putea avansa, ceea ce adesea nu va duce la rezultatul pe care îl preferă clientul.
- **Responsabilitatea #5:** Să respecte evaluările dezvoltatorilor cu privire la cost și fezabilitate.
  - Toate funcțiile software au un preț, iar dezvoltatorii sunt în cea mai bună poziție pentru a estima aceste costuri.
  - Unele caracteristici pe care clientul le-ar dori ar putea să nu fie fezabile din punct de vedere tehnic sau ar putea fi surprinzător de costisitor de implementat.
  - Dezvoltatorul poate fi purtător de vești proaste referitor la fezabilitate sau cost, iar clientul ar trebui să respecte această judecată.
  - Uneori, clientul poate rescrie cerințele într-un mod care să le facă fezabile sau mai ieftine.
  - De exemplu, a cere ca o acțiune să aibă loc „instantaneu” nu este fezabilă, dar o cerință de sincronizare mai specifică („în 50 de milisecunde”) ar putea fi realizabilă.
- **Responsabilitatea #6:** Să stabilească priorități pentru cerințele individuale, pentru caracteristicile sistemului sau pentru cazurile de utilizare.
  - Majoritatea proiectelor nu au timpul sau resursele pentru a implementa fiecare parte dorită de funcționalitate, astfel încât clientul trebuie să stabilească:
    - (1) Ce caracteristici sunt esențiale.
    - (2) Care sunt importante de încorporat eventual.
    - (3) Ce ar fi doar un vis frumos.
  - De obicei, dezvoltatorii nu pot determina prioritățile din perspectiva clientului, dar ar trebui să estimeze costul și riscul tehnic al fiecărei caracteristici, caz de utilizare sau cerință pentru a ajuta clientul să ia decizia.
  - Când clientul acordă priorități, el îi ajută pe dezvoltatori să ofere cea mai mare valoare la cel mai mic cost.
  - Nimănui nu-i place să audă că ceva ce își dorește nu poate fi finalizat în limitele proiectului, dar aceasta este de fapt realitatea.
  - Uneori mai trebuie luată o decizie de afaceri pentru a reduce scopul bazat pe priorități al proiectului, sau pentru a extinde planificarea, pentru a oferi resurse suplimentare sau pentru a face compromisuri cu privire la calitate.
- **Responsabilitatea #7:** Să examineze și să revizuiască cu atenție documentele și prototipurile.
  - Participarea clienților la recenzii formale și informale este o activitate valoroasă de control al calității.
  - Este singura modalitate de a evalua dacă cerințele sunt complete, corecte și necesare.

- Este dificil de imaginat cum va funcționa de fapt software-ul citind o specificație.
- Pentru a înțelege mai bine nevoile clientilor și pentru a explora cele mai bune modalități de a le satisface, dezvoltatorii construiesc adesea prototipuri.
- Feedback-ul clientilor cu privire la aceste implementări preliminare, parțiale sau posibile, ajută la asigurarea faptului că toată lumea înțelege cerințele.
- Cu toate acestea, un prototip nu este un produs final.
- Dezvoltatorii trebuie să se străduiască să construiască sisteme pe deplin funcționale bazate pe prototip.
- **Responsabilitatea #8:** Să comunice cu promptitudine modificările la cerințele produsului.
  - Cerințele în continuă schimbare reprezintă un risc serios pentru capacitatea echipei de dezvoltare de a livra un produs de înaltă calitate în programul planificat.
  - Schimbarea este inevitabilă, dar cu cât o schimbare este introdusă mai târziu în ciclul de dezvoltare, cu atât impactul acesteia este mai mare.
  - Modificările extinse ale cerințelor indică adesea că procesul inițial de elicitație a cerințelor nu a fost adecvat.
  - Modificările pot cauza reprelucrari costisitoare și planificările se pot altera dacă se solicită o nouă funcționalitate în timp ce ce construcția este în plină desfășurare.
  - Clientul trebuie să notifice analistul cu care lucrează de îndată ce devine conștient de orice modificare necesară a cerințelor.
  - Clientii cheie ar trebui, de asemenea, să participe la procesul de decizie în care se aprobă sau se resping cererile de modificare.
- **Responsabilitatea #9:** Să se conformeze procesului de control al modificărilor definit de organizația de dezvoltare.
  - Pentru a minimiza impactul negativ al schimbării, toți participanții trebuie să urmeze procesul de control al schimbării al proiectului. Acest lucru asigură că:
    - (1) Modificările solicitate nu se pierd.
    - (2) Impactul fiecărei modificări solicitate este evaluat.
    - (3) Toate modificările propuse sunt luate în considerare în mod consecvent.
    - (4) Ca urmare, clientul poate lua decizii de afaceri bune pentru a încorpora anumite modificări în produs.
- **Responsabilitatea #10:** Să respecte cerințele proceselor de inginerie pe care le folosește dezvoltatorul.
  - Colectarea cerințelor și verificarea acurateței acestora sunt printre cele mai mari provocări în dezvoltarea de software.
  - Deși clientul ar putea deveni frustrat de proces, este o investiție excelentă care va fi mai puțin dureroasă dacă clientul înțelege și

respectă tehniciile folosite de analiști pentru a colecta, documenta și asigura calitatea cerințelor software.

- Clienții ar trebui să fie educați cu privire la procesul de cerințe, în mod ideal, participând la cursuri împreună cu dezvoltatorii.
- O modalitate eficientă este prezentarea de seminarii unui public care include dezvoltatori, utilizatori, manageri și specialiști în cerințe.
- Oamenii pot colabora mai eficient atunci când învață împreună.

### 3.4 Semnarea contractului

- A conveni asupra **cerințelor unui produs nou** este o parte **critică** a parteneriatului client-dezvoltator.
- Multe **organizații** folosesc actul de a semna **documentul de cerințe** pentru a indica **aprobarea clientului**.
- Toți participanții la procesul de aprobare a cerințelor trebuie însă să știe exact ce înseamnă semnarea documentului.

(1) O problemă potențială este reprezentantul clientului care consideră semnarea cerințelor ca pe un **ritual lipsit de sens**:

- „*Mi s-a dat o bucătă de hârtie care avea numele meu tipărit sub o linie în partea de jos, aşa că am semnat pentru că altfel dezvoltatorii nu ar fi început codificarea.*”
- Această atitudine poate duce la **conflicte viitoare** atunci când acel client doresc să modifice cerințele sau când este surprins de ceea ce este livrat:
- *"Sigur, am semnat cerințele, dar nu am avut timp să le citesc pe toate. Am avut încredere în voi băieți – și voi m-ați dezamăgit!"*

(2) La fel de problematic este managerul de dezvoltare care vede **semnarea** ca o modalitate de a **îngheța cerințele**.

- Ori de câte ori este prezentată o cerere de modificare, managerul de dezvoltare poate indica specificația cerințelor software și poate protesta:
  - *"Ați semnat aceste cerințe, aşa că asta construim. Dacă ați vrut altceva, ar fi trebuit să spuneți asta înainte"*
- Ambele abordări omit să recunoască realitatea că de fapt:
  - (1) Este **imposibil** să fie cunoascate **toate cerințele** la **începutul proiectului**.
  - (2) **Cerințele se vor schimba**, fără îndoială, **în timp**.
- Semnarea cerințelor este o acțiune care închide oarecum procesul de dezvoltare a cerințelor.
- Cu toate acestea, participanții trebuie să înțeleagă și fie de acord cu exact ceea ce prespun semnăturile lor.
- Mai important decât **ritualul de semnare** este **conceptul** de a stabili o „linie de bază” a acordului pe cerințe, de fapt, un **instantaneu** al acestora la momentul semnării documentului.
- Subtextul unei semnături pe o pagină de semnare a specificațiilor de cerințe software ar trebui, prin urmare, să se citească ceva în cheia:

- (1) Sunt de acord că acest document reprezintă **cea mai bună înțelegere** a cerințelor pentru proiectul la **momentul de astăzi**.
- (2) **Modificările viitoare ale acestei linii de bază pot fi făcute** prin procesul de modificare definit în proiect.
- (3) Îmi dau seama că modificările aprobată ar putea necesita să **renegociem costurile, resursele și angajamentele de planificare** ale proiectului.
- O **înțelegere comună** a procesului de aprobată a cerințelor ar trebui să atenuze fricțiunile care pot să apară pe măsură ce proiectul avansează și sunt dezvăluite limitări ale cerințelor sau pe măsură ce piata și cerințele de afaceri evoluează.
- **Sigilarea activităților inițiale de dezvoltare a cerințelor** cu un astfel de acord explicit ajută enorm consolidarea unui **parteneriat continuu client-dezvoltator** pe drumul succesului proiectului.

## **4. Dezvoltarea unui proiect**

### **4.1 Modalități de dezvoltare ale unui proiect**

- În mod tradițional, de-a lungul timpului, majoritatea programelor au fost analizate și proiectate de **sus în jos** și codificate și testate de **jos în sus**.
- În timpul analizei și proiectării, a părut firesc să se înceapă prin a lua în considerare **mai întâi sistemul în ansamblu** și apoi **descompunerea sa în bucăți** din ce în ce mai mici pe care indivizi să le poată gestiona.
- Apoi aceste bucăți sunt codificate, testate și combinate („integrate”) în grupări din ce în ce mai mari și mai complexe, până când în cele din urmă întregul sistem a fost asamblat de **jos în sus**.
- Multe sisteme sunt încă construite în acest fel. Dar tendința modernă este către o **dezvoltare completă de sus în jos**, în care nu numai analiza și designul sunt atacate de sus, ci și codificarea și testarea de integrare.

### **4.2 Dezvoltarea unui proiect ideal**

- Paragraful de față descrie pe scurt un **proiect bine condus și de succes**, din păcate, nu atât de tipic în zilele noastre.

#### **(1) Etapa propunerii**

- Un proiect de programare începe cu o **idee** pe care un utilizator o are cu privire la o nevoie pe care un sistem de calcul o poate gestiona.
- Utilizatorul solicită **idei** de la **asociați și contractanți** despre caracterul rezonabil al ideii și posibilele îmbunătățiri.
- După unele incubății și revizuirii, **ideea** acum mai **fermă** este prezentată oficial, de obicei contractorilor aflați în competiție pentru oferte, de regulă în forma unui **caiet de sarcini**.
- Concurenții încep o activitate febrilă numită **scrierea propunerii**.
  - Fiecare încearcă să-și dea seama cum poate satisface nevoile acestui utilizator la un cost mai mic și de o calitate mai bună decât este probabil să propună ceilalți.

- Fiecare redactează un **document** despre modul în care a **înțeles problema** și cum ar rezolva-o printr-un **program de calculator**.
- Fiecare adaugă un nivel de **realizări și aprecieri personale laudative** pentru a încerca să impresioneze clientul cu acreditările sale.
- Propunerile sunt înaintate spre **evaluare** într-un **proces de licitație**.
- Dintre concurenții considerați receptivi la nevoile sale, clientul **selectează unul**, de obicei **cel mai jos ofertant**, pentru a face treaba.
- Dacă niciunul dintr ofertanți nu este considerat suficient de credibil, clientul redefineste cerințele și solicită noi propunerii.
- Câștigătorul își sărbătorește norocul în timp ce învinșii aplaudă, iar proiectul începe.
- Câștigătorul numește un **manager de proiect** care organizează o **echipă** (partial gata de când propunerile au fost depuse pentru prima dată) pentru a face acest lucru.

## (2) Definire, Planificare

- Echipa constiuță abordează două sarcini imediate:
  - (1) **Definirea** la nivel de detaliu a **nevoilor clientului**.
  - (2) **Redactarea** unui **plan** pentru satisfacerea acestor nevoi.
- Ambele sarcini au fost parțial „realizate” în etapa de propunere, dar acum trebuie să fie perfecționate.
- Trebuie scrisă o **specificație a problemei** foarte precisă, structurată sau un **document cu specificațiile cerințelor**, care să servească drept **bază pentru proiectarea și planificarea ulterioară**.
- Trebuie de asemenea redactat un **plan detaliat al proiectului**, pentru a ghida toate fazele de dezvoltare ale proiectului.

## (3) Selectarea echipei, Proiectarea

- Acum, Managerul de Proiect (MP) trebuie să recruteze și să organizeze talentele necesare pentru următoarea fază: **proiectarea sistemului de programe**.
- MP selectează **cei mai buni designeri** pe care îi are la dispoziție, inclusiv pe unii dintre analiști și îndrumă să proiecteze **cea mai bună arhitectură posibilă pentru sistemul de programe**, care să se potrivească cu problema definită de analiști.
- În timp ce proiectarea continuă, managerul de proiect este ocupat cu **recrutarea de personal** și găsirea altor resurse pentru munca rămasă de făcut.
- MP ține ochii pe planul proiectului și ia măsuri pentru a îndeplini toate etapele de referință menționate în acesta.
- Uneori, MP este nevoie să schimbe planul și o va face.
- (4) **Aprobarea specificației**
- Când **proiectarea generală a sistemului de programe** este gata:

- Designul este revizuit și aprobat de managementul de proiect și de client.
- Este stabilit ca bază pentru proiectarea și codificarea detaliată.
- Este predat programatorilor.

#### (5) Proiectarea detaliată, programarea, integrarea

- Programatorii rafinează și mai mult design-ul liniei de bază în bucăți mai mici, până când perfecționările ating nivelul codului real.
- Pieșele („modulele”) sunt codificate și testate apoi îmbinate cu atenție („integrate”) una cu cealaltă într-o manieră preplanificată.
- Pe măsură ce modulele sunt adăugate cu succes, sistemul de programe crește în complexitate și utilitate.
  - Se poate ajunge eventual la un anumit număr de versiuni prin care, la anumite momente de timp, se poate demonstra îndeplinirea unui anumit subset al funcțiilor propuse.
- Deoarece integritatea designului a fost planificată în fazele anterioare, sistemul în ansamblul său se potrivește bine.
- După toate probabilitățile vor exista greșeli de analiză, proiectare sau codare.
- Modificările vor fi făcute după cum este necesar, dar ele vor fi strict controlate printr-un mecanism de control simplu, planificat și dezvoltat din timp.

#### (6) Documentare, Testare

- În cele din urmă, sistemul de programe este pregătit pentru client, împreună cu setul său de documentație descriptivă și un set de documentație pentru utilizator.
- Dar produsul nu ajunge încă în mâna clientului.
  - Mai întâi este trecut printr-un alt set de teste numite „teste de sistem”.
  - Pentru a asigura integritatea și obiectivitatea, aceste teste sunt concepute și conduse de un grup separat, mai degrabă decât de programatori.
  - Acest grup își imaginează utilizatorul și încearcă să „creeze iadul” cu sistemul pe care îl testează pentru a-l face să eșueze.
  - Va eșua, dar probabil doar în aspecte minore, deoarece cerințele au fost bine analizate și sistemul a fost bine conceput pentru a satisface aceste cerințe.
- Se fac eventualele modificări pentru a corecta problemele găsite și, în sfârșit, un sistem compilat curat, complet cu documentație curată, este gata de livrare.

#### (7) Acceptarea sistemului

- Se face demonstrația sistemului în fața clientului, probabil cu implicarea lui directă, pentru a-i câștiga acceptarea formală.
- Condițiile de acceptare nu sunt subiective; au fost stabilite și convenite la începutul vieții proiectului.
- Tot ce este necesar acum este să arătăm că programele îndeplinesc acele criterii convenite anterior.

## (8) **Livrare**

- Odată acceptat, sistemul este livrat clientului.
- Dacă nu a fost posibil sau fezabil să se facă teste de acceptare la instalarea sa, ar putea exista încă un set de teste desfășurat în condiții de funcționare reale.
- La încheierea acestor teste la fața locului, proiectul este finalizat, cu excepția oricărora lucrări ulterioare convenite pentru a ajuta la menținerea sau îmbunătățirea sistemului.
- În cazul sistemelor mari, pot fi create și apoi construite următoarele versiuni ale sistemului.

## (9) **Încheierea proiectului**

- La această etapă managerul de proiect:
  - (1) Finalizează **istoricul** activităților proiectului.
  - (2) Face o **comparație** între **ceea ce a fost planificat** inițial și **ceea ce a avut loc de fapt**.
  - (3) Realizează o **analiză post mortem** a desfășurării proiectului (ce a fost bun, ce mers prost, ce trebuie reținut pe viitor, ce trebuie evitat)
  - (3) Apoi promovează pe toată lumea și toți pleacă acasă pentru a-și reîntâlni familiile pe care în ultima perioadă le-au neglijat.

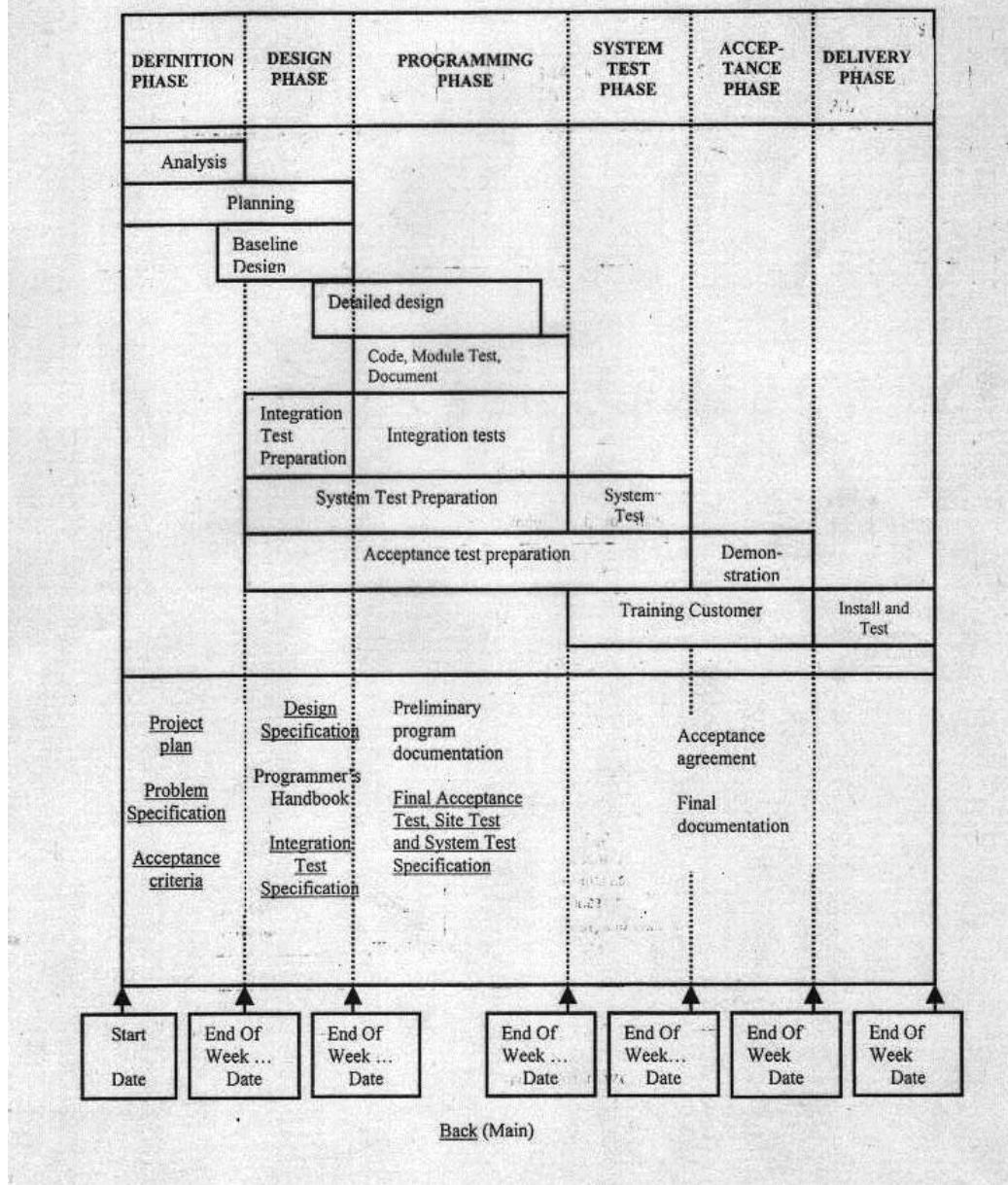
## 5. Ciclul de viață al unui proiect

### 5.1 Model de ciclu de viață al unui proiect software

- Problema centrală cu atât de multe proiecte eșuate este **pierderea controlului**, deoarece lucrurile nu sunt păstrate **suficient de vizibile**.
  - **Cerințele** sunt adesea **invizibile**, sau cel puțin obscure, deoarece nu ne facem timp să le explicăm.
  - **Designul și codul** sunt adesea **invizibile**, deoarece, dacă există, ele sunt fie în capul oamenilor, fie pe liste private fie bucăți de hârtie.
- Unul dintre obiectivele noilor tehnologii de programare este de a face **fiecare etapă a sistemului de programe** în curs de dezvoltare **vizibilă** și **disponibilă** pentru toți.
- Dar asta nu este suficient.
  - **Proiectul în sine** trebuie să fie **vizibil**.
  - El **nu** trebuie să devină o **colecție amorfă** de oameni, documente și activități.
  - Trebuie să fie împărțit în **bucăți** care **pot fi abordate**, la fel cum un program trebuie împărțit în părți abordabile de către programatori.
  - Cu alte cuvinte, pentru ca **proiectul să fie gestionabil** el trebuie conceput **modular**.
- Maniera de a face un **proiect modular** constă în furnizarea unui **cadrul** numit **ciclul de dezvoltare** și **împărțirea** acestuia într-o serie de **module** numite **faze**.
  - Pot fi imaginat orice număr de faze, atâtă timp cât acestea asigură vizibilitatea și permit exercitarea unui control eficient asupra proiectului.

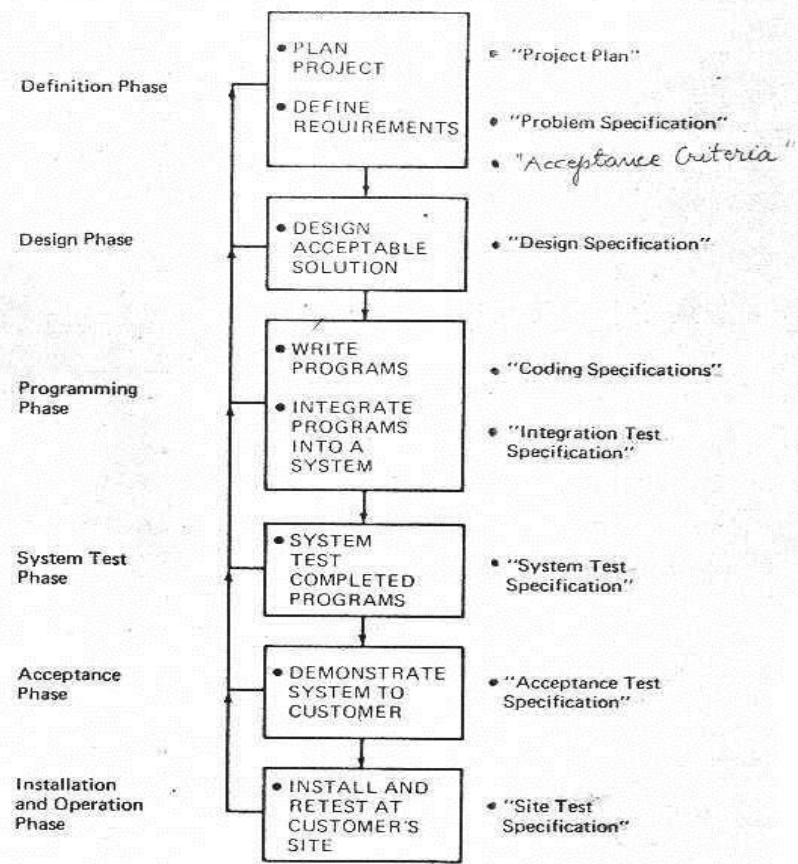
- Ceea ce este important este ca **fiecare fază** să aibă un **set foarte clar de obiective și rezultate definibile**, astfel încât toți cei implicați în proiect, să înțeleagă complet ciclul de dezvoltare planificat.
- În acest curs adoptăm Modelul Metzger pentru un **ciclu de viață al unui proiect software**.
- Ciclul de viață propus se bazează pe ciclul de viață cascădă și descrie un **ciclu de dezvoltare generic clasic** constând din următoarele faze:
  - (1) **Faza de definire**
  - (2) **Faza de proiectare (design)**
  - (3) **Faza de programare**
  - (4) **Faza de testare**
  - (5) **Faza de acceptare**
  - (6) **Faza de instalare și exploatare**
- În figura 5.1.a se prezintă o împărțire tipică a **timpului total de execuție** al proiectului între **faze**.
- În figură, fazele sunt descrise ca felii verticale de timp, ceea ce implică faptul că o fază se termină și începe următoarea, în manieră secvențială, fiecare la un moment precizat în timp.
  - Acest lucru este nerealist deoarece în practică, fazele se pot suprapune într-o oarecare măsură.
  - Deși uneori acest lucru se poate întâmpla, regula de bază ar trebui să fie următoarea: **nici o fază un începe până când faza anterioară a fost finalizată în mod satisfăcător**.
- Pot exista abateri mari de la această planificare în timp pentru unele proiecte.
  - Este foarte posibil ca faza de definiție să consume o treime din timpul total.
  - La un proiect mare de apărare, ultimele două faze ar putea dura jumătate din timpul proiectului.
  - Părțile ciclului de dezvoltare cel mai adesea scurte sunt partea din față și cea din spate a proiectului
    - (1) La începutul proiectului, planificarea este adesea întâmplătoare (să începem, să scriem programe), analiza este slabă (înțelegem cu toții problema clientului) și designul de bază este inexistent.
    - (2) În partea din spate, testarea sistemului uneori nici măcar nu este inclusă într-un plan (nu mai este timp și, oricum, testul de integrare a programatorilor face aceeași treabă).
- Nu există un standard de încredere pentru alocarea timpului.
  - Experiența cu proiecte similare este cel mai bun ghid.
- În general, în majoritatea proiectelor se alocă o treime din timpul calendaristic total pentru **fazele de definire și proiectare**, o treime pentru **faza de programare** și o treime pentru **rest**.

### The Model of Project Life Cycle



**Fig.5.1.a.** Model de ciclu de viață pentru un proiect software

- Figura 5.1.b sumarizează **fazele, activitățile** și cele mai importante **documente** asociate.



**Fig.1.9.b.** Faze, activități și documente cheie

### (1) **Faza de definire –**

- În această fază, se **definește problema clientului** și se scrie un **plan pentru proiect**.
- În timpul activității de **definire a problemei**, ideile despre **soluții** vor fi inevitabil discutate, dar adoptarea oricărei soluții specifice este amânată până în faza de **proiectare**.

### (2) **Faza de proiectare –**

- De moment ce clientul și dezvoltatorul au convenit asupra definirii problemei, dezvoltatorul edactează un document de proiectare care descrie arhitectura unei soluții acceptabile la problemă.

- De obicei, sunt propuse multe soluții, dar dezvoltatorul și clientul trebuie aleagă una și să rămână cu ea.

(3) **Faza de programare –**

- Odată definită problema și concepută o schiță a soluție de rezolvare, se trece la construcția și testare componentelor sistemului de programe în conformitate cu acea schiță.

(4) **Faza de testare a sistemului –**

- După ce programatorii au construit un produs de care sunt mulțumiți, un grup separat efectuează un nou **set de teste** într-un mediu cât mai aproape posibil de cel real de funcționare.

(5) **Faza de acceptare –**

- Pentru sistemul de programe odată terminat, cu documentația inclusă, se realizează o demonstrație în fața clientului pentru a obține acordul formal al acestuia că sistemul îndeplinește contractul.
- Acceptarea se bazează pe îndeplinirea criteriilor pe care dezvoltatorul și clientul le-au convenit la începutul ciclului de dezvoltare.

(6) **Faza de instalare și exploatare –**

- Programele acceptate sunt transferate în mediul lor final de operare pe echipamentul clientului, retestate în acel mediu și apoi puse în funcțiune.

## 5.2 Documente cheie

- În cadrul unui proiect există câteva There are three **documente tehnice cheie** (Fig.5.2.a). Este vorba despre:

DOCUMENT NAME	WHEN WRITTEN	WHAT IT DOES	WHO WRITES	SOME POSSIBLE FORMS
"Problem Specification"	Definition Phase	Defines the problem for which a solution is needed	Analysts	Narrative HIPO Tables Data flow diagrams Data dictionaries
"Design Specification"	Design Phase	Describes the overall solution	Designers	Narrative HIPO Tables Flow charts
"Coding Specification"	Programming Phase	Describes the detailed solution	Programmers	HIPO Pseudo code Procedure charts Code Flow charts

**Fig.5.2.a. Documente tehnice cheie**

(1) **Specificația problemei** este documentul produs de analiști dezvoltatorului care descrie problema clientului. Definește **cerințele** jobului care trebuie efectuat.

(2) **Specificația de proiectare** este scrisă în timpul fazei de proiectare. Descrie arhitectura soluției generale a problemei.

(3) **Specificația de codificare**, este extensia detaliată a specificației de proiectare.

- Acesta este **setul fundamental de documente** care descriu din punct de **vedere tehnică** sistemul programului în detaliu.
- Există, de asemenea, un set de **documente de management** reunite în Planul de proiect
- Este foarte important ca Managerul de Proiect să stabilească cât mai timpuriu, un Sistem de Documentare pentru a gestiona documentele proiectului.

### 5.3 Testarea documentelor

- Este extrem de benefică ideea de a **testa documentele în detaliu**, aşa cum se testează programele.
- **Documente** precum cele discutate în secțiunea precedentă, precum și **manualele utilizatorului, specificațiile de testare** și aşa mai departe, sunt la fel de esențiale pentru un proiect de succes ca orice alt element dezvoltat.
- O modalitate bună de a testa un document este de a-l **supune controlului atent** de către alții în timpul unei „**examinări structurate**”.
  - Nu este indicat ca un document să fie transmis pur și simplu altora pentru comentarii; bazându-ne pe faptul că dacă noi suntem lenesi este mai comod să presupunem că următorul cititor va fi mai riguros.
- Există **două criterii** pentru **testarea documentelor**:
  - (1) Să fie **complete** și **absolut exacte**.
  - (2) Să fie **lizibile** și **ușor de înțeles**.
- **Documentele proiectului** reprezintă pentru **utilizator produsul realizat de dezvoltator**. Documentele sunt **tangibile** și **vizibile**, în timp ce programele nu au aceste calități. De aceea, activitatea de a le face prietenoase și ușor de citit este la fel de importantă ca orice altă activitate din proiect.

### Exercițiul #3

1. Care este definiția termenului de management de proiect?
2. Descrieți elementele cheie ale Cadrului de Management al Proiectului: părțile interesante, domeniile de cunoaștere, instrumente și tehnici.
3. Explicați următorii termeni: manager, manager de nivel 1, manager de nivel 2, program, software.
4. Care sunt regulile de bază pentru un proiect SW? Descrieți semnificația lor.
5. Ce este un contract? Ce conține un şablon de contract? Care este semnificația semnării?
6. Ce tipuri de contracte din punct de vedere al prețului cunoașteți? Descrieți-le precizând avantajele și dezavantajele fiecăruia.
6. Descrieți principalele drepturi ale clientilor și principalele responsabilități ale clientului în relația cu un dezvoltator.
7. Descrieți dezvoltarea unui proiect ideal.

8. Descrieți un ciclu de viață generic pentru un proiect SW. Subliniați conținutul fazelor principale.
9. Care sunt principalele activități și documente pentru dezvoltarea unui proiect SW?

## **Capitolul 4. FAZA DE DEFINIRE**

### **1. Obiectivele fazei de definire**

### **2. Analiza problemei**

- 2.1 Recomandări pentru activitatea de analiză
- 2.2 Documentul de specificare a problemei
- 2.3. Sarcinile analiștilor

### **3. Activități de planificare a proiectului**

#### **3.1 Sistemul**

- 3.1.1 Definiție
- 3.1.2 Caracteristicile unui sistem

#### **3.2 Instrumente de planificare**

- 3.2.1 Schița planului de proiect
- 3.2.2 Diagrame de bare
  - 3.2.2.1 Ghid pentru utilizarea diagramelor de bare
  - 3.2.2.2 Ghid pentru utilizarea etapelor de reper
- 3.2.3 Diagrame de reper
  - 3.2.3.1 Ghid pentru utilizarea etapelor de reper
- 3.2.4 Rețele de activitate

### **4. Estimarea dimensiunii software-ului**

#### **4.1 Context**

- 4.2 Cadrul de estimare a dimensiunii software-ului
  - 4.2.1 Relația dimensiune-resurse
  - 4.2.2 Câteva experiențe de estimare
  - 4.2.3 Criterii de estimare a dimensiunii software-ului

#### **4.3 Metode de estimare a dimensiunii software-ului**

- 4.3.1 Metode orientate pe dimensiune
  - 4.3.1.1 Metoda Wideband-Delphy
  - 4.3.1.2 Metoda Fuzzy-Logic
  - 4.3.1.3 Metoda Componentelor standard
- 4.3.2 Metode orientate pe funcții
  - 4.3.2.1 Metoda Punctelor-funcționale
  - 4.3.2.2 Conversia punctelor funcționale în SLOC
  - 4.3.2.3 Metoda Punctelor caracteristice
  - 4.3.2.4 Estimarea bazată pe proxy-uri

### **5. Estimarea costurilor proiectelor software**

- 5.1 Obiective de estimare a costurilor
- 5.2 Resursele unui proiect software
- 5.3 Elementele de cost ale unui proiect software
- 5.4 Metode de estimare a costurilor proiectelor software
  - 5.4.1 Metoda analogică
  - 5.4.2 Judecata expertilor
  - 5.4.3 Metoda de estimare de jos în sus

- 5.4.4 Metoda de estimare de sus în jos
- 5.4.5 Metoda combinată
- 5.4.6 Metoda Parkinson
- 5.6.7 Metoda prețului câștigător
- 5.5 Modele de evaluare a costurilor software
  - 5.5.1 Modele de descompunere
    - 5.5.1.1 Modelul estimare efort (model EE)
    - 5.5.1.2 Modelele LOC (linii de cod) și FP (puncte funcționale)
  - 5.5.2 Modele parametrice
    - 5.5.2.1 Modelul COCOMO 81
    - 5.5.2.2 Modelul COCOMO II
    - 5.5.2.3 PRET S Model
    - 5.5.2.4 Modelul SEER SEM
- 5.6 Productivitatea activității software

## **6. Ghid de estimare a proiectelor**

- 6.1 Istoricul proiectului

## **7. Planul proiectului**

- 7.1 Caracteristicile unui plan bun
- 7.2 Redactarea planului de proiect
- 7.3 O schiță a planului de proiect
  - 7.3.1 Prezentare generală
  - 7.3.2 Planul de etape
  - 7.3.3 Planul de organizare
  - 7.3.4 Plan de testare
  - 7.3.5 Modificarea planului de control
  - 7.3.6 Plan de documentare
  - 7.3.7 Plan de instruire
  - 7.3.8 Planul de revizuire și raportare
  - 7.3.9 Plan de instalare și exploatare
  - 7.3.10 Resurse și plan de livrare
  - 7.3.11 Indexul planului

## **8. Criterii de acceptare**

## **9. Tehnologia WBS**

- 9.1 Ce este WBS
- 9.2 Scopurile WBS
- 9.3 Abordări ale dezvoltării WBS
  - 9.3.1 Utilizarea ghidurilor
  - 9.3.2 Abordarea analogiei
  - 9.3.3 Abordarea de sus în jos
  - 9.3.4 Abordarea de jos în sus
- 9.4 Câteva principii de bază pentru a crea un WBS bun
- 9.5 Cum se stabilește WBS
- 9.6 Pachetele de lucru
- 9.7 Concluzii

**Exercițiu #4**

**Exercițiu #6**

**Exercițiu #7**

## Capitolul 4. FAZA DE DEFINIRE

### 1. Obiectivele fazei de definire

- (1) Primul obiectiv al **fazei de definire** este *analiza problemei*.
- (2) Un **al doilea obiectiv** este *planificarea proiectului*, care înseamnă conceperea unei scheme care va produce o soluție acceptabilă a problemei.
- (3) Un **al treilea obiectiv** este scrierea și obținerea *aprobării clientului* cu privire la setul de criterii de acceptare care să vor ajuta să determinați dacă produsul dvs. îndeplinește cerințele contractului.

### 2. Analiza problemei

- **Tînta** analizei problemei constă în:
  - Redactarea unui **document de cerințe** care descrie cu acuratețe și la nivel de detaliu **problema clientului**.
- Acest document este cunoscut sub diferite denumiri:
  - Specificația problemei;
  - Specificația cerințelor;
  - Specificație (Spec);
  - Specificația cerințelor software (SRS).
- În unele cazuri, acest document a fost deja redactat, cel puțin la un nivel moderat de detaliu, în timpul eforturilor de propunere premergătoare atribuirii contractului.
- În acest caz, munca în timpul fazei de definire poate fi pur și simplu:
  - (1) Ajustarea documentelor existente.
  - (2) Completarea detaliilor.
  - (3) Formalizarea lor.
- De obicei, cerințele au fost doar **schițate** doar în timpul fazei de propunere.
  - În acest caz, această primă fază a contractului va implica multă **muncă de analiză** în vederea definitivării cerințelor
- Este adesea logic să se contracteze **două joburi separate**; unul pentru **a defini problema** și unul pentru **a rezolva problema** respectivă.
  - Guvernul federal al SUA folosește adesea acest principiu, în special în cazul **contractelor foarte mari** sau când **problemele tehnice sunt formidabile**.
  - Guvernul încheie un acord **cu doi sau mai mulți contractori simultan** și îi **finanțează pe fiecare** pentru a elabora o **definire a problemei și un concept de proiectare**.

- Guvernul este apoi liber să aleagă ceea ce îi place dintre diferitele concepte prezentate.
- În continuare se poate alege **un singur contractant** pentru a continua cu **dezvoltarea completă** în conformitate cu abordarea aleasă.
- Perioada în care sunt elaborate abordările separate se numește de obicei **Faza de definire a contractului**.
- Unii furnizori de software fac afaceri în mod regulat în acest fel.
  - **Abordarea în două etape** limitează răspunderea acestora și asigură că clientul înțelege ceea ce cumpără.
- Nu este indicat a se presupune că problema este evidentă și că toată lumea o înțelege.
- Este recomandat să fie redactat documentul de cerință chiar și în cazul lucrărilor interne sau proiectelor interne în care lucrarea se desfășura sub „comanda magazin” (un fel de contract intern în cadrul companiei) către un alt departament.
- O **analiză serioasă a problemei**, **evită** posibilele agravări, pierderi de timp și de bani (neplata contractului, pierderea profitului și chiar acțiunea în instantă).

## 2.1 Recomandări pentru activitatea de analiză

- (1) Rezistați tentației de a începe imediat **proiectarea programelor**.
- (2) Abordarea trebuie să se concentreze mai întâi asupra **care este problema**, nu pe maniera în care ea va fi rezolvată.
- (3) În timp ce analiștii descriu **care este problema**, ei se vor gândi și vor discuta cu siguranță și despre concepte de design.
  - Dar asigurați-vă că analiștii știu că **primul lor obiectiv** este **să scrie un document** care să descrie **problema, nu soluția**.
- (4) Începeți documentul prin a descrie problema clientului de la zero, într-un limbaj non-tehnic.
- (5) Identificați:
  - Cine este clientul;
  - Care este ambientul (mediul) problemei;
  - De ce se caută o soluție informatică.
- (6) Descrieți problema tehnică la niveluri crescătoare de detaliu.
- (7) Fiți foarte specifici cu privire la capabilitățile care urmează să fie incluse în sistem;
  - Uneori este util de subliniat ceea ce **nu este inclus**.
- (8) Fiți cât se poate de precisi.
  - Nu lăsați cititorul să deducă ce este inclus și ce nu.

## 2.2 Documentul de specificare a problemei

- Documentul aflat în atenție în această fază, **Specificația problemei**, definește în termeni cantitativi cerințele clientului.
- Această specificație stabilește cerințele grupate pe câteva categorii majore (fig. 2.2.a.)
  - (1) **Performanță**, inclusiv capacitați de fișiere, constrângeri de sincronizare, rate de intrare și încărcări ale sistemului.
  - (2) **Cerințe operaționale**: funcții, sau operațiuni, sau caracteristici care trebuie furnizate de sistem.
  - (3) **Cerințe de date**.
  - (4) **Interfețe umane**: interacțiuni între utilizator și produsul software.
  - (5) **Performanțe umane**: considerații cum ar fi timpii minimi pentru luarea deciziilor, timpii maximi admisibili pentru răspunsurile sistemului și restricții privind afișajele generate de program.

### SPECIFICAȚIA PROBLEMEI (MODEL)

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZATOR:**

#### SECȚIUNEA 1: SCOP

Specificația problemei descrie „ce-ul” proiectului și cerințele sistemului de programe, adică treaba care urmează să fie realizată de programe. Este un document de bază și cea mai recentă ediție a acestuia trebuie respectată cu strictețe de către întreg personalul proiectului.

#### SECȚIUNEA 2: DOCUMENTE APLICABILE

#### SECȚIUNEA 3: CERINȚE

Această secțiune reprezintă inima documentului, precizează cât mai detaliat posibil ce trebuie să le facă programul. Aproape toate informațiile de aici vor fi o combinație de descriere narativă, matematică și date tabelare. Diagramele HIPO și diagramele fluxului de date pot fi de

asemenea folosite pentru a exprima relația funcțională necesară, dar nu și designul logic al programului.

### 3.1. Parametrii de performanță

Această subsecțiune explică cerințele de performanță ale sistemului referitor la ratele de transfer, debitul de informație etc., așa cum sunt ele impuse de mediul de lucru. Aceste cerințe pot fi precizate în termeni de capacitați de fisierare, constrângeri de timp acceptabile, rate de intrare permise și așa mai departe. Aceste cerințe sunt enunțate în termeni cantitativi, cu toleranță acolo unde este cazul.

### 3.2. Cerințe Operaționale

Această subsecțiune descrie cerințele funcționale ale sistemului de programe. Intenția este de a preciza toate operațiunile funcționale ale programelor, relația dintre aceste funcții și legătura dintre funcțiile programului și alte funcții ale subsistemului. Fiecare funcție de program este definită în continuare în subsecțiuni separate, așa cum se arată mai jos.

#### 3.2.1. Funcție (Caracteristică, Trăsătură) 1

3.2.1.1. Sursa și Tipurile de Intrari

3.2.1.2. Sursa și Tipurile de Intrari

3.2.1.3. Diagrama funcțională

.....

#### 3.2.n. Funcție (Caracteristică, Trăsătură) n

3.2.n.1. Sursa și Tipurile de Intrari

3.2.n.2. Sursa și Tipurile de Intrari

3.2.n.3. Diagrama funcțională

### 3.3. Cerințe de date

Această secțiune definește parametrii de date care afectează proiectarea sistemului de programe, de exemplu, coordonatele geografice pentru site-urile operaționale. Definiția detaliată a parametrilor include descrierii ale datelor, definiții ale unităților de măsură și cerințe de precizie.

### 3.4. Interfețe umane

Această secțiune descrie interacțiunile dintre utilizator și produsul software definind tipul de interacțiuni, conținutul, cerințele generale ale formei interacțiunii, etc.,

### 3.5. Performanțe umane

*Această secțiune descrie cerințele care implică interacțiuni umane cu sistemul de programe, cum ar fi timpii minimi pentru decizii, timpii maximi pentru răspunsurile sistemului, restricții privind afișajele generate de program.*

---

**Fig.2.2.a. Modelul documentului Specificația problemei**

### **2.3 Sarcinile analiștilor**

- Analiștii echipei de dezvoltare care redactează **Specificarea problemei**, au următoarele sarcini:
  - (1) **Să ia legătura cu clientul real.**
    - De obicei, există mulți oameni care pot fi considerați „client” și care toți pot formula cerințe diferite analiștilor. De exemplu:
      - (a) Un **cumpărător** care are în vedere să mențină costurile cât mai coborâte;
      - (b) Un **analist de personal** care dorește un sistem cu o mulțime de gadget-uri nostime;
      - (c) Un **administrator de contract** care poate să ști puține despre partea tehnică a jobului;
      - (d) Un **utilizator** care în cele din urmă va fi agasat de sistemul care se construiește.
    - Analiștii **nu** trebuie să presupună că membrii organizației clientului vorbesc între ei;
      - Ca să nu mai vorbim despre **acordul** asupra ceea ce **dezvoltatorul** ar **trebui să ofere** la sfârșit.
      - Dacă analiștii **nu** vorbesc cu **oamenii potriviti** la **momentul potrivit**, este posibil să nu înțeleagă clar la ce se așteaptă cu adevărat acest client cu multe capete.
        - De exemplu, dacă analiștii ignoră **utilizatorul real** până când este timpul să predea sistemul terminat, rezultatele pot fi tragice.
        - **Utilizatorul** care **nu** ia în niciun fel parte la specificarea sistemului poate fi cel mai reticent în a-l accepta.
      - Analиstii trebuie să afle cu diplomație, din punctul de vedere al utilizatorului:
        - (a) Cine controlează ce.
        - (b) Cine are puterea reală, cine aproba produsul
        - (c) Cine va folosi în cele din urmă produsul.

- (2) **Să capteze mentalitatea clientului.**

- Analistii trebuie să fie pricepuți să afle ce își dorește cu adevărat clientul, deoarece acest lucru poate fi diferit de ceea ce a fost specificat într-un contract vag.
- Trebuie să citească atât ceea ce li se oferă, cât și ceea ce este între rânduri.
- Trebuie să intervieze oamenii și să încerce să înțeleagă ceea ce se dorește de fapt cu adevărat.
- Analistii trebuie să asculte cu atenție, să reformuleze ceea ce a spus clientul, să îi reproducă spusele acestuia și să întrebe: Asta vrei să spui?

### (3) Să consemneze totul în scris.

- Analistii pot scrie **specificația problemei** într-un număr mare de moduri.
  - Unele dintre aceste moduri vor fi ușor de implementat de către designeri, altele dificile, altele imposibile.
- O echipă de analiză lipsită de experiență în programare poate genera mari probleme.
  - Apropo, la fel poate și o echipă cu doar experiență de programare.
- Analistii trebuie să prezinte problema într-un limbaj clar și precis, care să fie acceptabil atât pentru client, cât și pentru proiectanți.

### (4) Să păstreze specificația cât simplă.

- Întotdeauna să prefere **simplitatea** în loc de **complicăție**.

### (5) Să aprobe documentul în mod treptat (gradual).

- Pe măsură ce scriu secțiuni din **Specificația problemei**, analistii ar trebui să obțină **aprobarea provizorie a clienților** pentru ceea ce au scris.
- **Nu** este indicat a prezenta clientului **documentul terminat** dintr-o dată, pentru că dacă nu îi convine, analistii sunt puși într-o poziție ingrată.
- Clientul trebuie **să cunoască îndeaproape** ceea ce fac analistii pe **măsură ce** se avansează în redactare.

- **Remarci**
- În mod evident, analistul ar trebui să reprezinte mai multe discipline: **programator, vânzător, inginer, psiholog și scriitor**.
- **Managerul de proiect** **nu** va găsi probabil toate atributile de care are nevoie la o singură persoană, dar poate construi **un grup** care să aibă **în mod colectiv** toate **acreditările** necesare.
- **Recomandare:** **Nu** vă bazați pe **titluri** atunci când vă **selectați** oamenii.
- Există suficient de mulți oameni în jur care se intitulează „analisti” pentru că nu se încadrează în nicio altă categorie.

### **3. Activități de planificare a proiectului**

- Multe proiecte de programare sunt tratate ca niște romane de mister din cauza multiplelor probleme pe care le ridică.
- Iată o listă de posibile **probleme** care tind să iasă la suprafață:
  - Planificare slabă;
  - Contract prost definit;
  - Definirea instabilă a problemei;
  - Analiza slabă a riscurilor;
  - Management neexperimentat;
  - Presiuni politice;
  - Control ineficient al modificărilor;
  - Termene limită stabilite nerealist;
  - Planificare slabă.
- Lista ar putea avea mai multe pagini dar două elemente ar rămâne vizibile, unul prin prezența sa, altul prin absența sa:
  - Întotdeauna este prezentă **planificarea proastă**.
    - Uneori, asta înseamnă **eșecul** de a lua în considerare jobul din toate unghiiurile.
    - Alteori aceasta înseamnă că în esență **nu există niciun plan**.
  - Lipsesc de pe listă este **dificultățile tehnice**.
    - Asta nu înseamnă că multe joburi nu sunt dificile din punct de vedere tehnic, dar sunt puține care depășesc „starea de ultimă generație” (state-of-the-art).
    - De fapt, dacă sunteți de acord să faceți o treabă care este dincolo de stadiul actual al tehnicii, respectiv un job care necesită progrese tehnice de ultimă oră, aceasta în definitiv este o **planificare proastă**.
    - O excepție evidentă este un proiect al cărui obiectiv este unul de **cercetare fundamentală**.

#### **3.1 Sistemul**

- În ultimă instanță, **lucrul** pe care dezvoltatorul intenționează să îl construiască în cadrul unui proiect software se numește de fapt **sistem**.

##### **3.1.1 Definiția unui sistem**

- Un **sistem** este o combinație structurată de **părți care interacționează** care satisfac un **set de obiective**.
- Există exemple de sisteme pretudindeni:
  - Sistemul solar;
  - Un sistem de distribuție a energiei electrice;
  - Corpul uman;
  - Sistemul digestiv;
  - O corporație;
  - O ascușitoare;
  - Un sistem de calcul.
- Toate aceste sisteme satisfac definiția inițială.
- În realitate, fiecare **sistem** este de fapt un **subset** al unui alt **sistem mai cuprinsător**.
- Dacă ne referim spre exemplu la termenul „**sistem de programe**”.
  - Da, desigur, **sistemul de programe** este un subsistem al unui **sistem de procesare a datelor**;
  - Care, la rândul său, poate fi un subsistem al unui **sistem de arme**;
  - Care este un subsistem al unui **sistem de apărare**;
  - și mai departe și mai departe.

- **3.1.2 Caracteristicile unui sistem**

- Orice sistem, indiferent de natura sa are două caracteristici esențiale: **interacțiunile și modificările**.

#### (1) **Interacțiunile unui sistem**

- Prin definiție, un sistem conține **părți** care trebuie să **interacționeze** între ele (fig.3.1.2.a).
  - Numărul de interacțiuni potențiale în cadrul unui sistem depinde de numărul de elemente din sistem.
- **Controlul interacțiunilor** dintre părți devine o **sarcină majoră** pe măsură ce sistemul crește în dimensiune și complexitate.
- **Energia de management** necesară pentru a **controla, minimiza și simplifica interacțiunile** din cadrul sistemului este semnificativă.
  - **Interacțiunile** pot fi controlate dacă recunoaștem mai întâi că ele există.
- Există modalități deja consacrate care pot ajuta la minimizarea efectelor interacțiunilor. Dintre cele mai cunoscute se remarcă:

- (1) Modularizarea;
- (2) Definirea interfețelor;
- (3) Organizarea proiectului.

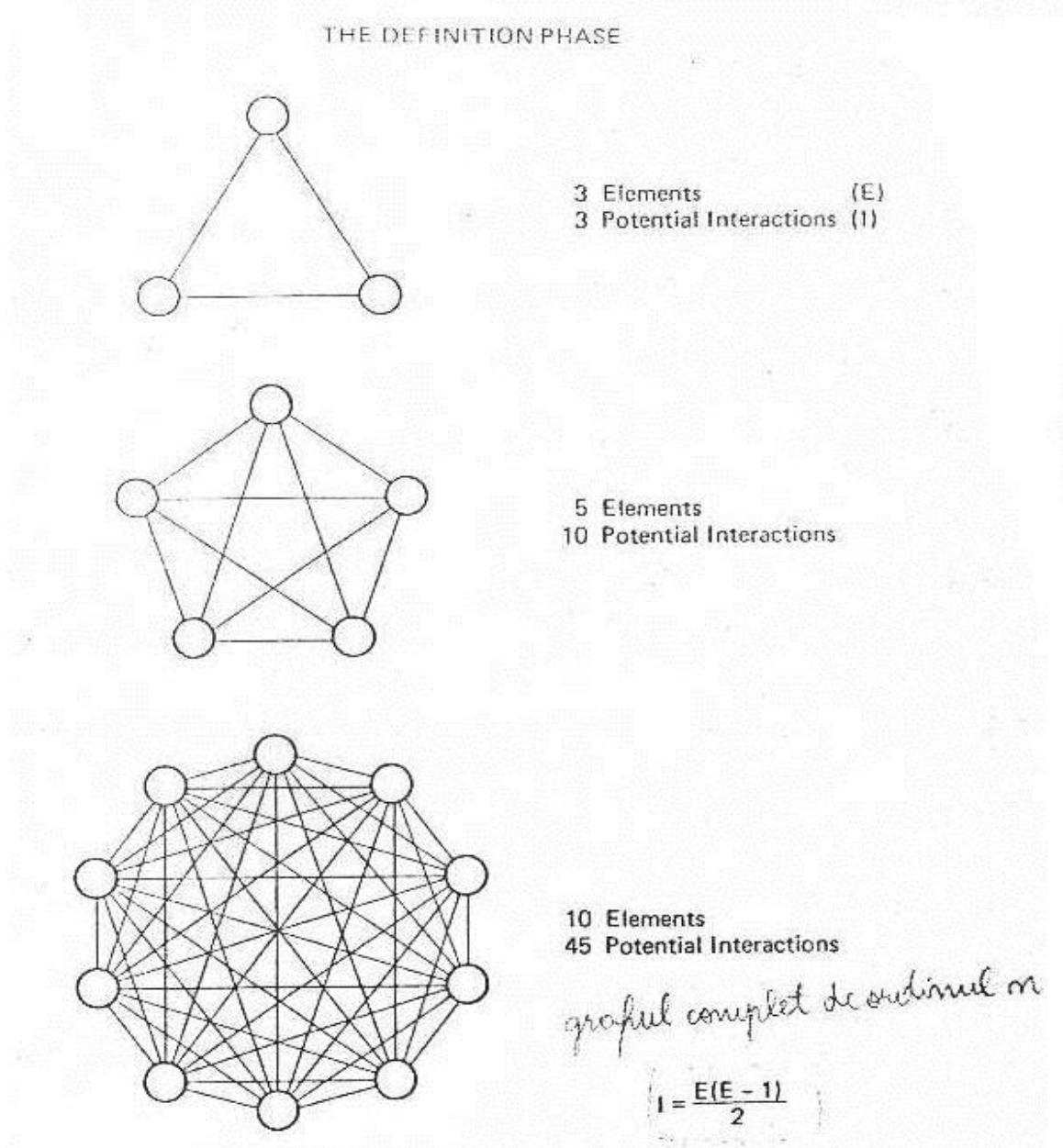


Figure 2.3. Interactions<sup>2</sup>

**Fig.3.1.2.a. Interacțiuni**

## (2) Modificările unui sistem

- Având în vedere orice job care se întinde pe mai mult de câteva săptămâni, putem anticipa cu certitudine că vor apărea **modificări (schimbări)**.
- Câteva exemple de tipuri de modificări sunt enumerate în continuare:

### (1) Modificări ale cerințelor:

- Definirea problemei la care lucrează analiștii de la începutul jobului rareori rămâne nemondicată.
- Cu cât jobul este mai mare, cu atât este mai probabil să apară schimbări în cerințe.

### (2) Modificări de design:

- Designul de bază este conceput ca **fundament** pentru efortul de programare.
- Dar, după cum știe orice proprietar de case, fundațiile se modifică, crapă și trebuie să fie uneori petice.
- Un sistem de programe nu este diferit.
- Este recomandat ca designul de bază să constituie un început bun, dar este extrem de probabil ca el să fie schimbat.

### (3) Schimbări tehnologice:

- Uriașele joburi de programare sponsorizate de guvern (de exemplu, sisteme de rachete antibalistice, sisteme militare de comandă și control), dar nu numai, sunt deosebit de vulnerabile la schimbările tehnologice din două motive:
  - În primul rând, acestea se întind pe perioade atât de lungi de timp, încât noile dezvoltări științifice și de inginerie (de exemplu, în armament și echipamente de prelucrare a datelor) sunt inevitabile.
  - În al doilea rând, însăși natura acestor proiecte constă în faptul că ele împing înainte limitele stadiului actual și sunt adesea direct responsabile pentru inovația tehnologică.

### (4) Schimbări sociale:

- Multe proiecte sunt victimele involuntare ale schimbărilor care au loc în modul în care se comportă un segment mare al societății.
- De exemplu, programele de gestionare a cecurilor de salarizare sunt bombardate în mod constant cu schimbări, deoarece legile fiscale se modifică sau pentru că oamenii fac noi solicitări sistemului prin efectuarea din ce în ce mai multe deduceri personale.

### (5) Oamenii se schimbă:

- Oamenii pleacă, mor, se îmbolnăvesc, și schimbă locul de muncă.

- Când se pierde o persoană cheie din personal, sau când un membru important al organizației dispare, apare o problemă potențială.

(6) **Corecții:**

- Oamenii fac **erori**.
- Întotdeauna au făcut și o vor face întotdeauna.
- Erorile pot fi majore sau minore, tehnice sau administrative, scandalioase sau subtile, dar sunt totuși erori și trebuie remediate.
- Ceea ce este important la **schimbări** este ca acestea să fie **controlate, nu eliminate**.
  - De exemplu: Dacă există o modificare în specificația problemei:
    - (1) Se estimează impactul modificării asupra costurilor proiectului și a datelor de livrare;
    - (2) În cazul în care clientul dorește totuși schimbarea, se poate negocia eventual o modificare a contractului;
    - (3) Se emite o notificare formală de modificare;
    - (4) Se încorporează modificarea și se continuă treaba.
- În acest context, și nu numai, este foarte important ca dezvoltatorul:
  - (1) Să se definească un **mecanism coherent de control a modificărilor**
  - (2) Să stabilească un **sistem de documente de bază exacte și semnificative**.

## 3.2 Instrumente de planificare

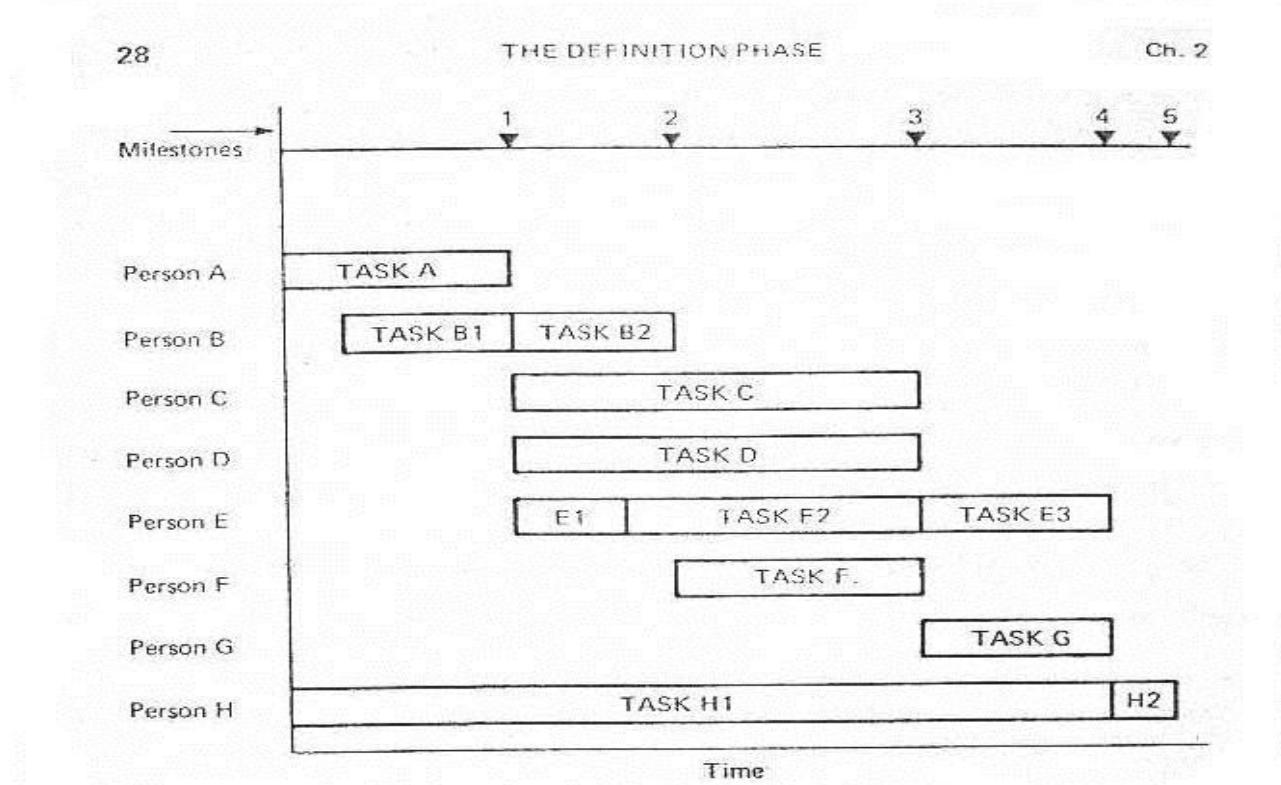
### 3.2.1 Schița planului de proiect

- Cea mai grea parte a oricărei activități de scris este începutul.
- Acesta este motivul pentru care se recomandă începerea unui proiect cu un **Plan de proiect (un cuprins de proiect) predefinit**.
- Acesta ar putea fi:
  - (1) Un model publicat în literatură [Me81], [Kr99].
  - (2) Un model dezvoltat în organizația dumneavoastră bazat pe istoria și experiența dumneavoastră.
- Acesta poate fi folosit ca punct de pornire, poate fi modificat pentru a se potrivi situației curente și poate constitui eșafodajul viitorului Plan al Proiectului.
- A începe cu o schiță a planului de proiect are multe avantaje:
  - (1) Nu se pierde timpul încercând să se conceapă un plan nou.
  - (2) Această schiță are o credibilitate încorporată, deoarece este rezultatul contribuției mai multor manageri de programe cu experiență.

(3) Orice schiță folosită ca starter poate contribui substanțial la reducerea numărului de rescrieri, economisind foarte multe ore de lucru mai târziu.

### 3.2.2 Diagrame de bare

- **Diagramele de bare**, numite și **diagrame Gantt**, sunt foarte familiare ca instrumente de planificare.
- Diagramele de bare sunt simplu de construit și pot fi utile pentru a descrie **planificarea** sau **utilizarea resurselor** în funcție de **tempo** (fig. 3.2.2.a).



**Fig.3.2.2.a.** Exemplu de diagram de bare

- Avantajele diagramei cu bare sunt următoarele:
  - Indică dintr-o privire cine este repartizat cărei sarcini (task).
  - Prefigurează într-o manieră intuitivă proiecția timpului.
- Diagrama cu bare operează de obicei cu sarcini sau taskuri
  - Sarcinile pot fi cât de mari sau cât de mici se stabilește; ceea ce este important este că dimensiunea lor să ofere nivelul de control care se dorește.

- Un **manager de proiect** va dori probabil o diagramă care să indice **sarcinile mari**;
  - Sarcinile mari pot avea o durată de una până la câteva luni.
- **Managerii de nivel 1** (cei care supraveghează direct programatorii) vor avea nevoie de diagrame care să arate **sarcinile mici**.
  - Managerul de nivel 1, totuși, nu ar putea exercita un control eficient asupra unei diagrame prea mari.
  - El trebuie să împartă fiecare sarcină (task) în bucăți mai mici (module), astfel încât posibilele probleme să poată fi identificate cât mai devreme.
  - Teoretic, expunerea maximă a managerului de nivel 1 este egală cu sarcina de cea mai lungă durată.
- Cât de fin urmează a fi defalcate sarcinile, depinde de complexitatea lor în raport cu jobul și de experiența și competența programatorilor.
  - O anumită sarcină poate fi subdivizată, spre exemplu, în trei **bucăți de o săptămână** pentru un programator.
  - Pentru un alt programator, aceeași sarcină poate fi atribuită spre exemplu, într-o **bucată de două săptămâni**.

### **3.2.2.1 Ghid pentru utilizarea diagramelor de bare**

- Se recomandă următorul **ghid general** pentru utilizarea diagramelor de bare:
  - (1) **Timpul minim** pentru care trebuie planificată orice sarcină este de regulă de **o săptămână**.
  - (2) Un **temp mai rezonabil** în majoritatea cazurilor ar putea fi de **două săptămâni**.
    - Dacă se încearcă subdivizarea în sarcini mai fine:
      - Programatorii vor petrece cantități disproporționate de timp raportând mai progresul în detrimentul dezvoltării propriu-zise.
      - Managerii de nivel 1 vor deveni un fel de contabili, deoarece vor fi preocupăți de înregistrarea și de urmărire tuturor acestor sarcini (taskuri) mici.
  - (3) **Timpul maxim** pentru care ar trebui programate majoritatea sarcinilor este de aproximativ **o lună** – și, din nou, **două săptămâni** ar fi mai confortabil.
    - Ideea este că, dacă o sarcină are probleme, acest lucru trebuie să fie cunoscut suficient de curând pentru a lua măsuri de remediere.
- **Diagramele cu bare** pot fi utilizate în mai multe moduri diferite pentru a evidenția:
  - (a) Oameni, sarcini și temp.
  - (b) Sarcini în funcție de temp.
  - (c) Orice versus orice, atâtă temp cât ajută în planificare și control.

- **Sarcinile** (Taskurile) afișate pe o **diagramă de bare** trebuie:
  - (1) Să fie **listate și definite** pe o foaie separată.
  - (2) Fiecare sarcină trebuie să se refere la **livrarea unui produs clar definit**, cum ar fi un **modul de program, un document concret, sau un artefact**.
- Există multe instrumente de planificare bazate pe diagrame de bare Microsoft Project Plan, Open Project, etc.,.

### **3.2.3 Diagrame de repere (Milestones) (Etape de reper)**

- Dicționarul Webster definește o **etapă de reper** ca un „**punct semnificativ în dezvoltare**”.
- Când se definesc reperele pentru un proiect, nu trebuie uitat sub nicio formă cuvântul **semnificativ**.
- Nu trebuie făcut neapărat sfârșitul **fiecărei sarcini** o **etapă de reper**.
  - În schimb, trebuie selectate acele puncte din program, la care ar fi trebuit să se realizeze ceva cu **adevărat semnificativ** și la care trebuie luată o **decizie** (de exemplu: continua, re-planifică, obține mai multe resurse etc).
- Fiecare **etapă** trebuie să se bazeze **pe ceva măsurabil**; altfel, nu se poate stabili când este atinsă.
- Câteva exemple de **repere** stabilite **greșit**:
  - **Testarea 50% finalizată**.
    - Chiar dacă „50% finalizat” semnifică ceva pentru cineva, sunt șanse să însemne ceva diferit pentru altcineva.
    - Este un **reper slab**, deoarece nu există o modalitate sigură de a ști când se ajunge acolo - **nu este măsurabil**.
    - Chiar dacă se bazează pe numărul de teste care trebuie executate, este **reperul** este neclar, deoarece spre exemplu, testele variază atât de mult în complexitate încât o „jumătate” poate necesita o săptămână de activitate, iar cealaltă „jumătate” poate dura trei luni.
  - **Codificare 50% finalizată**.
    - Aceleași obiectii se aplică aici.
    - Cum se stabilește că codarea este completă în proporție de 50%?
    - Înseamnă asta 50% din numărul anticipat de linii de cod?
    - Sau 50% din module sunt codificate?
    - Poate că 50% dintre linii sunt codificate, dar sunt cel 50% „mai ușoare”, cele cu problemele urmând să vină.

- Din nou, reperul este înșelător, deoarece poate înseamnă lucruri diferite pentru diferiți oameni.
- Iată câteva exemple de **repere stabilite corect** pentru managerul de prim nivel.
  - **Proiectarea detaliată a modulului X a fost aprobată.**
    - Aceast lucru este important chiar dacă designul respectiv poate fi modificat ulterior.
    - Înseamnă de fapt că modulul de program este conceput pe hârtie și este gata pentru codare.
  - **Modulul X a fost codificat.**
    - Aceast lucru este util, dar nu la fel de util ca la alte etape, deoarece codul se poate schimba radical pe măsură ce testarea continuă.
  - **Testarea modulului X a fost finalizată.**
    - Aceasta înseamnă că programatorul și-a testat modulul individual de cod spre satisfacția sa și este gata pentru integrare cu alte module testate.
    - Este o etapă bună, deoarece programatorul în acest moment își trimit programul fizic pentru următorul nivel de testare.
    - Înseamnă, de asemenea, așa cum va fi explicat mai târziu, că documentația descriptivă pentru modulul respectiv este completată în formă de schiță, iar acesta fapt este măsurabil – este ceva ce poate fi ținut în mâini și care poate fi văzut.
  - **Specificația Y a fost redactată.**
    - Se poate verifica foarte simplu dacă documentul este realizat fizic, sau nu.
    - Managerul poate arunca o privire rapidă asupra acesteia și poate decide dacă este sau nu într-o formă suficient de bună pentru a fi considerată finalizată și, prin urmare, dacă obiectivul a fost atins sau nu.
- Figura 3.2.3.a prezintă o listă de repere de bază pentru fiecare fază de dezvoltare pentru managerul de proiect.
  - Lista de repere poate fi modificată de o manieră convenabilă pentru a se potrivi cât mai bine cu proiectul în desfășurare.

REPER	PERIOADĂ
<ul style="list-style-type: none"> <li>• Specificația problemei redactată</li> <li>• Acceptat de client</li> <li>• Specificația preliminară a testului de acceptare redactată</li> <li>• Acceptată de client</li> </ul>	Sfârșitul Fazei de Definire
<ul style="list-style-type: none"> <li>• Specificația preliminară de proiectare scrisă</li> <li>• Acceptată de client</li> </ul>	Mijlocul Fazei de Design
<ul style="list-style-type: none"> <li>• Specificația de proiectare finalizată</li> <li>• Acceptată de client</li> <li>• Prima distribuție a Manualului Programatorului</li> <li>• Revizuirea fazei de proiectare a fost finalizată</li> <li>• Specificația testelor de integrare a fost finalizată</li> </ul>	Sfârșitul Fazei de Design
<ul style="list-style-type: none"> <li>• Specificația de testare a sistemului a fost finalizată</li> <li>• Specificația testului de acceptanță finală scrisă</li> <li>• Specificația de testare locală scrisă</li> <li>• Aprobată de client</li> <li>• Documentația programului în formă de draft curat</li> </ul>	Sfârșitul Fazei de Programare

<ul style="list-style-type: none"> <li>• Testarea sistem finalizată</li> </ul>	Sfârșitul Fazei de Testare
<ul style="list-style-type: none"> <li>• Înțelegerea de acceptanță semnată</li> <li>• Instruirea beneficiarului finalizată</li> </ul>	Sfârșitul Fazei de Acceptanță
<ul style="list-style-type: none"> <li>• Documentația programului corectată și furnizată</li> <li>• Sistemul este operațional</li> <li>• Istoria proiectului terminată</li> </ul>	Sfârșitul Fazei de Instalare și Operare

**Fig.3.2.3.a.** Etape de repere de proiect

### 3.2.3.1 Ghid pentru utilizarea etapelor de repere

- Întrebarea firească care se ridică este următoarea: [Câte etape de reper sunt necesare pentru un anume proiect?](#)
- Ca de obicei, **nu există un număr magic**, dar se recomandă a fi luate în considerare următoarele recomandări în calitate de **ghid**.
  - (1) Fiecare **manager** ar trebui să aibă [propriul său set de repere](#) pentru munca pe care trebuie să o facă oamenii care îi raportează.
  - (2) Aceasta înseamnă că [managerul de nivel 1](#) are repere pentru munca programatorilor.
  - (3) [Managerul de proiect](#) are repere pentru munca managerilor de nivel 1 și aşa mai departe.
  - (4) Totuși, **reperele** unui manager nu sunt doar suma tuturor celor utilizate de managerii care îi raportează.
  - (5) Fiecare **manager** de la fiecare nivel superior trebuie să-și concentreze energia pe problemele specifice [nivelului său](#).
    - În ultimă instanță, de aceea au fost inventate organizațiile ierarhice.
  - (6) **Nu** trebuie definit sfârșitul fiecărei sarcini ca un reper.
  - (7) Reperele trebuie să fie considerate suficient de importante pentru ca oamenii din proiect să depună un efort suplimentar pentru a le atinge.
    - Aceasta înseamnă că ar trebui stabilite suficient de puține pentru a nu avea o criză de reper săptămânală.

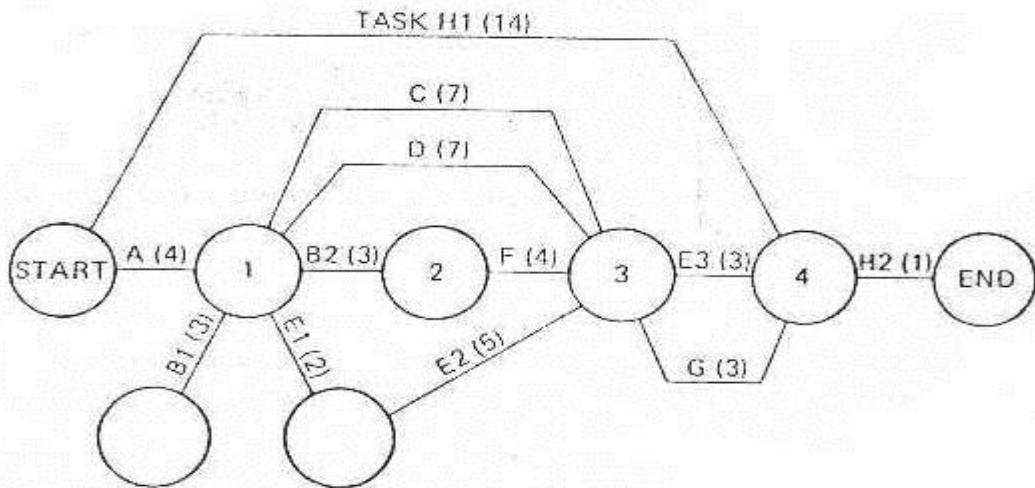
- După trei sau patru astfel de crize, oamenii vor începe să caște atunci când se strigă după ajutor.
- Este destul de evident că, dacă în aceste circumstanțe, ar apărea o criză de program adevărată, ea nu ar produce nicio emoție în rândul echipei, deoarece în timp, managerul său a strigat prea des „lupul”, fără vreun motiv intemeiat

(8) Stabilirea etapelelor de referință:

- Reperele stabilite de **managerul de proiect** pentru de managerii de nivel 1 se recomandă a fi eșalonate **la cel puțin două săptămâni**.
- Pentru nivelul următor superior, se recomandă eșalonarea **la cel puțin trei săptămâni**.
- Următorul nivel, la **intervale de patru săptămâni**.
- și aşa mai departe.

### **3.2.4 Diagrame de rețele de activități**

- Diagramele de bare sunt utile, dar au o slăbiciune semnificativă:
  - Diagramele cu bare **nu arată** în mod adecvat **interdependențele** dintre sarcini sau dintre oameni.
- Pentru a evidenția aceste interdependențe este nevoie de o diagramă de tip **rețea de activități**.
- **O diagramă de rețea de activități** este de fapt un **grafic orientat ponderat**. (fig.3.2.4.a) [Me81].



**Fig.3.4.2.a.** Rețea de activități

- (1) Cercurile reprezintă evenimente;
- (2) Liniile arată activități care sunt necesare pentru a trece de la un eveniment la altul.
- (3) Unitățile estimate de timp (sau alte resurse) pentru fiecare activitate sau sarcină sunt afișate în paranteze.
- (4) Liniile care alimentează un cerc de eveniment din stânga reprezintă activități care trebuie încheiate înainte ca acel eveniment să poată avea loc.
- În mod normal, când o diagramă de rețea este terminată, va exista o cale de la început până la sfârșit, pentru care **timpul necesar total** va fi mai mare decât pentru orice altă cale.
  - Această rută se numește **drum critic**.
  - Drumul critic necesită o monitorizare suplimentară și o atenție specială din partea managerului, pentru că dacă el alunecă, data de încheiere a întregului proiect este în pericol.
- O rețea de activități:
  - (1) Este cea mai valoroasă în timpul **activităților de proiectare** și de testare de **integrare** atunci când atenția managementului este îndreptată către **problemele de asamblare a lucrurilor**.

(2) Este mai puțin utilă pe perioada în care modulele individuale ale programului sunt scrise și testate, deoarece accentul se pune mai mult pe piesele individuale ale sistemului decât pe interacțiunile dintre ele.

(3) Deoarece utilitatea acestor diagrame dispare după finalizarea proiectării, acestea sunt adesea abandonate în acel moment.

- Acest lucru poate explica de ce **PERT** a avut doar un succes slab în programare.
- Reamintim că PERT (Tehnica de evaluare și revizuire a programelor) este o implementare formalizată, de obicei computerizată, a conceptului de **rețea de activități**, larg răspândită în mediul economic.
  - Există mai multe forme de bază și multe versiuni specifice ale rețelelor PERT.
  - Literatura abundă de cărți și articole pe această temă.
  - Efortul necesar pentru a menține rețelele de activitate PERT actualizate pe măsură ce proiectul progresează este costisitor și îi deturnează pe manageri de la supravegherea sarcinilor individuale.
- Referitor la dezvoltarea de proiecte software, nu este neobișnuit ca o rețea de activitate să se deterioreze în timpul fazei de programare.
  - Dacă nu mai face față, trebuie abandonată.
  - Nu trebuie actualizată orbește dacă nu mai este utilă.
- Rețeaua de activități ar putea fi desenată și funcție de o scară de timp, distanța dintre cercurile de evenimente reprezentând timpul calendaristic.
  - Aceste diagrame necesită foarte multă muncă și tind să acopere peretii.
- **Recomandare**
  - **Diagrame de bare** pentru se recomandă a fi utilizate pentru a indica **planificarea calendaristică**,
  - **Rețelele de activități** pot fi limitate la indicarea **interdependențelor**.

## 4. Estimarea dimensiunii software-ului

### 4.1 Context

- Principalul motiv pentru care trebuie estimată dimensiunea unui produs software este aceea de a ajuta planificarea dezvoltării produsului.
- **Calitatea** unui **plan de dezvoltare software**, la rândul său, depinde în general de calitatea estimării dimensiunii acestuia.
- Dacă **planul de dezvoltare** este bun:
  - (1) Există o bază solidă pentru **estimarea finanțării** și a **stabilirii personalului** necesar lucrării.
  - (2) Se cunoaște cu exactitate **ce trebuie făcut, când și de către cine**.

(3) De asemenea, se poate estima cât timp va dura și se pot determina dependențele critice sau alte constrângeri.

- Planificarea slabă a software-ului este unul dintre principalele motive pentru care proiectele software au probleme. [Humphrey 89].
  - Cauza cea mai frecventă a unei planificări proaste o reprezintă estimarea deficitară a dimensiunii.
- Practica acceptată în *inginerie, producție și construcții* este de a baza planurile de dezvoltare pe estimări ale dimensiunii produselor.
- Construcția clădirilor oferă un bun exemplu.
  - Constructorii competenți obțin informații detaliate despre tipul de clădire pe care îl dorîți înainte de a vă oferi o estimare a construcției.
  - Apoi fac estimări ale costurilor materialelor și forței de muncă pe baza experiențelor lor anterioare.
  - Folosesc factori istorici pentru articole precum costurile pe metru pătrat pentru cablaje, tâmplărie, tencuieli și vopsire.
  - Constructorii cu experiență pot estima adesea proiecte mari cu o abatere de unul sau două procente din costul final real.
- Gradul de acuratețe și de precizie cu care poate fi planificat un job depinde de ceea ce se cunoaște despre el.
  - (1) În stadiul de **început** sau **înainte de propunere**, există doar o idee generală despre cerințele produsului.
    - De regulă, singura modalitate de a face o estimare este prin analogie cu produsele anterioare.
    - Aceasta este situația cu care se confruntă un constructor atunci când un cumpărător de locuință dorește prețul pentru o vilă cu un etaj și trei dormitoare.
    - Un constructor cu experiență ar putea arăta cumpărătorului mai multe locuințe care se potrivesc acestei descrieri și ar putea oferi prețurile acestora, dar nu ar da un preț pentru o astfel de cerință generală.
  - (2) După **faza de pre-propunere**, se află progresiv mai multe informații despre produsul planificat.
    - Acum se pot face estimări mai precise ale dimensiunii jobului.
    - Din nou, este instructiv să se ia în considerare ce fac constructorii atunci când au nevoie de o estimare competitivă pentru o lucrare de construcții de mai multe milioane de dolari.
    - Lucrează cu specificații arhitecturale detaliate și cu un grup de subcontractanți dovediți.

- Ei calculează dimensiunile liniare ale peretilor interiori și exteriori, metri patrați de podea și tavan și metri cubi de beton.
- Pot determina apoi numărul de șuruburi și cantitatea de metri liniari de cherestea finită necesari, precum și costurile instalațiilor sanitare, a cablajelor și a zidăriei.
- Cu alte cuvinte, când au nevoie de o estimare precisă, definesc proiectul în detaliu.
- Estimările pentru lucrări mari de software pot fi gestionate aproape în același mod.
- Pentru a face o estimare precisă:
  - Se pornește de la specificație de proiectare.
  - Se examinează și se estimează fiecare parte a jobului.
  - Această estimare necesită estimări separate pentru:
    - Fiecare componentă software;
    - Fiecare document major;
    - Cazurile de testare;
    - Planificarea instalării;
    - Conversia fișierelor;
    - Instruirea utilizatorilor.
  - Componentele programului pot avea sub-elemente diferite.
    - Dacă există de dezvoltat ecrane sau interfețe, de generat rapoarte sau de proiectat logică funcțională, acestea trebuie, de asemenea, estimate.
  - Pentru produsele software de mari dimensiuni, există o mulțime de detalii potențial utile.
    - Dacă se dorește o estimare precisă, vor trebui definite toate aceste detalii și trebuieesc luate în considerare fiecare în devizul de calcul.

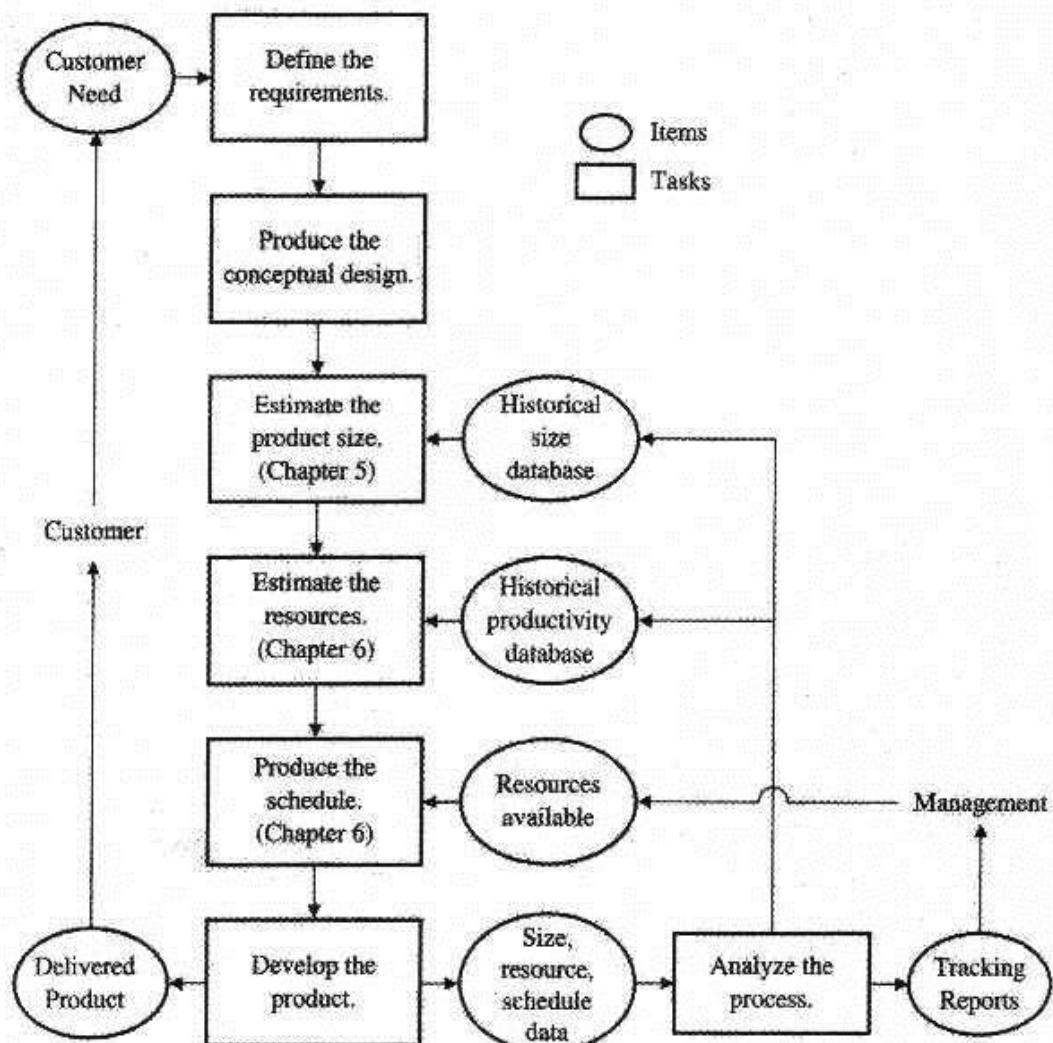
#### 4.2 Cadrul de estimare a dimensiunii software-ului

- Cadrul generalizat de planificare pentru un proiect software este prezentat în figura 4.2.a. [Humphrey97].
  - Sarcinile sunt afișate în dreptunghiuri.
  - Diferitele date, rapoarte și produse sunt afișate în ovale.
- În estimarea dimensiunii software-ului
  - (1) Prima dată trebuieesc definite cerințele.

(2) Apoi se produce un design conceptual.

(3) Se compară elementele acestui design conceptual cu dimensiunea cunoscută a elementelor de program care au fost dezvoltate anterior în companie.

- Acest proces ajută în judecare dimensiunii pieselor noului produs.
  - Cea mai complicată parte a estimării dimensiunii software-ului constă în caracterizarea elementelor produsului nou și în determinarea relației lor cu experiența de dezvoltare istorică.



**Fig.4.2.a.** Cadru general de planificare pentru un proiect software

#### **4.2.1 Relația dimensiune-resurse**

- **Dimensiunea** este un **predictor precis al resurselor** necesare pentru a dezvolta un **produs** [Boehm]
- Relația dintre dimensiune și resursele necesare a condus la dezvoltarea unui număr de modele de cost precum:
  - **COCOMO**;
  - **PriceS**;
  - **SLIM** [Boehm, Park 89, Putnam].
- Deși aceste modele pot fi în general utile, ele trebuie să fie **calibrate** pentru fiecare organizație care le utilizează.
  - Fiecare astfel de organizație trebuie astfel să adune propriile date istorice și să le folosească pentru a stabili factorii de calibrare în modelele sale de cost.
- Toate modelele necesită, de asemenea, ca **dată de intrare** o **estimare a dimensiunii**.
- Ca atare, înainte de a putea fi folosite, trebuie **estimată dimensiunea produsului**.
- Precizia modelelor este astfel limitată de acuratețea estimărilor de dimensiune.
- Deci, chiar și atunci când se utilizează un model de estimare, este nevoie de **o estimare exactă** a **dimensiunii**.

#### **4.2.2 Câteva experiențe de estimare**

- Cercetările recente evidențiază mai multe studii care arată erori de estimare a mărimii de până la 100% [Hihn].
- **Estimările de mărime** care sunt greșite au ca rezultat:
  - (1) **Estimări slabe** ale **resurselor**.
  - (2) **Planificări nerealiste** ale **proiectelor**.
- Acesta este unul dintre motivele majore pentru care grupurile de software au probleme.
  - Fie **nu** au avut o **estimare inițială**, fie cea pe care o aveau era greșită grav.

- În aceste condiții, proiectele demarează adesea cu probleme și nu își revin niciodată.
- **Necunoscutele** de la începutul procesului de dezvoltare a software-ului cauzează inexactități majore de estimare a costurilor.
- **Boehm** afirmă că:
  - (1) Aceste inexactități variază până la 400% în fazele timpurii de fezabilitate.
  - (2) Până la 60% sau mai mult în timpul fazei de cerințe.
  - (3) Ele scad doar la 25% sau mai puțin în timpul și după proiectarea detaliată.
- În mod clar, dacă se face o estimare de dezvoltare prea devreme, când se cunosc mai puține despre produsul care urmează să fie construit, estimarea va fi cu mare probabilitate mai puțin exactă.

#### **4.2.3 Criterii de estimare a dimensiunii software-ului**

- O **metodă de estimare a dimensiunii software-ului** utilizabilă pe scară largă ar trebui să îndeplinească următoarele criterii:
  - (1) Ar trebui să utilizeze **metode structurate și instruibile**.
    - O metodă structurată facilitează formarea și îmbunătățirea procesului.
    - De asemenea, ea permite urmărirea și îmbunătățirea metodei de estimare în sine.
  - (2) Ar trebui să fie **o metodă** care se poate utiliza în timpul **tuturor fazelor de dezvoltare și întreținere** a software-ului.
    - De regulă, metoda se utilizează la estimări de dimensiune la începutul ciclului de dezvoltare pentru a face planurile și angajamente cât mai realiste.
    - În timpul dezvoltării, poate fi necesar ca planurile să fie modificate pentru a fi adaptate la schimbări.
    - În cazul proiectelor mai mari, este, de asemenea, o bună practică să reestimăm periodic dimensiunea produsului și resursele necesare la sfârșitul fiecărei etape majore.
    - Pe măsură ce aceste planuri devin mai precise, ele oferă o bază mai solidă pentru managementul și urmărirea proiectelor.
  - (3) Ar trebui să fie utilizabilă pentru **toate elementele produsului software**.
  - (4) Ar trebui să gestioneze **nu numai cod** ci și:
    - Fișiere;
    - Rapoarte;

- Ecrane;
- Documentație.

(5) Ar trebui să fie, de asemenea, potrivită pentru **toate tipurile de sisteme și aplicații la care se lucrează**, precum și pentru dezvoltarea, îmbunătățirea și repararea de produse noi

(6) Ar trebui să fie adecvată pentru **analiza statistică**.

- O metodă de estimare a mărimii bazată pe statistică va oferi mijloacele pentru a ajusta parametrii de estimare pe baza datelor istorice.

(7) De asemenea, ar trebui să fie adaptabilă la **tipurile joburi pe care ar putea să apară în viitor**.

- Pe măsură ce se acumulează date de estimare și experiență, se poate construi și apoi dezvolta un activ de valoare continuă.

(8) Ar trebui să ofere **mijloace de apreciere a acurateței estimărilor realizate**.

- Ar trebui întotdeauna la final, să fie comparată fiecare estimare cu dimensiunea produsului real rezultat.
- Prin revizuirea acestor comparații, adesea se pot determina cauzele erorilor și astfel crește capabilitatea de ajustare și de îmbunătățire a metodei de estimare.

#### **4.3 Metode de estimare a dimensiunii software-ului**

- Pentru a exprima dimensiunea proiectelor software sunt utilizate diferite **metriki**.
- O **metrică** pentru un proiect SW este de fapt o **metodă** care permite **caracterizarea din punct de vedere cantitativ a produsului**.
  - Cu alte cuvinte, o metrică permite o estimare cantitativă a unui proiect software.
- Există **două categorii de metode** utilizate în **evaluarea proiectelor software**, metode care utilizează metriki specifice

##### **(1) Metode orientate pe dimensiune.**

- Folosesc în esență în estimare ca metrică **numărul de linii sursă de cod**.

##### **(2) Metode orientate pe funcții.**

- Utilizează ca metrică în estimare **complexitatea funcțiilor realizate de produsul software** rezultat în urma procesului de dezvoltare.

#### **4.3.1 Metode de estimare orientate pe dimensiune**

- **Metodele orientate pe dimensiune** folosesc ca metrică **numărul de linii sursă de cod** sau **numărul de instrucțiuni sursă livrate**.
- De obicei, **numărul de linii sursă de cod** este mai mare decât **numărul de instrucțiuni sursă livrate** din cel puțin două motive:
  - (1) Pe parcursul procesului de dezvoltare există posibilitatea ca anumite **module** să fie **codificate de mai multe ori**, într-un proces de perfecționare succesivă până la obținerea versiunii finale, sau **varianta optimă** să fie selectată dintre mai **multe variante elaborate**.
  - (2) Unele părți ale modelelor construite **nu** vor fi incluse în produsul final deoarece există **simulări, experimente sau versiuni de probă** ale produsului dezvoltat
- Unitățile de mărime obișnuite de măsurare a software-ului sunt:
  - **Numărul de linii de cod LOC** (Line Of Code)
  - **Numărul de kilo-linii de cod KLOC** (Kilo-Line Of Code)
  - **Numărul de linii sursă de codului SLOC** (Source Line Of Code)
  - **Numărul de kilo-linii sursă de cod KSLOC** (Kilo-Source Line Of Code)
  - **Numărul de linii sursă livrate DSI** (Delivered Source Instructions)
  - **Numărul de kilo-linii sursă livrate KDSI** (Kilo Delivered Source Instructions)
- Există câteva metode clasice de estimare a dimensiunii proiectelor software bazate pe aceste metriki:
  - (1) **Metoda Wideband-Delphi**;
  - (2) **Metoda Fuzzy-Logic**;
  - (3) **Metoda Componentelor-standard**.

##### **4.3.1.1 Metoda Wideband-Delphi**

- **Metoda de estimare Wideband-Delphi** a fost creată de Rand Corporation și rafinată de Boehm.
  - (1) Metoda necesită **mai mulți experți ingineri** care produc **estimări individuale** ale produsului analizat.
  - (2) Apoi utilizează un **proces Delphi** pentru a **converge** către o **estimare consensuală**.
- **Procedura de estimare** este în esență următoarea [Humphrey 89]:

(1) Grupul de experți primește specificațiile programului și un formular de estimare (fig. 4.3.1.1.a).

(2) Membrii grupului se întâlnesc pentru a discuta obiectivele proiectului, ipotezele și problemele de estimare.

(3) Fiecare membru al grupului, listează anonim sarcinile proiectului și estimează dimensiunea.

(4) Estimările sunt date moderatorului estimărilor, care tabulează rezultatele și le returnează experților.

- După cum se poate deduce din figura 4.3.1.1.a:

- Pentru fiecare expert este identificată doar estimarea personală.
- Estimările celorlalți sunt anonime.
- Se evidențiază și estimarea mediană.

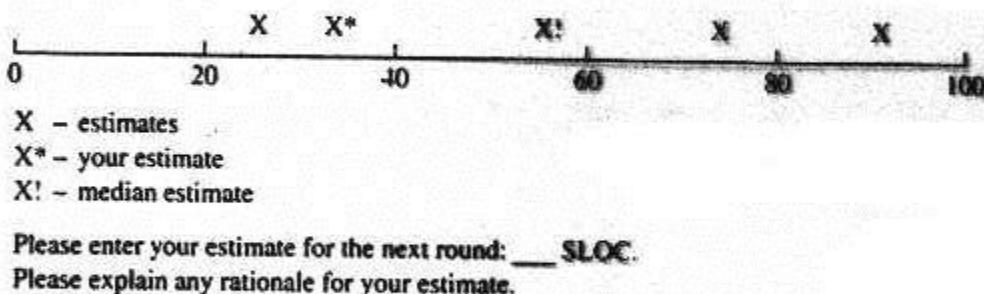
(5) Experții se întâlnesc sub conducerea moderatorului pentru a discuta rezultatele anonime.

(6) Experții pot eventual să revizuiască fiecare activitate pe care le-au definit și estimările lor de dimensiune.

(7) Ciclul se reia la pasul (3) până când estimările converg într-un interval acceptabil.

- Pentru programe de mari dimensiuni, estimatorii pot efectua estimări simultane pentru mai multe componente ale produsului.
- La sfârșitul procesului de estimare, aceste estimări sunt combinate pentru a produce estimarea finală totală.

Project: XYZ  
Estimator: John Doe  
Date: 4/1/94  
Here is the range of estimates from the 1st round:



**FIGURE 5.2**  
Wideband-Delphi

**Fig.4.3.1.1.a.** Formular de estimare pentru metoda Delphi

- **Procesul de estimare** este condus de un **moderator** care trebuie să fie întotdeauna foarte atent să nu dezvăluie niciodată sursa unei estimări individuale.
- Atitudinea potrivită este că **nimeni nu știe** de fapt **răspunsul corect**.
  - Toată lumea are o **vedere parțială**.
  - Scopul **procesului Delphi** este de a **partaja aceste vederi**.
- Încurajând participanții să discute despre sarcinile proiectului, un moderator calificat poate facilita **discuții foarte informative**.
- Persoana care a făcut o estimare deosebit de mare sau scăzută este uneori dispusă să explice de ce a fost aleasă acea valoare.
- Discuțiile aruncă adesea lumină asupra aspectelor problemei și în mod surprinzător, pot convinge pe ceilalți ingineri să-și schimbe estimările.
- Deși confidențialitatea este adesea o problemă, decizia de a dezvălui identitatea oricărui estimator trebuie lăsată exclusiv la latitudinea respectivului estimator.
- Această metodă poate produce **estimări destul de precise**.
- De asemenea, oferă adesea:
  - (1) O **bază solidă** de la care să înceapă jobul.

(2) **Angajamentul inginerilor** de a face treaba la **costul și timpul** pe care l-au pus în estimare.

#### 4.3.1.2 Metoda Fuzzy-logic

- Putnam descrie metoda de estimare cu Fuzzy-logic unde:
  - Estimatorii** evaluează un **produs planificat**.
  - Apoi apreciază aproximativ cum se compară dimensiunea sa cu datele istorice despre produsele anterioare. [Putnam].
- De exemplu, se pot împărți dimensiunile tuturor produselor dezvoltate anterior de organizația dezvoltatoare în categorii și subcategorii de mărime aşa cum rezultă din Tabelul 4.3.1.2.a.

Domeniu \ Subdomeniu	LOC-mic	LOC-mediu	LOC-mare
Foarte mic	1000	2000	4000
Mic	4000	8000	16000
Mediu	16000	32000	64000
Mare	64000	128000	256000
Foarte mare	256000	512000	1024000

**Table 4.3.1.2.a.** Tabelă de domenii Fuzzy-logic

- Cu date suficiente, aceste intervale de dimensiune brută pot fi subdivizate în categorii mai detaliate (subdomenii), după cum se arată în Tabelul 4.3.1.2.b.

<b>Domeniu</b>	<b>Subdomeniu</b>	<b>Foarte mic LOC Ig(LOC)</b>	<b>Mic LOC Ig(LOC)</b>	<b>Mediu LOC Ig(LOC)</b>	<b>Mare LOC Ig(LOC)</b>	<b>Foarte mare LOC Ig(LOC)</b>
<b>Foarte mic</b>	1,148 (3.06)	1,514 (3.18)	2,000 (3.30)	2,630 (3.42)	3,467 (3.54)	
<b>Mic</b>	4,570 (3.66)	6,025 (3.78)	8,000 (3.90)	10,471 (4.02)	13,804 (4.14)	
<b>Mediu</b>	18,197 (4.26)	23,988 (4.38)	32,000 (4.50)	41,687 (4.62)	54,954 (4.74)	
<b>Mare</b>	72,444 (4.86)	95,499 (4.98)	128,000 (5.10)	165,958 (5.22)	218,776 (5.34)	
<b>Foarte mare</b>	288,403 (5.46)	380,189 (5.58)	512,000 (5.70)	660,693 (5.82)	870,964 (5.94)	

**Table 4.3.1.2.b.** Tabelă Fuzzy-logic cu domenii și logaritmi

- Pentru a arăta cum sunt construite aceste intervale, **Tabelul 4.3.1.2.b.** oferă logaritmii zecimali ai intervalelor (în albastru). Se poate observa că:
  - (1) **Intervalle corespunzătoare domeniilor** sunt distanțe uniforme pe o scară logaritmică cu raport 0,60 (vertical).
  - (2) Fiecare diviziune este subdivizată în cinci categorii de subdomenii, cu incrementul  $0,60/5=0,12$  (orizontal)
  - (3) Cele 25 de categorii rezultate oferă baza pentru a face estimări rezonabil de precise.
- **EXEMPLU: Construcția unui tabel simple de estimare Fuzzy-logic.**
- Se presupune că cel mai mic program dezvoltat în organizație a avut 173 LOC și cel mai mare 10.341 LOC.

- Cum s-ar putea împărti această gamă de dimensiuni pe categorii pentru a construi o tabelă Fuzzy-logic ca în Tabelul 4.3.1.2.c?

<b>Subdomeniu Domeniu</b>	<b>Mic</b>	<b>Mediu</b>	<b>Mare</b>
<b>Foarte mic</b>	104 (2.016)	173 (2.238)	288 (2.460)
<b>Mic</b>	288 (2.460)	481 (2.682)	802 (2.904)
<b>Mediu</b>	802 (2.904)	1338 (3.126)	2230 (3.348)
<b>Mare</b>	2230 (3.348)	3719 (3.570)	6202 (3.792)
<b>Foarte mare</b>	6202 (3.792)	10341 (4.014)	17234 (4.236)

**Table 4.3.1.2.c.** Construcția unei tabele simple

- Se poate proceda după cum urmează:
  - (1) Se împarte intervalul în cinci seturi egale pe o scară logaritmică:
    - (1.1) Se iau logaritmii în baza 10 ai lui 173 și 10,341, rezultând  $\lg(173) = 2,238$  respectiv  $\lg(10,341) = 4,014$ .
    - (1.2) Se calculează diferența dintre aceste două valori:  $4,014 - 2,238 = 1,776$ .
    - (1.3) Pentru a obține cinci intervale, se împarte diferența dintre aceste numere la 4, rezultând  $1,776/4 = 0,444$ . Aceasta este incrementalul logaritmice dintre fiecare categorie de interval.
  - (2) Se calculează valorile intervalului:

- Cea mai mică valoare a intervalului este 173 LOC, numărul cu care s-a început, adică logaritmul 2,238.
- Următorul interval are valoarea logaritmă de  $2,238+0,444=2,682$  și o valoare de 481 LOC.
- În mod similar, celealte valori sunt 1338, 3719 și 10341 LOC (crește cu increment logaritm 0,444).

(3) Pentru a împărți fiecare interval în două subdomeni, incrementul logaritm (0,444) se reduce la jumătate, adică  $0,444/2=0,222$ .

- Cel mai mic subinterval va fi  $2,238-0,222=2,016$ , adică 104 LOC.
- Următorul subinterval din categoria foarte mic este  $2,238+0,222=2,460$ , ceea ce înseamnă 288 LOC.

(4) Se continuă în aceeași manieră pentru fiecare dintre valorile intervalelor.

(5) La acest moment sunt cunoscute punctele de sus, punctele de jos și punctele de mijloc ale fiecăruia dintre cele cinci intervale de dimensiuni și se poate construi tabelul (Tabelul 4.3.1.2.c):

- **Estimarea dimensiunii unui proiect software utilizând metoda Fuzzy-logic**
- Pentru a face o estimare este necesar să se judece care dintre categoriile din tabelă se seamănă cel mai mult cu noul proiect.
- Pentru a face acest lucru :
  - (1) Mai întâi se decide în ce **interval de dimensiune grosieră** se încadrează noul proiect.
  - (2) Apoi, **comparând** noul proiect cu **caracteristicile cunoscute ale proiectelor din acest interval** de dimensiuni, se selectează cel mai potrivit **subdomeniu**.
- Deși această tehnică de estimare oferă doar **o judecată grosieră** referitoare la dimensiunea proiectului, ea oferă o **modalitate ordonată de a compara dimensiunile proiectelor planificate** cu dimensiunile celor **dezvoltate anterior**.
- Pentru a obține **rezultate mai precise** această metodă poate fi complementată cu **metoda Wideband-Delphi**.
- În utilizarea acestei **metode de estimare**, sunt importante mai multe considerente:
  - (1) Sunt necesare **datele istorice semnificative** provenind de la un număr mare de proiecte.
    - Trebuie să existe un **număr rezonabil de produse istorice** în fiecare **categorie de mărime**.
  - (2) Este necesar să existe **ceva date despre produsele** din fiecare categorie.
    - În caz contrar, **nu** vor exista exemple cu care să se compare produsele noi care trebuie estimate.

(3) Se pot utiliza orice număr de intervale de dimensiuni, cu condiția ca acestea să acopere întregul interval de dimensiuni așteptate ale proiectului.

(4) Aceste intervale pot fi extinse în sus sau în jos pe măsură ce se obțin date suplimentare.

(5) Cu toate acestea, nu trebuie să se modifice intervalele existente, deoarece în timp, estimatorii devin experti în a judeca programele în funcție de intervalele pe care le-au învățat.

- Când se modifică intervalele, este foarte probabil ca acuratețea estimării să sufere, cel puțin până când se vor reînvăța noile intervale.

(6) Pentru a oferi un număr semnificativ de exemple în fiecare interval de dimensiuni, este nevoie de o cantitate considerabilă de date istorice.

- Dacă nu există cel puțin o distribuție modestă într-un interval dat, un estimator va avea probleme în a judeca dimensiunea relativă a unui nou program.

(7) Până când se acumulează o bază mare de date, este mai întelept ca datele istorice să fie distribuite în cel mult cinci categorii majore.

- Judecările comparative trebuie limitate la acest nivel.
- Pentru a face acest lucru corect, desigur, estimatorii ar trebui să fie familiarizați cu toate programele din baza de date.

- O problemă cu metoda logicii fuzzy este că dimensiunea aplicațiilor majore de programare a crescut istoric cu aproximativ un ordin de mărime la fiecare 10 ani.
- Estimările rezonabil de precise ale celei mai mari categorii de proiect sunt, prin urmare, deosebit de importante.
- Din păcate, orice metodă de dimensionare brută nu va fi probabil de mare ajutor în estimarea unui nou program care este mult mai mare decât orice a fost creat anterior.
- Pentru a face față acestei situații, se recomandă ca noul produs de mari dimensiuni să fie divizat în componente mai mici și să se estimeze fiecare componentă în parte.

#### 4.3.1.3. Metoda Componentelor-standard

- **Metoda de estimare a componentelor standard** este descrisă de Putnam ca o modalitate de a utiliza datele istorice ale unei organizații pentru a face estimări ale dimensiunii programului din ce în ce mai rafinate [Putnam].

(1) Se începe prin a se colecta date despre dimensiunile diferitelor niveluri abstracte de programe care sunt utilizate în organizație, de exemplu, subsisteme, module și ecrane.

(2) Apoi se estimează care dintre aceste componente vor aparține noului program.

(3) În continuare, se apreciază cel mai mare număr care poate fi imaginat în program, cel mai mic număr și numărul cel mai probabil pentru fiecare componentă.

(4) Se combină aceste estimări luând de patru ori numărul cel mai probabil și adunând la acestea numărul cel mai mic cât și numărul cel mai mare pentru fiecare componentă

(5) Se împărte acest total la șase pentru a da numărul estimat al acelei componente.

- **Abaterea standard** a estimării finale va fi de aproximativ o șesime din diferența dintre cel mai mic număr imaginabil și cel mai mare număr imaginabil.
- Sub formă de ecuație, această formulă este:

$$\begin{aligned} \text{Număr estimat} = & [\text{cel mai mic\_număr\_imaginabil} + 4 * (\text{număr probabil}) \\ & + \text{cel mai mare\_număr\_imaginabil}] / 6. \end{aligned}$$

(6) Pe baza acestor numere și pe baza datelor istorice despre componentele standard achiziționate de organizație în timp, se înmulțește numărul estimat de componente cu dimensiunea lor în SLOC.

(7) Se adună numerele calculate pentru a obține dimensiunea totală a produsului.

(8) Nu trebuie uitat ca după finalizarea proiectului să se revizuiască dimensiunile în SLOC ale componentelor standard pe baza analizei post-mortem.

- **Exemplu:** Unele dintre valorile utilizate de Putnam pentru estimarea componentelor standard sunt prezentate ca exemplu în Tabelul 4.3.1.3.a. [Hu95]
- Acest exemplu include fișiere, module, ecrane și rapoarte.
- În baza datelor istorice componentele fișier au fiecare în medie 2535 SLOC.
- În această aplicație, estimatorul a estimat că:
  - Există cel puțin 3 componente ale fișierului.
  - Cel mai mare număr așteptat este 10.
  - Numărul cel mai probabil este 6.
- Folosind formula dată mai sus, se obțineți o estimare de 6,17 componente ale fișierului.
- Deoarece se așteaptă ca fiecare fișier să conțină media istorică de 2535 SLOC, se obține un total de 15.633 SLOC.

- SLOC-urile celorlalte componente sunt calculate în același mod și totalizate pentru a da estimarea finală 46.359 SLOC.

Componente Standard	SLOC per Componentă	S	M	L	X=[S+4*M+L]/6	SLOC
SLOC	1					
Instrucții Obiect	0.28					
Fișiere	2,535	3	6	10	6.17	15,633
Module	932	11	18	22	17.5	16,310
Subsisteme	8,175					
Ecrane	818	5	9	21	10.3	8,453
Rapoarte	967	2	6	11	6.17	5,963
Programe Interactive	1,769					
Programe de fundal (Batch)	3,214					
<b>Total</b>						<b>46,359</b>

**Table.4.3.1.3.a.** Exemplu de estimare a componentelor

### **4.3.2 Metode orientate pe funcții**

- Metodele de estimare orientate pe funcții utilizează metrici bazate pe estimarea complexității funcțiilor implementate de produsul software care urmează să fie dezvoltat.
- Există 2 categorii de metrici orientate pe funcții:
  - (1) **Metrici orientate pe puncte funcționale.**
  - (2) **Metrici orientate pe puncte caracteristice.**
- Există, de asemenea, câteva metode de estimare specifice bazate pe aceste metrici:
  - (1) **Metoda Punctelor-funcționale**
  - (2) **Metoda Punctelor caracteristice.**
  - (3) **Metoda bazată pe proxy-uri.**

#### **4.3.2.1 Metoda Punctelor funcționale**

- **Metoda punctelor funcționale**, inventată de Albrecht la IBM în 1979, este probabil cea mai populară metodă de estimare a dimensiunii aplicațiilor software comerciale.
- Albrecht:
  - (1) A identificat cinci funcții de bază care apar frecvent în dezvoltarea de software comercial.
  - (2) Le-a clasificat în funcție de **complexitatele lor relative de dezvoltare**.
- Definițiile acestor **cinci funcții de bază** sunt următoarele: [Jones]
  - (1) **Intrări:**
    - Intrările sunt ecrane sau formulare prin care utilizatorii umani ai unei aplicații sau ai altor programe adaugă date noi sau actualizează datele existente.
    - Dacă un ecran de intrare este prea mare pentru un singur afișaj normal (de obicei 80 de coloane pe 25 de linii) și se trece la un al doilea ecran, setul contează ca o singură intrare.
    - Doar intrările care necesită o prelucrare unică sunt cele care sunt luate în considerare.
  - (2) **Ieșiri:**
    - Ieșirile sunt ecrane sau rapoarte pe care aplicația le produce pentru uz uman sau pentru alte programe.
    - Doar ieșirile care necesită procesare separată sunt considerate unități de numărat.

- De exemplu, într-o aplicație de salarizare, o funcție de ieșire care a creat, să zicem, 100 de cecuri contează ca o singură ieșire.

**(3) Interogări:**

- Interogările sunt ecrane care permit utilizatorilor să interogheze o aplicație și să ceară asistență sau informații.

**(4) Fișiere de date:**

- Fișierele de date sunt colecții logice de înregistrări pe care aplicația le modifică sau le actualizează.
- Un fișier poate fi, de exemplu:
  - Un fișier normal, cum ar fi un fișier disc.
  - O ramură a unei baze de date ierarhice.
  - Un tabel într-o bază de date relațională.
  - O cale printr-o bază de date de rețea.

**(5) Interfețe:**

- Interfețele sunt fișiere partajate cu alte aplicații și includ:
  - Fișiere pe bandă sau disc de intrare sau de ieșire.
  - Baze de date partajate.
  - Liste de parametri.
- **Estimarea dimensiunii unui proiect software utilizând metoda Punctelor funcționale**
- Pentru a face o estimare utilizând metoda punctelor-funcționale:
  - (1) Se examinează cerințele și se determină numerele fiecărui tip de funcție de care programul va avea probabil nevoie.
  - (2) Se introduceți aceste numere în zona Numere de bază din Tabelul 4.3.2.1.a.
  - (3) Se înmulțesc numerele de bază cu zona Ponderi pentru a produce Numărul total de puncte funcționale pentru fiecare categorie.
  - (4) Se însumează valorile obținute.
  - (5) Numărul obținut este totalul neajustat de puncte funcționale ale software-lui estimat.

Numere de Bază	Tip Functional	Pondere	Total
	Intrări	x 4	
	Ieșiri	x 5	
	Interogări	x 4	
	Fișiere Logice	x 10	
	Interfețe	x 7	
	<b>Total Neajustat</b>		

**Table 4.3.2.1.a.** Categorii de puncte funcționale

- În tabelul 4.3.2.1.b este prezentat un exemplu de estimare bazat pe puncte funcționale [Hu95]
- Analiza cerințelor unei aplicații a condus la următoarele estimări:
  - 8 intrări;
  - 12 ieșiri;
  - 4 interogări;
  - 2 fisiere logice;
  - 1 interfață.
- Pe baza algoritmului de estimare propus, se înmulțesc aceste numere cu ponderile specifice pentru a obține valorile din coloana Total.
  - De exemplu, cele 8 intrări sunt înmulțite cu o pondere 4, dând un total de 32 (Tabelul 4.3.2.1.b.).
- Se însumează valorile obținute pentru fiecare tip de funcție
- Totalul neajustat de puncte-funcționale are valoarea 135.

Numere de Bază	Tip Functional	Pondere	Total
8	Intrări	8 x 4	32
12	Ieșiri	12 x 5	60
4	Interogări	4 x 4	16
2	Fișiere Logice	2 x 10	20
1	Interfețe	1 x 7	7
	<b>Total Neajustat</b>		<b>135</b>

**Table 4.3.2.1.b.** Exemplu de categorii de puncte-funcționale

- Jones, în lucrarea publicată pe acest subiect, sugerează că acest total de puncte-funcție neajustat să fie ajustat de un **multiplicator de complexitate** bazat pe **14 factori de influență**.
- În Tabelul 4.3.2.1.c sunt prezentate ca exemplu cei **14 factori de influență** propuși.
- Factorii de influență pot lua valori întregi între 0 la 5, în funcție de gradul în care se consideră că un anumit factor se încadreză de la foarte simplu la foarte complex respectiv de la foarte scăzut la foarte mare.
- Deoarece există 14 factori și valorile acestora pot varia fiecare de la 0 la 5, suma totală a\_factorilor\_de\_influență variază apoi de la 0 la 70.
- Formula de calcul a multiplicatorului de complexitate este:  

$$\text{Multiplicator de complexitate} = 0,65 + 0,01 * (\text{Suma}_\text{factorilor}_\text{de}_\text{influență})$$
- Din formula de mai sus, se poate observa că valoarea multiplicatorului complexității variază de la 0,65 (când Suma=0) la 1,35 (când Suma=70)
- Astă înseamnă că influența multiplicatorului de complexitate variază de la minus 35 la sută la plus 35 la sută.

- Pentru a finaliza estimarea dimensiunii în aceste noi condiții:
  - (6) Se estimează impactul fiecărui factor de influență cu valori de la 0 la 5.
  - (7) Se însumează valorile atribuite factorilor și se obține Suma\_factorilor\_de\_influență.
  - (8) Se calculează multiplicatorul de complexitate pe baza formulei:  

$$\text{Multiplicator de complexitate} = 0,65 + 0,01 \cdot (\text{Suma_factorilor_de_influență})$$
  - (9) Se înmulțește Totalul neajustat de puncte funcționale cu Multiplicatorul de complexitate pentru a obține numărul de puncte de funcție pentru software-ul estimat.
- Pentru exemplul de estimare prezentat mai sus:
- Se presupune că au fost selectate următoarele valori pentru factorii de influență pentru proiectul în analizat (Tabelul 4.3.2.1.c)

Nr.crt	Factor	Influence
		<b>Influence</b> 0 = non influență 1 = influență mică 2 = influență moderată 3 = influență medie 4 = influență semnificantivă 5 = influență puternică
1.	Comunicații de date (Data communications)	2
2.	Funcții distribuite (Distributed functions)	0
3.	Obiective de performanță (Performance objectives)	3

4	Configurații utilizate din greu (Heavily used configuration)	3
5.	Rate de transfer (Transaction rate)	4
6.	Date de intrare online (On line data entry)	4
7.	Eficiență utilizatorului final (End-user efficiency)	3
8.	Aducere la zi online (On line update)	2
9.	Procesare complexă (Complex processing)	3
10.	Reutilizare (Reusability)	2
11.	Instalare facilă (Installation ease)	3
12.	Operare simplă (Operational ease)	4
13.	Site-uri multiple (Multiple sites)	5
14.	Facilitarea modificărilor (Facilitate change)	3
	<b>Suma factorilor de influență (Sum of influence factors) =</b>	<b>41</b>
	<b>Multiplicatorul de complexitate (Complexity Multiplier) = <math>0.65+0.01(\text{Suma factorilor de influență})</math></b>	<b>1.06</b>
	<b>Puncte funcționale (Function points) =</b> Multiplicatorul de complexitate*Total neajustat de puncte funcționale	<b><math>135*1.06=143</math></b>

**Table 4.3.2.1.c.** Factori de influență în metoda punctelor funcționale (un exemplu)

- Factorii de influență selectați din Tabelul 4.3.2.1.c. sunt utilizati pentru a calcula valoarea Multiplicatorului de complexitate de 1,06.
- Se înmulțește totalul de puncte funcționale neajustat de 135 cu acest factor și se obține un total de 143 de puncte de funcție.
- În realitate, oricât de utile sunt, punctele funcționale nu sunt pe deplin satisfăcătoare din două motive.
  - (1) În primul rând, ele nu pot fi măsurate direct.
  - (2) În al doilea rând, nu sunt sensibile la deciziile de implementare.
- Jones a abordat prima problemă prin producerea unui număr de factori de conversie care permit determinarea LOC-urilor pornind de la estimarea proiectului în puncte funcționale [Jones].
- A doua problemă, cea legată de independență de implementare, este mai discutabilă.
- În timp ce independența implementării este un avantaj în comparațiile între limbaje de programare sau sisteme de calcul, este un dezavantaj în estimările costurilor de dezvoltare.
- Costurile de dezvoltare sunt de obicei sensibile la:
  - (1) Limbajul de implementare;
  - (2) Stilul de design;
  - (3) Domeniul de aplicare.
- Deci, fie metoda de estimare trebuie să le ia în considerare, fie trebuie făcută o altă compensație.
- Metoda punctelor-funcționale este încă de interes și continuă să fie rafinată.
  - Pe internet pot fi obținute copii ale celor mai recente ghiduri și standarde ale **Grupului Internațional de Utilizatori ai Punctelor Funcționale** (IFPUG) [Sprouts, Zwanzig].

#### **Rezumatul metodei punctelor funcționale**

- Punctele funcționale „de bază” (BFP):  $4(EI) + 5(EO) + 4(EQ) + 10(ILF) + 7(EIF)$
- (cu  $\pm 35\%$  ajustarea complexității)
- Puncte funcționale neajustate (UFP): cele cinci atribute pot fi ponderate ca simple, medii sau complexe (Tabelul 4.3.2.1.d.)

Attribute	Complexitate			Total
	Simplă	Medie	Complexă	
Intrări externe (External Inputs)	3	4	6	
Ieșir externe (External Outputs)	4	5	7	
Interrogări externe (External inQuerries)	3	4	6 (or 7)	
Fișiere interne (Internal Files)	7	10	15	
Interfețe externe (External InterFaces)	5	7	10	

**Table 4.3.2.1.d.** Ponderile atributelor punctelor funcționale

- Puncte funcționale ajustate (AFP): UFP ( $0,65 + [0,01(SIF)]$ )
- (SIF este Suma factorilor de influență: Suma a 14 factori, cotați de la 1 la 5 pentru influență [0 - Niciuna, 1 - Mică, 2 - Moderată, 3 - Medie, 4 - Semnificativă, 5 - Puternică]; Evaluările se definesc pentru fiecare Factor) (Tabelul 4.3.2.1.e.)

<b>14 Factors:</b>	
1. Comunicații de date (Data Communications)	2. Procesare de date distribuită (Distributed Data Processing)
3. Obiective de performanță (Performance Objectives)	4. Configurații utilizate din greu (Heavily-Used Configuration)
5. Rate de transfer (Transaction Rate)	6. Intrări de date online (On-Line Data Entry)
7. Eficiență utilizatorului final (End-user Efficiency)	8. Aducere la zi online (On-Line Update)
9. Procesare complexă (Complex Processing)	10. Reutilizare (Reusability)
11. Conversie și instalare facilă (Conversion and Installation Ease)	12. Operare facilă (Operational Ease)
13. Utilizare în site-uri multiple (Multiple Site Usage)	14. Facilitarea modificării (Facilitate Change)

**Table 4.3.2.1.e.** Factori de influență

#### 4.3.2.2 Conversia punctelor funcționale în SLOC

- Uneori apare necesitatea conversiei SLOC-urilor în puncte funcționale sau invers.
- Cele mai multe modele de cost software, acceptă la intrare SLOC. Ca atare dacă estimarea dimensiunii s-a realizat în puncte funcționale (sau variante ale acestora), acestea trebuie convertite în SLOC, deoarece algoritmii modelelor de cost se bazează pe SLOC.
- Situația opusă apare în modelul CHECKPOINT® în care intrările SLOC trebuie convertite în puncte funcționale.

- Acest proces de conversie este uneori numit „backfiring”.
- Pentru a ajuta acest proces de conversie, au fost dezvoltate tabele care ecivalează la punctele funcționale cu SLOC pentru diferite limbaje de programare.
- Tabelul 4.3.2.2.a prezintă un astfel de exemplu.
- Relația dintre nivelul limbajului de programare și productivitate nu este liniară.

Language	Default SLOC / UFP	Language	Default SLOC / UFP
Access	38	Jovial	107
Ada 83	71	Lisp	64
Ada 95	49	Machine Code	640
AI Shell	49	Modula 2	80
APL	32	Pascal	91
Assembly - Basic	320	PERL	27
Assembly - Macro	213	PowerBuilder	16
Basic - ANSI	64	Prolog	64
Basic - Compiled	91	Query – Default	13
Basic - Visual	32	Report Generator	80
C	128	Second Generation Language	107
C++	55	Simulation – Default	46
Cobol (ANSI 85)	91	Spreadsheet	6
Database – Default	40	Third Generation Language	80
Fifth Generation Language	4	Unix Shell Scripts	107
First Generation Language	320	USR_1	1
Forth	64	USR_2	1
Fortran 77	107	USR_3	1
Fortran 95	71	USR_4	1
Fourth Generation Language	20	USR_5	1
High Level Language	64	Visual Basic 5.0	29
HTML 3.0	15	Visual C++	34
Java	53		

**Table 4.3.2.2.a.** Conversia SLOC/PF

- Deși astfel de tabele sunt utile și adesea necesare, se recomandă ca ele să fie utilizate cu prudență.

#### 4.3.2.3 Metoda Punctelor caracteristice

- Metrica bazată pe puncte caracteristice a fost propusă de C.A. Jones. [C.A. Jones, „A short History of Function Point Method and Feature Points Software Productivity” Research, Inc., Burlington, MA, iunie 1986].
- Metoda nu este foarte diferită de metoda punctelor funcționale.

- De fapt, există trei diferențe:

- (1) **Punctele-funcționale** sunt înlocuite cu **puncte caracteristice**.
- (2) **Ponderile** asociate au valori diferite.
- (3) Se adaugă un nou parametru: **algoritmi** cu pondere 3.

### **Estimarea dimensiunii unui proiect software utilizând metoda Punctelor caracteristice**

- (1) Se estimează **numărul de bază** asociat fiecărui punct caracteristic.
- (2) Se înmulțește fiecare număr cu **ponderea** sa.
- (3) Se calculează **Totalul** pentru fiecare punct caracteristic.
- (4) Se calculează **totalul general** prin însumarea totalurilor parțiale (Tabelul 4.3.2.3.a.).
  - Există versiuni ale metodei în care ponderile diferă în funcție de complexitatea programului (simplu, mediu, complex).
  - Totalul determinat poate fi ajustat într-un mod similar ca la metoda punctelor funcționale.
- (5) Se stabilesc ponderile celor 14 factori de influență.
- (6) Se calculează valoarea factorului de ajustare a complexității **SIF** (Suma factorilor de influență) ca sumă a celor 14 factori de influență.
- (7) Se calculează valoarea **totalului ajustat** conform formulei:

$$\text{Total ajustat} = \text{Total} * (0,65 + 0,01 * \text{SIF})$$

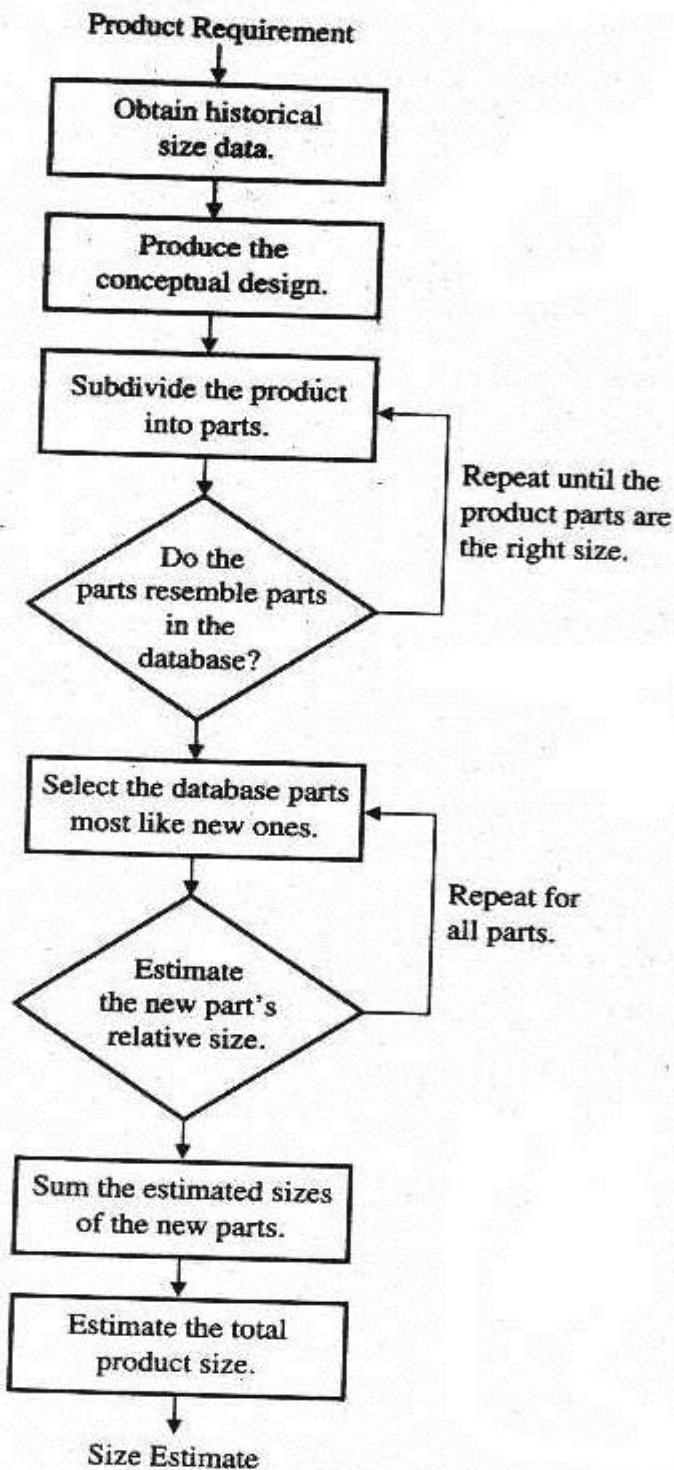
Tip Caracteristic	Numere de bază	Pondere	Total
Intrări Externe		4	
Ieșiri Externe		5	
Interogări Externe		4	
Fișiere Interne		7	
Interfaces Externe		7	

Algoritmi		3	
<b>Total</b>			

**Table 4.3.2.3.a.** Puncte caracteristice și pondările asociate

#### 4.3.2.4 Estimarea bazată pe Proxy-uri

- În planificarea unui proiect, estimările sunt în general necesare încă dinainte de a începe dezvoltarea.
- În această etapă incipientă, cerințele pot fi înțelese, dar în general se știe puțin despre produs în sine.
- Problema estimării se referă în esență la a prezice dimensiunea finală probabilă a produsului ce urmează să fie dezvoltat.
- Deoarece nimeni nu poate să cunoască dinainte cât de mare va fi un produs planificat, estimarea dimensiunii software-ului va fi în întocmire un proces incert.
- În realitate apare însă necesitatea stringentă de a face o estimare cât mai exactă posibilă.
- În general, toate metodele de estimare utilizează date despre programe similare dezvoltate anterior pentru a stabili o bază de evaluare a dimensiunii noului program.
- Aceasta sugerează un proces de estimare generalizat care este prezentat în Fig.4.3.2.4.a. [Humphrey95].



**Fig.4.3.2.4.a.** Cadrul general al estimării dimensiunii

### Proxy-uri

- Se reia în acest context din nou exemplul **construcțiilor**.

- În construcția de locuințe, numărul de metri pătrați de spațiu de locuit oferă o bază pentru estimarea costurilor de construcție.
- Puțini oameni, însă, pot vizualiza casa pe care și-o doresc în termeni de metri pătrați.
- Aceeași problemă apare și la produsele **software**.
  - Dacă s-ar putea judeca cu exactitate numărul de LOC pentru un produs software planificat, s-ar putea realiza estimări de foarte bună calitate pentru dezvoltare.
  - Din păcate, puțini oameni pot judeca în mod direct câte LOC sunt necesare pentru a îndeplini o cerință software.
- Ar fi nevoie de un **proxy** care să coreleză dimensiunea produsului cu funcțiile pe care estimatorul le poate vizualiza și descrie.
  - Un **proxy** este un substitut sau “ceva care ține locul a altceva”.
- Presupunând că proxy-urile sunt mai ușor de vizualizat decât aplicarea metodelor de măsurare curente a dimensiunii unui produs, un proxy poate ajuta substantial la estimarea dimensiunii produsului.
- Prin urmare, proxy-urile pot fi utilizate pentru a estima LOC-urile unui produs.
- Exemple de proxy sunt:
  - Obiecte;
  - Ecrane;
  - Dosare;
  - Scripturi;
  - Puncte funcționale;
  - Puncte caracteristice.

### **Selectarea unui proxy**

- Criteriile pentru alegerea unui proxy bun sunt următoarele:
  - (1) **Măsura mărimii unui proxy ar trebui să fie strâns legată de efortul necesar pentru dezvoltarea produsului.**
    - Referitor la efortul de dezvoltare.
    - Pentru ca un proxy să fie util, el trebuie să aibă de maniera demonstrabilă, o relație strânsă cu resursele necesare dezvoltării produsului.
    - Astfel, estimând dimensiunea proxy-ului, se poate judeca cu exactitate dimensiunea jobului.

- Eficacitatea unui proxy se determină obținând date istorice despre un număr de produse care au fost dezvoltate în organizație, comparând valorile proxy cu costurile de dezvoltare.
- Folosind metoda corelării, se poate determina apoi dacă proxy-ul în cauză sau un alt proxy este un predictor mai bun al dimensiunii produsului sau al efortului de dezvoltare.
- Dacă proxy-ul nu trece acest test, nu are rost să fie folosit.

**(2) Conținutul proxy al unui produs ar trebui să fie numărabil automat.**

- Deoarece sunt necesare date istorice despre proxy pentru a face noi estimări, este de dorit să existe o cantitate mare de astfel de date.
- Acest lucru necesită ca datele să fie numărabile automat, ceea ce sugerează, la rândul său, că proxy-ul trebuie să fie o entitate fizică care să poată fi definită cu precizie și identificată algoritmice.
- Dacă nu se poate contoriza automat conținutul în proxy-uri al unui program, nu există o modalitate convenabilă de a obține în mod fiabil datele statistice care sunt necesare pentru a genera factori de estimare personalizați pentru procesul specific de dezvoltare și pentru stilul de proiectare.

**(3) Proxy-ul ar trebui să fie ușor de vizualizat la începutul unui proiect.**

- Dacă proxy-ul este mai greu de vizualizat decât numărul de ore de programare necesare pentru realizarea dezvoltării, se pot estima direct orelele și să nu vă faceți griji pentru proxy.
- Utilitatea unui proxy depinde astfel de gradul în care el ajută vizualizarea dimensiunii proiectului de realizat.
- Aceasta, la rândul său, depinde de antecedentele și preferințele estimatorului. Deci, probabil că nu va exista un cel mai bun proxy pentru toate scopurile.
- Cu date istorice adecvate, se pot utiliza chiar mai multe proxy-ri diferite într-o singură estimare.
- Regresiile multiple pot fi utile în acest scop.

**(4) Proxy-ul ar trebui să fie personalizabil pentru nevoile speciale ale organizațiilor care îl utilizează.**

- O mare parte din dificultățile pe care le întâmpină organizațiile cu diferite metode de estimare provine din încercările lor de a utiliza datele provenind de la un grup de dezvoltare pentru planificarea dezvoltărilor de către un alt grup.

- Deși aceasta este în principal o problemă care apare la utilizarea modelelor de resurse, ea se aplică și în cazul utilizării proxy-urilor. Prin urmare, este important să se culeagă și să se utilizeze date care sunt relevante pentru proiectul estimat.
- Acest lucru sugerează că organizațiile mari ar trebui să aibă baze de date de dimensiuni și resurse pentru fiecare tip de produs software major.
- În mod similar, fiecare programator individual ar trebui să adune și să utilizeze date despre propria sa activitate.
- Dacă proxy-urile selectate nu sunt potrivite pentru o parte din activitățile care se desfășoară, atunci pe baza datelor istorice, ar trebui identificați alți proxy mai potriviti și factorii lor de estimare.

**(5) Proxy-ul ar trebui să fie sensibil la orice variații de implementare care afectează costurile sau efortul de dezvoltare.**

- Această problemă implică un compromis dificil.
- Proxy-urile care sunt cel mai ușor vizualizate la începutul unui proiect sunt entitățile aplicației, cum ar fi intrările, ieșirile, fișierele, ecranele și rapoartele.
- Din păcate, o bună estimare a dezvoltării necesită entități care au legătură strânsă cu produsul care urmează să fie construit și cu forța de muncă necesară pentru a-l construi.
- Această nevoie necesită, de asemenea, ca datele despre proxy și dimensiunea produsului să fie disponibile pentru fiecare limbaj de implementare, pentru fiecare stil de proiectare și categorie de aplicație.
- Prin urmare, este esențial ca limbajele, stilurile de design și categoriile de aplicații care urmează să fie utilizate într-un proiect să fie reprezentate în datele utilizate pentru calcularea factorilor de estimare care trebuie utilizati.

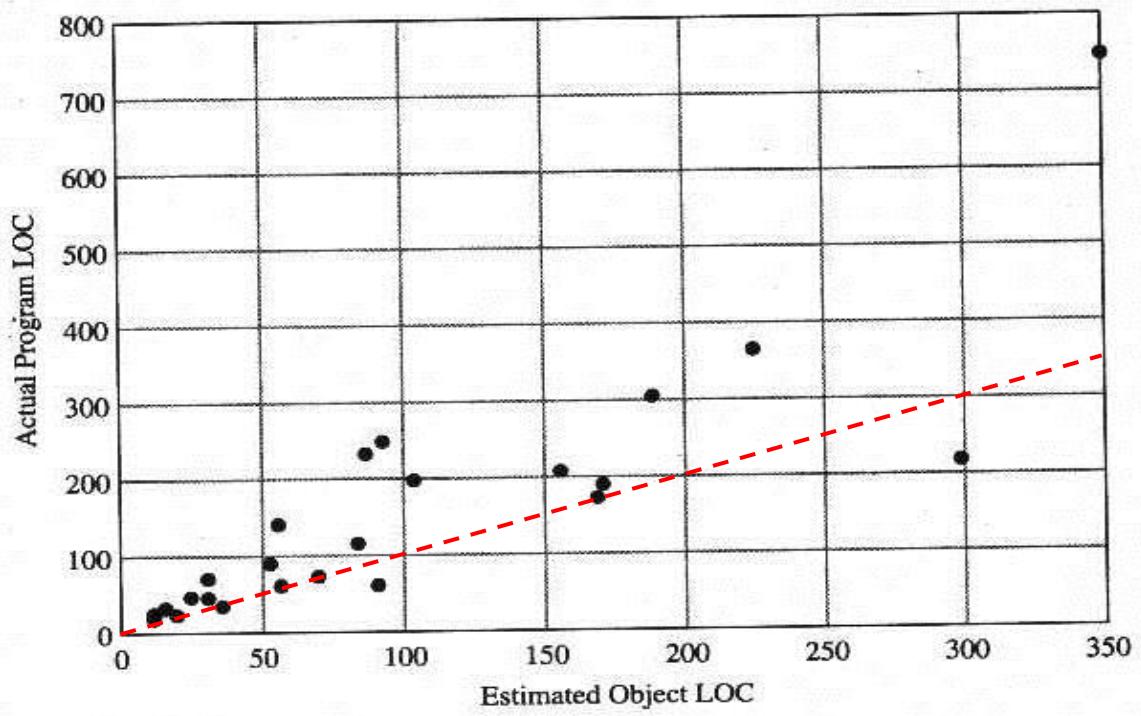
### **Proxy-uri posibile**

- Există mai multe tipuri potențiale de proxy care ar putea îndeplini criteriile prezentate anterior.
- Metoda punctelor funcționale este un candidat evident deoarece este utilizată pe scară largă.
  - Mulți oameni au considerat că punctele funcționale sunt utile pentru estimarea resurselor, așa că ar trebui luate în considerare.
- Cu toate acestea, după cum s-a menționat mai devreme în acest capitol, principalul dezavantaj al punctelor funcționale este acela că nu sunt contorizabile direct în produs finit.
- Alte proxy-uri posibile sunt:

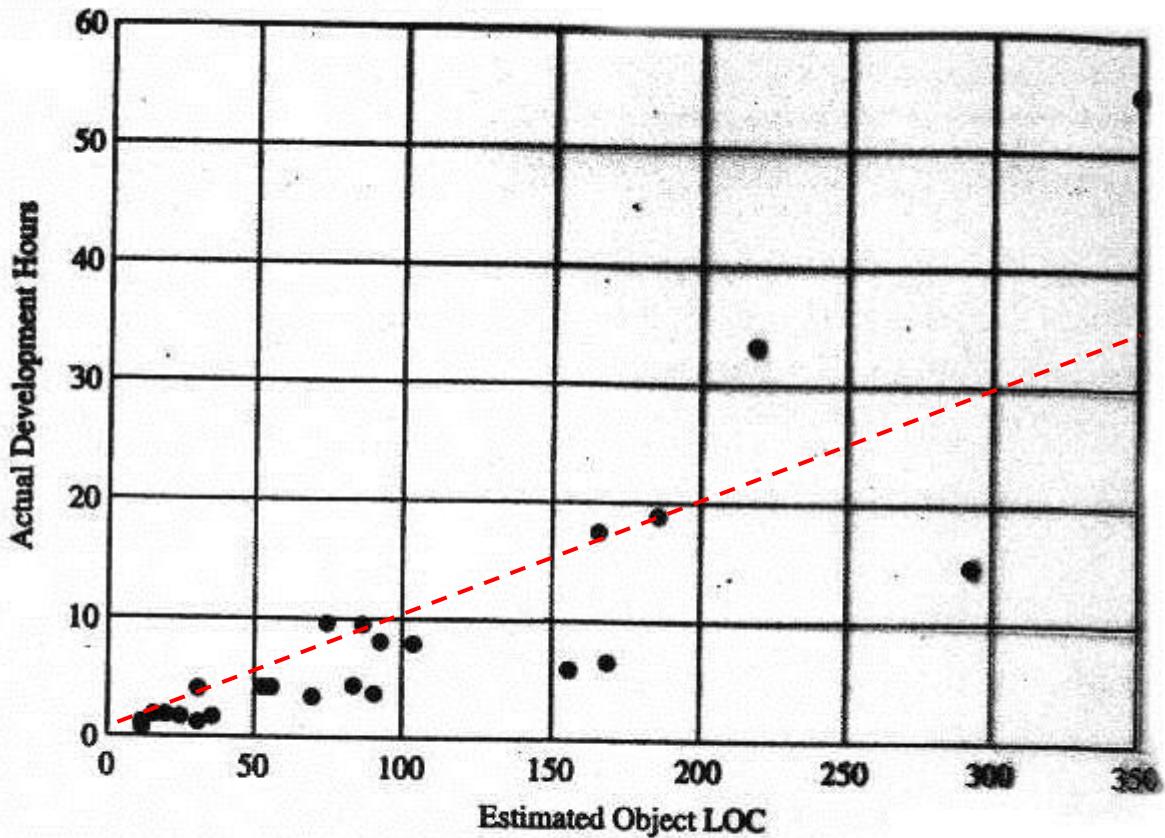
- Obiecte;
- Ecrane;
- Fișiere;
- Scripturi;
- Capitolele de documente;
- etc.

### **Objectele ca Proxi-uri**

- Principiile proiectării orientate pe obiecte sugerează că **obiectele** ar fi bune pentru estimarea proxy-urilor.
- În timpul **analizei și proiectării inițiale a programului**, **entitățile aplicației** sunt folosite ca bază pentru selectarea **obiectelor de sistem**. [Booch, Coad, Shlaer, Wirfs-Brock].
  - O entitate de aplicație este ceva care există în mediul aplicației.
- Într-un **proiect bazat pe obiecte**, se vor selecta **obiectele de program** care vor modela aceste **entități din lumea reală**.
  - Aceasta înseamnă că obiectele de cel mai înalt nivel ale produsului de dezvoltat pot fi vizualizate în timpul **analizei cerințelor**.
  - Astfel, **obiectele** pot îndeplini una dintre cerințele de bază pentru un proxy.
- Humprey a studiat în mod experimental relația dintre LOC-urile estimate și cele reale pentru proxy (fig. 4.3.2.4.b) și relația dintre LOC-urile obiect și orele reale de dezvoltare (fig. 4.3.2.4.c).
  - În ambele cazuri a obținut practic o dependență liniară.
  - În ambele cazuri, corelația și semnificația sunt foarte mari.



**Fig.4.3.2.4.b.** LOC-uri estimate pentru obiecte versus LOC-uri de program actuale



**Fig.4.3.2.4.c.** LOC-uri estimate pentru obiecte versus Ore de dezvoltare actuale

- Concluziile studiului:
  - (1) Deoarece **LOC-urile totale** de program se corelează cu **orele de dezvoltare** și **obiectele** se corelează foarte bine cu **LOC-urile totale de program**, **obiectele** îndeplinesc astfel cerința ca ele să fie strâns legate de efortul de dezvoltare.
  - (2) Obiectele sunt entități fizice care pot fi contorizate automat.
  - (3) Presupunând că se utilizează o metodologie de proiectare orientată pe obiecte și un limbaj de implementare și se adună date statistice despre dimensiunea obiectelor dezvoltate, obiectele vor îndeplini toate criteriile pentru un proxy.

## **Utilizarea obiectelor ca proxy în estimarea dimensiunii proiectelor software**

- Pentru a utiliza proxy-uri în estimarea dimensiunii proiectelor se procedează după cum urmează:
  - (1) Se împart datele istorice acumulate referitoare la obiecte în categorii și intervale de dimensiuni.
    - Din nou, exemplul din construcții ilustrează această activitate.
    - Atunci când estimează metri pătrați dintr-o casă, un constructor trebuie să se gândească la camere mari și camera mici și la câte dintre acestea dorește cumpărătorul.
    - Este important ca constructorul să se gândească în ansamblu la categoriile de camere; de exemplu, o baie mare ar fi mult mai mică decât chiar și o cameră mică de familie.
  - (1.1) Pentru a estima câte obiecte LOC conține un produs nou, trebuie mai întâi să se grupeze obiectele în **categorii de obiecte funcționale**.
    - Exemplu de categorii de obiecte funcționale:
    - C++: calcul, date, I/O, logică, configurare, text.
    - PASCAL: Control, Display, File, Logic, Print, Text.
  - (1.2) Folosind metoda fuzzy-logic a lui Putnam, se împart aceste categorii de obiecte funcționale în **intervale de obiecte** foarte mici, mici, medii, mari, foarte mari. [Putnam].
    - În partea de sus a Tabelului 4.3.2.4.c [Hu95] obiectele C++ sunt listate în **șase categorii**.
    - Aceste date arată că un obiect C++ de dimensiuni medii sau medii pentru calcule matematice ar avea aproximativ 11 LOC, în timp ce un obiect de calcul foarte mare ar avea 54 LOC.
- (2) Apoi se trece la etapa de estimare judecând:
  - (2.1) De câte obiecte din fiecare categorie este nevoie pentru noul produs.
  - (2.2) Mărimea relativă a fiecărui obiect din categoria sa.
    - Mărimea obiectului este, de asemenea, o chestiune de stil.
    - Unii oameni preferă să grupeze multe funcții în câteva obiecte; alții preferă să creeze multe obiecte cu relativ puține funcții.
    - Datele utilizate în scopuri de estimare ar trebui să reflecte cu acuratețe limbile și practicile pe care se utilizează în dezvoltarea software-ului.
    - Prin urmare, este o idee bună să utilizeze datele despre dimensiunea obiectelor provenind din activitatea proprie a organizației.

(2.3) Datorită existenței variațiilor mari referitoare la numărul de metode sau proceduri per obiect, este util să se **normalizeze** obiectele după **numărul lor de metode** (sau proceduri) și să se judece dimensiunea obiectului bazat pe metode.

- Datele din Tabelul 4.3.2.4.c sunt furnizate în LOC pe metodă.
- În termeni de Pascal, aceasta ar fi LOC per procedură;
- De exemplu, dacă un **obiect de tip text** Pascal de dimensiune medie care are patru metode, ar avea o dimensiune medie de  $16,48 \times 4$ , sau aproximativ 66 LOC.

**TABLE 5.7 OBJECT CATEGORY SIZES IN LOC PER METHOD**

<b>C++ Object Size in LOC per Method</b>					
<i>Category</i>	<i>Very Small</i>	<i>Small</i>	<i>Medium</i>	<i>Large</i>	<i>Very Large</i>
Calculation	2.34	5.13	11.25	24.66	54.04
Data	2.60	4.79	8.84	16.31	30.09
I/O	9.01	12.06	16.15	21.62	28.93
Logic	7.55	10.98	15.98	23.25	33.83
Set-up	3.88	5.04	6.56	8.53	11.09
Text	3.75	8.00	17.07	36.41	77.66

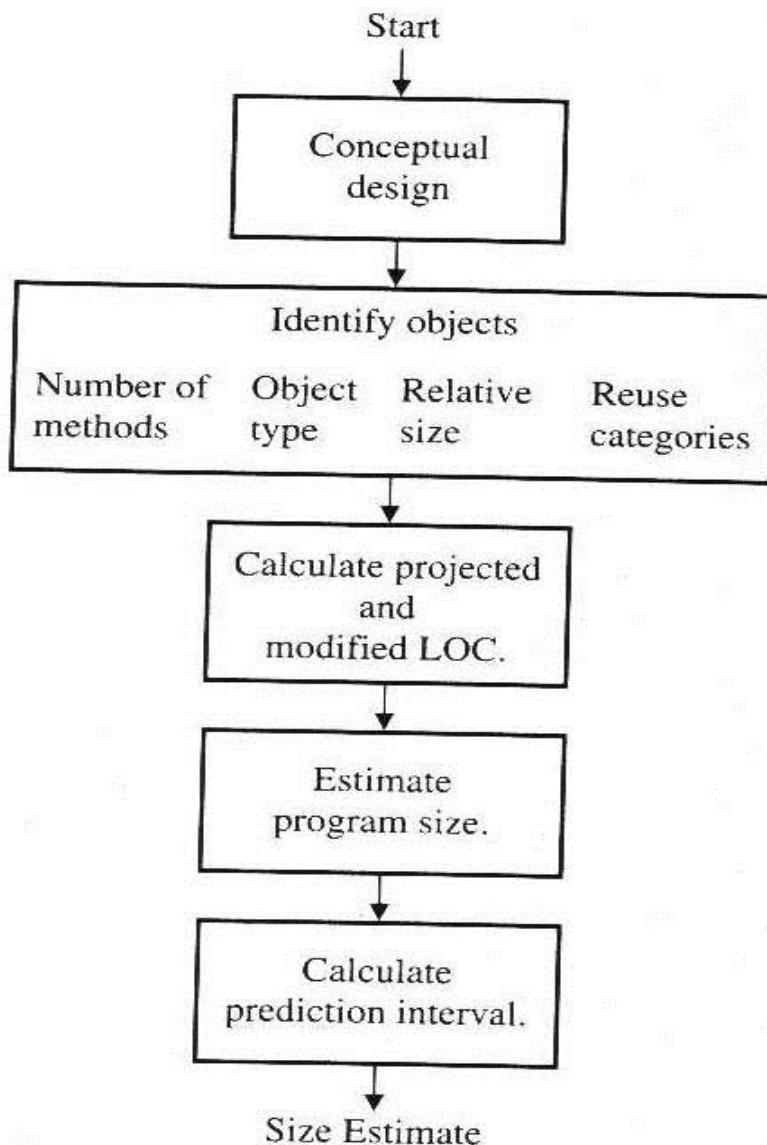
<b>Object Pascal Object Size in LOC per Method</b>					
<i>Category</i>	<i>Very Small</i>	<i>Small</i>	<i>Medium</i>	<i>Large</i>	<i>Very Large</i>
Control	4.24	8.68	17.79	36.46	74.71
Display	4.72	6.35	8.55	11.50	15.46
File	3.30	6.23	11.74	22.14	41.74
Logic	6.41	12.42	24.06	46.60	90.27
Print	3.38	5.86	10.15	17.59	30.49
Text	4.63	8.73	16.48	31.09	58.62

**Fig.4.3.2.4.c.** Dimensiuni pentru categorii de obiecte exprimate în LOC per metod (C++, PASCAL)

- (3) Estimarea dimensiunii totale a proiectului în **LOC** presupune:
  - (3.1) Se stabilește ce categorii de obiecte sunt necesare.

- (3.2) Pentru fiecare obiect se stabilește dacă se încadrează în intervalul de dimensiuni foarte mic, mic, mediu, mare sau foarte mare.
- (3.3) Se estimează câte metode va conține fiecare.
- (3.4) Se înmulțește numărul de linii de cod cu numărul de metode pentru fiecare din obiectele implicate
- (3.5) Se calculează mărimea în LOC prin însumarea valorilor obținute.

- Humphrey a dezvoltat metoda de estimare a dimensiunii PROBE folosind obiecte ca proxy.
- PROBE înseamnă estimare bazată pe Proxy (**PROxy Based Estimation**) [Hu95].
- Principiul acestei metode este prezentat în figura 4.3.2.4.d.



**Fig.4.3.2.4.d.** Metoda PROBE de estimare a dimensiunii proiectelor

**Exercițiu #4**

1. Care sunt obiectivele fazei de definire?
2. Care sunt obiectivele analizei problemei? Care sunt recomandările pentru activitatea de analiză?
3. Descrieți conținutul documentului de specificare a problemei.
4. Care sunt principalele sarcini ale analiștilor?
5. Ce este un sistem? Care sunt principalele caracteristici ale unui sistem?
6. Descrieți cele mai cunoscute instrumente de planificare subliniind avantajele, dezavantajele și domeniul de aplicare.
7. Care este relația dintre dimensiune și resurse pentru un proiect software? Care sunt criteriile de estimare a mărimii care ar trebui îndeplinite de o metodă de estimare?
8. Ce metriki orientate pe dimensiune cunoașteți?
9. Descrieți metoda Wideband-Delphi. Evidențiați avantajele și dezavantajele.
10. Descrieți metoda Fuzzy-Logic. Evidențiați avantajele și dezavantajele.
11. Descrieți metoda Componentelor standard. Evidențiați avantajele și dezavantajele.
12. Descrieți metoda Punctelor funcționale. Evidențiați avantajele și dezavantajele. Ce știți despre conversia puncte-funcționale/sloc?
13. Descrieți metoda Punctelor caracteristice. Evidențiați avantajele și dezavantajele.
14. Ce este un proxy? Care sunt principalele sale caracteristici? Puteti descrie o metodă de estimare a dimensiunii proiectelor bazată pe proxy?

## **Chapter 4. FAZA DE DEFINIRE**

### **Partea 2**

#### **5. Estimarea costurilor proiectelor software**

##### **5.1. Obiective de estimare a costurilor**

- Estimarea costurilor pentru un Proiect Software are ca scop esențial determinarea în mod predictiv a resurselor necesare procesului de dezvoltare a produsului.
- Obiectivele principale ale estimării costurilor sunt:
  - (1) Să stabilească un buget pentru dezvoltarea proiectului.
  - (2) Să furnizeze mijloacele pentru controlul costurilor proiectului.
  - (3) Să monitorizeze dezvoltarea proiectului pe baza bugetului planificat prin compararea costurilor planificate și efective.
  - (4) Întocmirea unei baze de date referitoare la costuri, care să fie utilizată în evaluările viitoare.
  - (5) Să coreleze estimarea costurilor cu activitățile de planificare.

##### **5.2 Resursele unui proiect software**

- Pentru a dezvolta un proiect SW este necesar să se investească resurse.
- Există trei tipuri de resurse: software, hardware și umane.

###### **(1) Resursele software** constau în:

- (1.1) Programe.
- (1.2) Medii de dezvoltare.
- (1.3) Instrumente de dezvoltare împreună cu licențele de utilizare corespunzătoare.
- (1.4) Diferite categorii de resurse de suport software direct implicate în dezvoltarea unui proiect software cum ar fi:
  - (1.4.1) Sisteme de operare
  - (1.4.2) Diferite tipuri de servere

- (1.4.3) Instrumente de management de proiect
- (1.4.4) Instrumente de suport (editore de text, poștă electronică, sisteme grupware, software de rețea)
- (1.4.5) Instrumente de programare (compilatoare, SGBD)
- (1.4.6) Instrumente de dezvoltare (instrumente de caz)
- (1.4.7) Instrumente de testare
- (1.4.8) Instrumente de depanare
- (1.4.9) Instrumente de simulare
- (1.4.10) Instrumente de documentare
- (1.4.11) Instrumente de management al configurației
  - Unele dintre instrumente pot avea un grad foarte mare de specificitate și nu sunt disponibile pe piață.
  - Aceste instrumente pot să fie dezvoltate ca parte a proiectului, eventual, în funcție de situația reală, înainte de derularea proiectului sub formă de contracte separate.
  - În acest caz, ele nu sunt considerate ca resurse, ci părți auxiliare ale proiectului.

(2) **Resursele hardware** pot fi clasificate după tip și după destinație.

- După **tip** există:
  - (2.1) Sisteme informaticice cu configurațiile lor
  - (2.2) Rețele de calculatoare
  - (2.3) Linii și dispozitive de comunicație
  - (2.4) Imprimante
  - (2.5) Scanere
  - (2.6) Unități speciale de rezervă
  - (2.7) Echipamente multimedia
  - (2.8) Echipamente speciale
- După **destinație** există:
  - (2.9) Resurse de dezvoltare
  - (2.10) Testarea resurselor
  - (2.11) Resurse de prototipare
  - (2.12) Resurse de implementare pilot

(3) **Resursele umane** constau în specialiști cu anumite competențe și experiență în domeniu. Resursele umane pot fi clasificate pe profil și specialități.

- După **profil**:
  - (3.1) Manager de proiect
  - (3.2) Analist
  - (3.3) Programator
  - (3.4) Proiectant
  - (3.5) Dezvoltator
  - (3.6) Tester
  - (3.7) Responsabil cu documentația
  - (3.8) Responsabil cu asigurarea calității
- După **specialitate**:
  - (3.9) Specialist în limbi străine
  - (3.10) Specialist în sisteme
  - (3.11) Specialist în asigurarea calității
  - (3.12) Specialist în securitatea datelor
- Dintre cele trei categorii de resurse, de obicei costurile legate de resursele umane sunt cele mai substanțiale.

### **5.3 Elementele de cost ale unui proiect software**

- Elementele de cost ale unui proiect software pot fi clasificate în următoarele grupe:
  - (1) Costurile resurselor hardware și software.
  - (2) Mobilitatea personalului și costurile de formare.
  - (3) Costurile efortului (de obicei cele mai semnificative). Ele constau în:
    - (3.1) Costuri cu salariile inginerilor, dezvoltatorilor și personalului implicat în realizarea diferitelor activități.
      - Activitățile sunt măsurate ca timp de lucru înmulțit cu costul corespunzător nivelului de calificare al persoanei responsabile.
      - Câteva elemente de cost incluse în această categorie:
      - (3.1.1) Studiul problemei
      - (3.1.2) Documentare
      - (3.1.3) Antrenament personal
      - (3.1.4) Analizează
      - (3.1.5) Început
      - (3.1.6) Proiectare

- (3.1.7) Codificare
  - (3.1.8) Test
  - (3.1.9) Elaborarea documentației
  - (3.1.10) Implementare
  - (3.1.11) Formarea utilizatorilor
  - (3.1.12) Garanție
  - (3.1.13) Managementul proiectelor
  - (3.1.14) Managementul configurației
  - (3.2) Costuri cu spațiu, electricitate, căldură.
  - (3.3) Costuri pentru rețele de calculatoare și comunicații.
  - (3.4) Costuri pentru facilități comune (biblioteca, cantina).
  - (3.5) Costuri pentru pensii, asigurări sociale, impozite, asigurări de sănătate.
  - (3.6) Costuri legate de fluctuațiile valutare.
  - (3.7) Alte costuri, măsurate prin caracteristici intrinseci specifice: răscumpărări (amortizare), cheltuieli generale, consumabile.
- Costurile sunt corectate cu un **factor de corecție** care reflectă riscurile proiectului.
  - De obicei, costul estimat nu este costul efectiv al produsului.
  - Costul final depinde de mulți alți factori precum modalitatea de estimare, politica de preț a organizației în curs de dezvoltare etc.

#### **5.4 Metode de estimare a costurilor proiectelor software**

- Boehm discută mai multe tipuri diferite de metode de estimare a costurilor care sunt de obicei implementate ca tehnici. Dintre acestea se pot menționa:
  - (1) Metoda analogică
  - (2) Judecata expertilor
  - (3) Estimarea de jos în sus
  - (4) Estimarea de sus în jos
  - (5) Metoda combinată
  - (6) Metoda Parkinson
  - (7) Preț pentru câștig
  - (8) Modele parametrice

- Figura 5.4.a oferă un scurt rezumat al acestor metode împreună cu o listă a avantajelor și dezavantajelor acestora.
- Aceste metode sunt discutate în detaliu mai târziu.

Categorii Metode	Descriere	Avantaje	Limitări
Metoda Analogică	Se compară proiectul cu proiecte similare dezvoltate în trecut.	Estimarea se bazează pe experiența acuală	Trebuie să existe proiecte similare dezvoltate în timp
Judecata expertilor	Se consultă unul sau mai mulți experti	Necesită date istorice puține sau deloc; aplicabilă pentru proiecte noi sau unice	Experții pot avea uneori înclinații; nivelul de cunoștințe este uneori discutabil
Estimarea de jos în sus (Bottoms-Up)	Se estimează individual fiecare componentă după care estimările componentelor sunt însumate pentru a calcula estimarea totală	Sunt posibile estimări de mare acuratețe deoarece metoda de estimare bazată pe detalii (detailed basis of estimate (BOE) promovează asumarea răspunerii individuale	Metoda este mare consumatoare de timp; datele la nivel de detaliu ar putea lipsi, în special la începutul dezvoltării; costurile de integrare nu sunt luate de regulă în considerare
Estimarea de sus în jos (Top-Down)	Proiectul este împărțit în componente de nivel inferior și faze ale ciclului de viață începând de la cel mai înalt nivel.	Sunt luate în considerare activitățile la nivel de sistem. De obicei, este mai rapid, mai	Poate fi mai puțin precis și poate să treacă cu vederea componente de nivel inferior și posibilele probleme

	Această metodă este aplicabilă estimărilor timpurii ale costurilor atunci când sunt cunoscute numai proprietățile globale.	ușor de implementat și necesită detalii minime ale proiectului.	tehnice și oferă foarte puține detalii pentru justificarea deciziilor sau estimărilor.
Modele parametrice	Estimarea generală se realizează folosind parametrii de proiectare și algoritmi matematici	Modelele sunt de obicei rapide și ușor de utilizat; sunt utile la începutul unui program; sunt, de asemenea, obiective și repetabile	Modelele pot fi inexacte dacă nu sunt calibrate și validate corespunzător; este posibil ca datele istorice utilizate pentru calibrare să nu fie relevante pentru programele noi

**Fig. 5.4.a.** Categorii de metode de estimare a costurilor proiectelor software

#### 5.4.1 Metoda analogică

- Metoda analogică este cea mai simplă formă de estimare.
- Tehnicile analogice sunt utilizate pentru a estima costurile prin compararea programelor propuse cu programe similare finalizate anterior, pentru care sunt disponibile informații istorice.
- Datele reale din proiectele finalizate sunt extrapolate pentru a estima proiectul propus.
- Estimarea prin analogie se poate face fie la nivel de sistem, fie la nivel de componentă.
- De exemplu, dacă o agenție dorea să dezvolte un nou sistem de salarizare care să servească 5.000 de oameni și care să conțină 100.000 de linii de cod COBOL, iar o altă agenție ar fi dezvoltat un program similar de 100.000 de linii pentru 2 milioane de dolari, ar putea fi de așteptat ca programul primei agenții, ignorând inflația și alți factori similari, ar costa, de asemenea, 2 milioane de dolari.
- Un avantaj al metodei analogiei este că se bazează pe experiența reală.

- Cu toate acestea, estimarea prin analogie are o utilizare limitată deoarece, în multe cazuri, nu există programe cu adevărat similare.
- De exemplu, costul a 100.000 de linii de cod Ada pentru un program bombardier-teren nu ar fi același cu cel a 100.000 de linii de cod COBOL pentru software-ul de salarizare.
- În plus, pentru majoritatea sistemelor moderne, programele software nu au precedente istorice adevărate.
- Prin urmare, este important ca diferențele dintre proiectele finalizate și proiectele propuse să fie identificate și impactul lor estimat atunci când sunt utilizate tehnici de analogie.
- Metodele analogice nu sunt utilizate în general singure pentru a estima costurile software-ului
- Sunt cel mai adesea folosite pentru a verifica rezonabilitatea altor estimări pentru a fi rezonabile (de exemplu, verificări de corectitudine).

#### **Avantaje:**

- (1) Estimările se bazează pe datele reale ale proiectului și pe experiența anterioară.
- (2) Diferențele dintre proiectele finalizate și proiectul propus pot fi identificate și impactul estimat.

#### **Dezavantaje:**

- (1) Nu este ușor să identifici diferențele față de proiecte similare.
- (2) Această metodă este limitată, deoarece este posibil să nu existe proiecte similare sau acuratețea datelor istorice disponibile este suspectă.

#### **5.4.2 Judecata expertilor**

- Metode de estimare bazată pe judecata expertilor implică consultarea unuia sau mai multor experti pentru a-și valorifica experiența și înțelegerea unui proiect propus, pentru a oferi o estimare a costului proiectului.
- Exemple de tehnici populare bazate pe judecata expertilor sunt metodele Delphi și Wideband Delphi.
- Metodele bazate pe judecata expertilor sunt utile în (1) evaluarea diferențelor dintre programele trecute și viitoare sau (2) în estimarea programelor noi sau unice pentru care nu există precedent istoric.
- Cu toate acestea, părtinirile unui expert și posibila lipsă de cunoștințe cu privire la diferențele dintre proiectele trecute și cerințele proiectului propus creează adesea dificultăți.

- Deși tehniciile Delphi pot ajuta la atenuarea problemelor de părtinire, experții sunt supuși de obicei la presiuni mari, în activitatea lor de a estima cu exactitate costul unui program software nou.
- Prin urmare, deși modelele de expertiză sunt utile în determinarea intrărilor pentru alte modele, ele nu sunt frecvent utilizate singure ca BOE (bază de estimare), în special pentru estimările software care vor fi transmise guvernului.
- Raționamentul expertilor implică consultarea expertilor umani pentru a-și folosi experiența și înțelegerea unui proiect propus.

#### **Avantaje:**

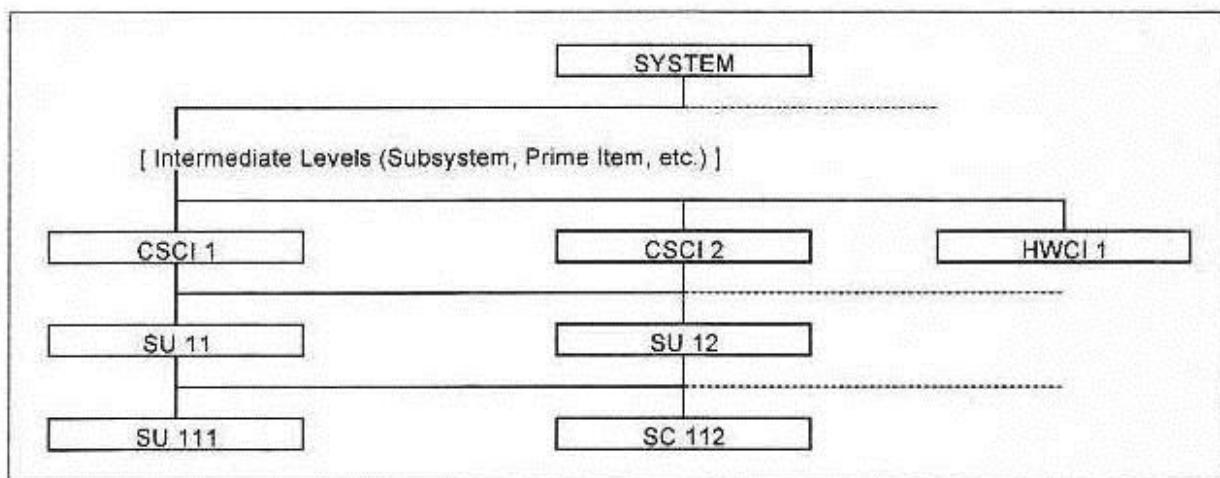
- (1) Expertul poate lua în considerare diferențele dintre experiențele anterioare ale proiectului și cerințele proiectului propus.
- (2) Expertul poate lua în considerare, de asemenea, impactul proiectului cauzat de noile tehnologii, aplicații și limbi.
- (3) Raționamentul expertului completează întotdeauna alte metodologii de estimare.

#### **Dezavantaj:**

- (1) Sunt greu de documentat factorii utilizați de expert.

#### **5.4.3 Metoda de estimare de jos în sus (Bottoms-Up Method)**

- Estimarea de jos în sus pentru estimarea costurilor proiectelor software este în general un proces foarte detaliat și mare consumator de timp.
  - (1) Poate fi utilizată pentru estimarea costurilor software prin efectuarea de analize detaliate ale costurilor specifice fiecărei unități (adică SU (Unități software)). (Fig. 5.4.3.a.)



**Fig. 5.4.3.a.** Ierarhia software ISO/EIA 12207

- (2) Costurile unitare sunt totalizate pentru a determina costul (sau efortul) pentru fiecare CSCI (Element de configurare software pentru computer);
  - (3) Apoi este posibil să se determine costul software-ului pentru întregul sistem.
- Estimarea de jos în sus este adesea folosită:
  - (1) În timpul pregătirii propunerii.
  - (2) Pentru urmărirea costurilor proiectului software.
- Avantajele acestei metode:
  - (1) Oferă o bază detaliată pentru estimarea costurilor.
  - (2) Poate fi mai precis decât alte metode.
  - (3) Întrucât personalul responsabil cu dezvoltarea unei anumite unități evaluează costurile asociate, acest tip de estimare poate promova responsabilitatea individuală pentru controlul costurilor.
  - (4) Estimarea de jos în sus poate fi utilă și pentru urmărirea costurilor, deoarece de obicei sunt stabilite estimări separate pentru activitățile care vor fi efectuate în timpul fiecărei etape de dezvoltare a software-ului.
- Estimarea de jos în sus are, de asemenea, câteva dezavantaje.
  - (1) Deoarece sunt necesare informații detaliate, aceasta tinde să consume mult timp și costuri.
  - (2) Datele istorice nu sunt întotdeauna disponibile pentru a susține aceste estimări și există tendință de a se baza în mare măsură pe raționament.
  - (3) Este posibil ca tehniciile de jos în sus să nu capteze costurile asociate cu activitățile de integrare a software-ului, care pot fi factori semnificativi ai costurilor.

**Avantaje:**

- (1) Metoda de estimare de jos în sus oferă o bază detaliată pentru estimarea costurilor.
- (2) Poate fi mai precisă decât alte metode.
- (3) Poate promova responsabilitatea individuală pentru controlul costurilor deoarece personalul responsabil cu dezvoltarea unei anumite unități evaluează costurile asociate.

**Dezavantaje:**

- (1) Sunt necesare informații detaliate, drept rezultat, tinde să consume mult timp și să aibă costuri ridicate.

- (2) Datele istorice nu sunt întotdeauna disponibile pentru a susține aceste estimări.
- (3) Nu evidențiază costurile asociate activităților de integrare software.

#### **5.4.4 Metoda de estimare de sus în jos (Top-Down Method)**

- Metoda de estimare de sus în jos se bazează pe caracteristicile generale ale proiectului software.
- Proiectul este împărțit în componente de nivel inferior și faze ale ciclului de viață începând de la cel mai înalt nivel.
- Această metodă este mai aplicabilă estimărilor timpurii ale costurilor atunci când sunt cunoscute numai proprietățile globale.

##### **Avantaje:**

- (1) Sunt luate în considerare activitățile la nivel de sistem (integrare, documentare, control de proiect, management de configurare etc.).
- (2) Este de obicei mai rapid și mai ușor de implementat.
- (3) Necesită detalii minime ale proiectului.

##### **Dezavantaje:**

- (1) Poate fi mai puțin precisă.
- (2) Are tendința de a trece cu vederea componente de nivel inferior și posibilele probleme tehnice.
- (3) Oferă foarte puține detalii pentru justificarea deciziilor sau estimărilor.

#### **5.4.5 Metoda combinată**

- Metoda de sus în jos începe cu o estimare globală care este detaliată printr-un proces de rafinare treptat
- Metoda de jos în sus estimează costurile software prin efectuarea de analize detaliate ale costurilor specifice fiecărei unități (adică SU (Software Units)).
- Costurile unitare sunt totalizate pentru a determina costul (sau efortul) pentru fiecare CSCI (Articole de configurare software de calculator).
- Costurile CSCI sunt totalizate pentru a determina costul software pentru subsisteme.
- Costurile subsistemelor sunt totalizate pentru a determina eventual costul întregului sistem.

- Cele două metode pot fi combinate în felul următor, process care beneficiază de avantajele ambelor metode:
  - (1) Proiectul este împărțit în componente de nivel inferior într-o manieră de sus în jos, în măsura în care este rezonabil.
  - (2) Se estimează cea mai mică unitate curentă.
  - (3) Costurile se determină de jos sus pornind de la cele mai mici unități actuale.
  - (4) Sumele totale sunt ajustate prin redistribuirea de sus în jos a costurilor elementelor.
  - (5) Pașii (2), (3) și (4) se repetă până când se atinge nivelul dorit de precizie.

#### **5.4.6 Metoda Parkinson**

- Legea lui Parkinson spune: „Proiectul costă oricît atât vreme cât sunt resurse disponibile”.
  - Aceasta înseamnă că dezvoltarea unui proiect poate fi prelungită în timp, până când există resurse care să-l alimenteze.
- Cu alte cuvinte, același proiect poate fi finalizat în 3 om\*luni sau în 6 om\*luni.
- Explicația este că un proiect care se finalizează într-un timp scurt:
  - (1) Va fi mai puțin elaborat.
  - (2) Va include mai puține funcționalități.
  - (3) Va fi mai puțin testat.
- În timp ce un proiect finalizat într-un timp mai lung nu va avea aceste probleme.
- De fapt, un proiect software complex nu este niciodată terminat.
  - Poate fi oricând îmbunătățit și optimizat, performanțe crescute și noi funcționalități adăugate.
  - Aceasta este motivul pentru care produsele software au de obicei mai multe versiuni și sunt într-o continuă evoluție.

#### **Avantaj:**

- (1) Metoda nu produce depășiri de buget.

#### **Dezavantaj:**

- (1) Produsele obținute sunt de obicei nefinalizate.

#### **5.4.7 Metoda prețului câștigător**

- Metoda prețului castigator stipulează că proiectul costă atât cât își permite clientul să-l cheltuiască.
  - În acest caz, estimarea proiectului trebuie să țină cont de opțiunile de plată ale clientului.
- Începând cu un preț impus, estimarea se face în astă fel încât suma disponibilă să nu fie depășită, bineînteles dacă acest lucru este posibil.
  - De fapt, clientul primește atât cât își poate permite să plătească.

#### **Avantaj:**

- (1) După evaluare, contractul este ușor de semnat.

#### **Dezavantaje:**

- (1) Probabilitatea ca un client să obțină ceea ce își dorește este foarte mică.
- (2) Pe de altă parte, costul nu reflectă cu acuratețe efortul necesar.

### **5.5 Modele de evaluare a costurilor software**

- Boehm discută, de asemenea, diferite modele de evaluare a costurilor.
- Modelele de evaluare a costurilor software sunt împărțite în două mari categorii:
  - (1) Modele de descompunere.
  - (2) Modele parametrice.

#### **5.5.1 Modele de descompunere**

- **Modelele de descompunere** împart proiectul în sarcini mici (elemente) care sunt evaluate separat.
- În această categorie sunt incluse modele de evaluare a costurilor ca:
  - (1) **Modelul estimare efort** (Modelul EE).
  - (2) **Modelul liniilor de cod** (Modelul LOC).
  - (3) **Modelul punctelor funcționale** (Modelul FP).

##### **5.5.1.1 Modelul estimare efort (Model EE)**

- **Modelul estimare efort** (EE) (Effort Estimation Model) este un model de descompunere.
- Construcția modelului EE constă în următorii pași:
  - (1) Proiectul este descompus în sarcini funcționale (funcții).

- Sarcinile funcționale sunt cele corespunzătoare caracteristicilor (trăsăturilor) proiectului.

(2) Fiecare sarcină funcțională este apoi descompusă în subsarcini corespunzătoare fazelor ciclului de viață al dezvoltării proiectului.

- Subsarcinile corespunzătoare fazelor ciclului de viață sunt: analiza, proiectarea preliminară, proiectarea detaliată, codificarea, testarea. (Fig. 5.5.1.1.a)

(3) Se estimează costurile pentru fiecare subsarcină a fiecărei caracteristici (de obicei în ore).

(4) Apoi sunt determinate costurile caracteristicilor, costurile fazelor și costul total.

Nr. Crt	Task Caracteristică	Analiză	Design preliminar	Design detaliat	Codificare	Testare	Total
1.	Caracteristică 1						
2.	Caracteristică 2						
3	Caracteristică 3						
4.	Caracteristică 4						
	...						
	...						
	<b>Total</b>						
	Cost unitar						
	<b>Cost Total</b>						

**Fig. 5.5.1.1.a.** Modelul de evaluare al costurilor EE

### **Avantaje:**

- (1) Costurile de fază ale fiecărei caracteristici pot fi evidențiate clar.
  - Acest lucru este foarte important pentru că în multe situații costurile diferă de la o fază la alta.
  - De exemplu, de obicei,  
cost\_analiză>cost\_desig>cost\_codare>cost\_test>cost\_furnizare (în bani om\*zi)
- (2) Metoda de evaluare este simplă, foarte eficientă și obține rezultate care sunt afectate de un coeficient de eroare redus.

### **Dezavantaje:**

- (1) Metoda necesită evaluatori cu experiență semnificativă și un grad ridicat de profesionalism.

#### **5.5.1.2 Modele LOC (linii de cod) și FP (puncte funcționale)**

- **Modelele LOC** și **FP** sunt, de asemenea, **modele de descompunere**.
- La nivel de structură generală sunt similare.
- Diferența apare la maniera de estimare:
  - **LOC** (Linii de cod) pentru modelul LOC
  - **FP** (Puncte Funcționale) pentru Modelul FP
- De exemplu, estimarea bazată pe **modelul LOC** necesită următorii pași:
  - (1) Proiectul este descompus în **sarcini funcționale** (funcții).
  - (2) Sarcinile funcționale sunt cele corespunzătoare **caracteristicilor proiectului**.
  - (3) Fiecare **caracteristică** este estimată în LOC.
  - (4) Cunoscând **costurile unitare** (Cost/LOC), se determină **costul pentru fiecare caracteristică**.
  - (5) Cunoscând **productivitatea** (LOC/zi) se determină **durata fiecărei caracteristici**.
  - (6) Prin însumare se pot determina **costul total** și **durata totală**.
  - (7) Durata totală poate fi redusă dacă mai mulți oameni vor lucra în paralel.(Fig.5.5.1.2.a.)

Nr.crt	Caracteristică	LOC	Cost/LOC	LOC/zi	Cost (col 1)* (col 2)	Durată (col 1)/ (col 3)
0	1	2	3	4	5	6
1.	Caracteristică 1					
2	Caracteristică 2					
3.	Caracteristică 3					
...						
...						
	<b>Total</b>					

**Fig.5.5.1.2.a.** Model de evaluare LOC

- Utilizarea modelului LOC trebuie să ia în considerare faptul că productivitatea software-ului (Cost/LOC) depinde de **limbajul de programare** utilizat.
  - Productivitatea software-ului este mai mare pentru limbajele de nivel înalt și mai mică pentru limbajele de asamblare.
- Modelul LOC este dificil de aplicat dacă sunt utilizate instrumente de dezvoltare vizuală.
  - În aceste cazuri, modelul FP este mai potrivit.
- În cazul estimării bazate pe **modelul punctelor funcționale**, estimarea este similară cu unica diferență că în loc de LOC-uri se operează cu **puncte funcționale** (vezi 4.3.2.1.)

## 5.5.2 Modele parametrice

- Modelele parametrice generează estimări bazate pe relații statistice.

- Modelele parametrice relaționează variabilele dependente (adică, costul și/sau planificarea) cu una sau mai multe variabile independente (adică, parametri).
- Modelele de estimare parametrică pentru proiectele software estimează, în general, costurile globale ale sistemului sau costurile CSCI (Articole de configurare a software-ului de calculator) pe baza caracteristicilor de proiectare ale unui program software.
- Aceste costuri totale pot fi partajate între:
  - SU-urile de nivel inferior (Unitățile software – la nivelul cel mai scăzut conțin între 100 și 200 de LOC).
  - Fazele ciclului de viață.
- **Avantajele modelelor parametrice** sunt:
  - (1) Sunt rapide și ușor de utilizat.
  - (2) Necesită puține informații detaliate (după ce sunt calibrate).
  - (3) Capturează costurile totale ale sistemului sau la nivel de CSCI (inclusiv costurile pentru activitățile de integrare).
  - (4) Pot fi la fel de precise (dacă nu mai precise) ca și alte tehnici de estimare, dacă sunt calibrate și validate corespunzător.
- Datorită acestor avantaje, modelele parametrice sunt, în general, tehnica de estimare software preferată de DOD.
- Există mai multe **modele parametrice comerciale** care sunt utilizate pe scară largă atât de industrie, cât și de guvern.
- Există, de asemenea, multe **modele de estimare software parametrice sofisticate** care utilizează parametri mulți pentru a calcula costurile și efortul software.
- În această secțiune vor fi discutate **trei modele parametrice software** comune și va fi oferită o înțelegere de bază a unora dintre atributele lor comune.
- Cele trei modele sunt:
  - **COnstructive COst MOdel** (COCOMO).
  - **PRICE Software Model** (PRICE S®).
  - **Galarath Software Evaluation and Estimation of Resources -** (SEER-SEM®).
- Pentru fiecare dintre aceste modele sunt discutate următoarele informații:
  - (1) Fundamente (Background).
  - (2) Intrări principale (adică, parametri).
  - (3) Prelucrare.

(4) leșiri principale.

(5) Capacități de estimare a costurilor pentru suport software. Acestea sunt abordate din cauza importanței tot mai mari a acestei zone.

#### **5.5.2.1 Modelul COCOMO 81**

- Dr. Barry Boehm, fost collaborator al TRW Corporation, a dezvoltat modelul software parametric COCOMO (COnstructive COst MOdel) pentru care a publicat prima ediție în 1981.
- COCOMO nu este un model proprietar.
  - Este descris integral în cartea Dr. Boehm din 1981, „Software Engineering Economics”.
  - În realitate, pentru a utiliza modelul COCOMO este necesar doar un calculator de buzunar cu o funcție exponentială.
  - Pe piață sunt disponibile o multitudine de versiuni computerizate ale lui COCOMO.
    - Spre exemplu, versiunea îmbunătățită revizuită a COCOMO intermediar (REVIC), dezvoltată de Ray Kile, a fost „ediția standard COCOMO a Air Force la sfârșitul anilor 1980 și începutul anilor 1990.
  - Pe parcursul timpului, Dr. Boehm a făcut revizuiri substanțiale la modelul COCOMO 81.
  - COCOMO81 a fost actualizat la COCOMO II.

#### **Prezentare generală COCOMO 81**

- COCOMO 81 (adică prima versiune a COCOMO) este un model bazat pe regresie care ia în considerare programele în trei moduri (tipuri) (Fig. 5.5.2.1.a.)
  - (1) **Organic**
  - (2) **Semi-decomandat** (Semi-detached)
  - (3) **Încorporat** (Embedded)

## Modelul COCOMO - Tipuri de proiecte

Tip de proiect	Caracteristici			Constrângeri de timp	Mediu de lucru
	Dimensiune	Nouitate			
Organic	Foarte mic	Mic		Nu foarte strânse	Stabil
Embedded	Mare	Mare		Foarte strânse	Interfață utilizator sau hardware complexe
Semidetached	Mediu	Mediu		Mediu	Mediu

## Modelul COCOMO de bază - Constante de calcul

Tip de proiect	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Embedded	3.0	1.12	2.5	0.35
Semidetached	3.6	1.20	2.5	0.32

### COCOMO de bază: Ecuatii fundamentale

$$\text{Effort} = a * \text{dimensiune}^b$$

$$\text{Durata} = c * \text{Effort}^d$$

$$\text{Numar persoane} = \text{effort}/\text{Durata}$$

Fig. 5.5.2.1.a. Tipuri (moduri) de proiecte COCOMO

- Pentru fiecare mod sunt stabilite ecuații separate care relaționează **nivelul de efort (LOE)** exprimat în **om-luni (MM)** cu dimensiunea programului exprimată în **mii de instrucțiuni sursă livrate (KDSI)**.
- Există trei niveluri în COCOMO 81:
  - De bază**
  - Intermediar nominal**
  - Detaliat**
- (1) **Nivelul de bază** constă din trei modele simple (câte unul pentru fiecare mod). Fiecare model constă în două ecuații cu un singur parametru: una pentru efort și una pentru planificare.
  - Ecuatiile de efort** relaționează MM (Om\*Lună) cu KDSI

- **Ecuățiile de planificare** raportează lunile de timp necesar (M) la MM, calculate din ecuațiile efortului.
- Nivelul de bază al COCOMO este adesea folosit pentru a dezvolta estimări grosiere ca de ordin de mărime (ROM - rough order of magnitude) pentru costurile software.
- Figurile 5.5.2.1.a și 5.5.2.1.b. (primele două coloane) listează ecuațiile de efort și de planificare pentru cele trei moduri ale Basic COCOMO 81.

Nivel Mode	<b>Basic</b> <i>Efort</i>	<b>Basic</b> și <b>Intermediar nominal</b> <i>Planificare</i>	<b>Intermediar nominal</b> <i>Efort</i>
Organic	$MM = 2.4 \text{ (KDSI)}^{1.05}$	$M = 2.5 \text{ (MM)}^{0.38}$	$MM = 3.2 \text{ (KDSI)}^{1.05}$
Semi-Detached	$MM = 3.0 \text{ (KDSI)}^{1.12}$	$M = 2.5 \text{ (MM)}^{0.35}$	$MM = 3.0 \text{ (KDSI)}^{1.12}$
Embedded	$MM = 3.6 \text{ (KDSI)}^{1.20}$	$M = 2.5 \text{ (MM)}^{0.32}$	$MM = 2.8 \text{ (KDSI)}^{1.20}$

**Fig. 5.5.2.1.b.** Ecuățiile elementare COCOMO 81

- (2) Nivelul **Intermediar nominal** COCOMO 81
  - Conține și ecuații nominale prezentate în figura 5.5.2.1.b. (coloanele 2-3)
    - Ecuățiile de planificare sunt similare cu ecuațiile pentru nivelul de bază pentru fiecare mod.
    - Coeficienții de efort sunt specifici.
  - În plus, **modelul intermediar** introduce **15 multiplicatori**.
    - Acești **multiplicatori** **ajustează rezultatul ecuației efortului intermediar nominal** pentru a reflecta atributele unice ale unui program (fig. 5.5.2.1.c.).
    - Fiecare multiplicator are valori predeterminate raportate de la foarte mici, la extrem de ridicate prezentate în tabelul 5.5.2.1.b.
    - Valoarea nominală pentru fiecare atribut este 1,00 (fără influență).

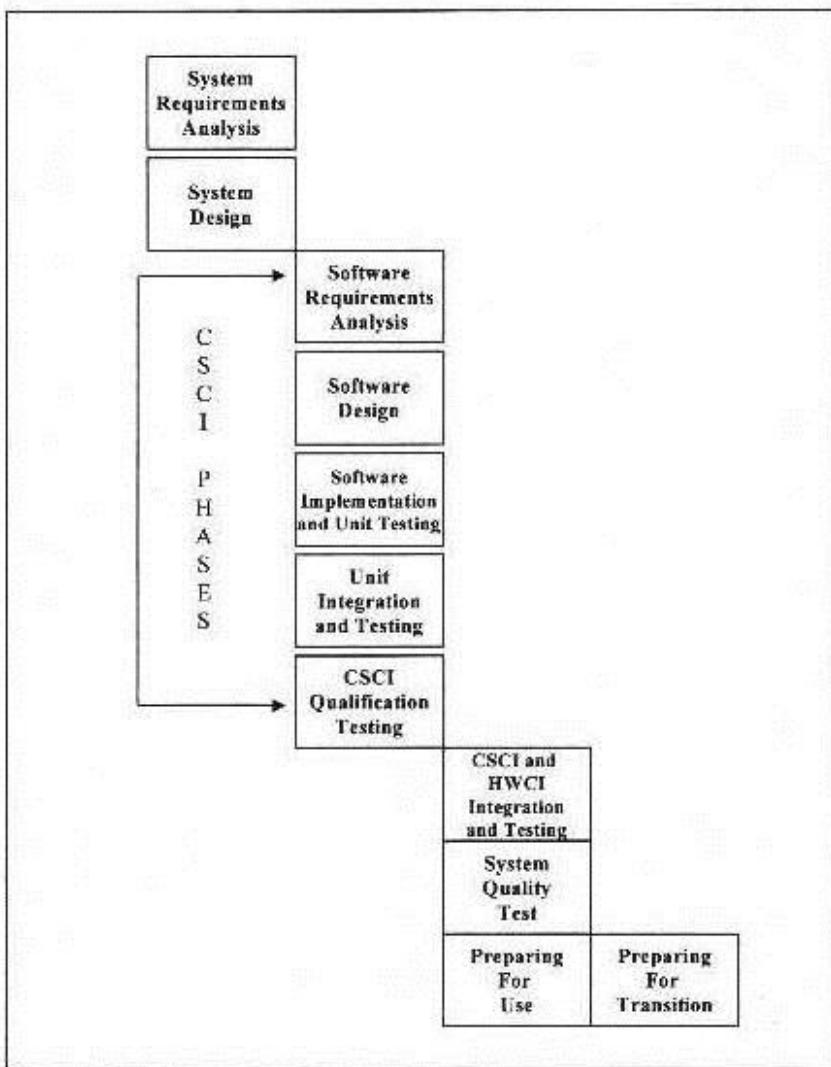
		Valori (Rating)					
Nr.	Atribute	foarte mic (VL)	mic (LO)	nominal (NM)	mare (HI)	foarte mare (VH)	extrem de mare (XH)
1	Cerințe de fiabilitate Required Reliability (RELY)	0.75	0.88	1.00	1.15	1.40	
2	Dimensiunea bazei de date Database Size (DATA)		0.94	1.00	1.08	1.16	
3	Complexitatea produsului Product Complexity (CPLX)	0.70	0.85	1.00	1.15	1.30	1.65
4	Constrângeri de timp de execuție Execution Time Constraints (TIME)			1.00	1.11	1.30	1.66
5	Constrângeri de memorie			1.00	1.06	1.21	1.56

	Main Storage Constraint (STOR)					
6	Volatilitatea mașinii virtuale Virtual Machine Volatility (VIRT)		0.87	1.00	1.15	1.30
7	Timp de răspuns Computer Turnaround Time (TURN)		0.87	1.00	1.07	1.15
8	Capabilitatea analiștilor Analyst Capability (ACAP)	1.46	1.19	1.00	0.86	0.71
9	Experiența în aplicații Applications Experience (AEXP)	1.29	1.13	1.00	0.91	0.82
10	Capabilitatea programatorilor Programmer Capability (PCAP)	1.42	1.17	1.00	0.86	0.70
11	Experiența cu mașini virtuale Virtual Machine Experience (VEXP)	1.21	1.10	1.00	0.90	

12	Experiența în limbajul de programare Programming Language Experience (LEXP)	1.14	1.07	1.00	0.95		
13	Utilizarea practicilor moderne de programare Use of Modern Programming Practices (MODP)	1.24	1.10	1.00	0.91	0.82	
14	Utilizarea uneltelelor software Use of Software Tools (TOOL)	1.24	1.10	1.00	0.91	0.83	
15	Cerințe de planificare a dezvoltării Required Development Schedule (SCED)	1.23	1.08	1.00	1.04	1.10	

**Fig. 5.5.2.1.c.** Multiplicatorii de efort pentru nivelul Intermediate COCOMO 81

- (3) Nivelul **detaliat** COCOMO 81
  - Ajustează multiplicatorii pentru fiecare fază CSCI a ciclului de viață al software-ului.
  - De exemplu în fazele CSCI în Software Waterfall Model (Fig.5.5.2.1.d.).



**Fig. 5.5.2.1.d.** Standardul IEEE/EIA 12207 - Fazele modelului cascadă pentru dezvoltarea de software

#### Observații:

- Toate nivelurile COCOMO sunt concepute pentru a fi utilizate pentru orice program, deși modelul este probabil cel mai bine utilizat pentru estimările la nivel CSCl.
- Toate nivelurile iau în considerare fazele CSCl.
  - De exemplu, ciclul de dezvoltare software de la proiectare până la testarea CSCl, deși faza de proiectare este împărțită în fazele de „proiectarea produsului” și „proiectare detaliată”.

- Toate cele trei niveluri COCOMO 81 permit o ajustare a KDSI pentru software reutilizat sau modificat.
- COCOMO 81 calculează „KDSI eficient” pe baza procentelor de reproiectare, recodificare și retestare necesare.

## COCOMO 81 Intrări

- Elementele prezentate în continuare se referă la **nivelul intermedian** al COCOMO 81.
- **Intrarea primară** pentru modelul intermedian nominal COCOMO este **dimensiunea programului, în KDSI**.
- Cu toate acestea, există cele **15 atribute suplimentare** care trebuie evaluate.
- Aceste atribute, care au fost prezentate în figura 5.5.2.1.c, sunt clasificate în următoarele patru categorii:
  - (1) **Atributele produsului**: descriu mediul în care programul operează. Cele trei atribute din această categorie sunt:
    - (1) Cerințe de fiabilitate (RELY),
    - (2) Dimensiunea bazei de date (DATE),
    - (3) Complexitatea produsului (CPLX).
  - (2) **Atributele sistemului de calcul**: descriu relația dintre program și sistemul său gazdă sau sistemul de dezvoltare. Cele patru atribute din această categorie sunt:
    - (4) Constrângeri de timp de execuție (TIME),
    - (5) Constrângeri de memorie (STOR),
    - (6) Volatilitatea mașinii virtuale (VIRT),
    - (7) Timp de răspuns al sistemului (TURN).
  - (3) **Atributele personalului**: descrieți capacitatea și experiența personalului desemnat programului. Cele cinci atribute din această categorie includ:
    - (8) Capabilitățile analiștilor (ACAP),
    - (9) Experiența în aplicații (AEXP),
    - (10) Capabilitățile programatorilor (PCAP),
    - (11) Experiența cu mașini virtuale (VEXP).
    - (12) Experiența în limbajul de programare (LEXP),

(4) **Atributele proiectului:** descriu aspectele selectate de management de proiect ale unui program. Cele trei atribute din această categorie includ:

- (13) Utilizarea practicilor moderne de programare (MODP),
- (14) Utilizarea instrumentelor software (TOOL),
- (15) Cerințe de planificare a dezvoltării (required development schedule) (SCED).

- Tabelele elaborate de Dr. Boehm descriu evaluări de la „foarte scăzut” la „extrem de ridicat”, care trebuie evaluate pentru fiecare dintre cele 15 atribute identificate mai sus.
- De exemplu, factorul de fiabilitate (adică RELY) ar fi evaluat:
  - „Foarte scăzut” dacă o eroare într-un anumit sistem software cauzează doar mici neplăceri;
  - „Nominal” dacă o eroare ar putea avea ca rezultat pierderi moderate, recuperabile;
  - „Foarte mare” dacă este în joc pierderea potențială de vieți umane.
- Figura 5.5.2.1.c, extrasă din cartea Dr. Boehm, arată valorile numerice atribuite unor evaluări specifice de Foarte Scăzut (VL), Scăzut (LO), Nominal (NM), Mare (HI), Foarte Mare (VH) și Extrem de mare (XH) pentru fiecare dintre cele 15 atribute.
  - De reținut că toate atributele „nominal” nu au niciun efect asupra efortului necesar.
- Cartea menționată a Dr. Boehm oferă îndrumări detaliate cu privire la aceste atribute.

### COCOMO 81 Procesare

- Pentru a utiliza modelul COCOMO 81 intermedier, se parcurg următorii pași:
  - (1) Pentru programul de estimat se realizează o evaluare a efortului în MM pornind de la dimensiune.
    - Sunt folosite în acest scop ecuațiile nominale prezentate în fig. 5.5.2.1.b. (coloanele 2-3).
  - (2) Se evaluatează valori pentru toți cei 15 multiplicatori ai efortului (fig. 5.5.2.1.c.).
  - (3) Evaluările realizate pentru toate cele 15 atribute sunt înmulțite pentru a obține un produs global pentru atribute (denumit în continuare „P”).
  - (4) Valoarea MM rezultată din ecuația nominală este înmulțită cu P pentru a calcula efortul MM necesar pentru programul în cauză.

- Un exemplu:
    - Să presupunem că un **program încorporat** are o dimensiune de 20 KDSI.
    - Ecuația nominală încorporată este:  $MM=2,8(20)1,20$ , ceea ce dă 102 MM.
    - În continuare, se presupune că toate atributele sunt evaluate la nivel „nominal”, cu excepția fiabilității (adică, RELY), care este evaluată „foarte ridicat” și i se atribuie o valoare de 1,40; P este acum 1,40.
    - **Nivelul de efort** calculat de COCOMO Intermediar ar fi  $(102 \times 1,4) = 143$  om-luni.
    - Planificarea calculate tot pe baza ecuațiilor prezentate în fig.2.6-5, este  $M=2,5(143)0,32 = 12,2$  luni.
  - Din revizuirea modelului, se constată că conceptul de bază din spatele Intermediate COCOMO 81 este destul de simplu.
  - Principalele provocări în utilizarea intermediarului COCOMO 81 se referă la:
    - (1) Estimarea corectă a dimensiunea software-ului.
    - (2) Atribuirea corespunzătoare a valorilor celor 15 atributelor.
  - COCOMO 81 oferă, de asemenea, facilități pentru reutilizarea software-ului existent prin adaptarea dimensiunii (KDSI) a software-ului care urmează să fie estimat.
  - Ecuația pentru această revizuire a mărimei este prezentată în fig. 5.5.2.1.e.
- 

**COCOMO 81:** Ecuația pentru revizuirea dimensiunii

**EDSI = ADSI (AAF / 100),**

unde

**EDSI** = număr echivalent de instrucțiuni sursă.

**ADSI** = numărul de instrucțiuni care urmează să fie adaptate (modificate sau reutilizate) din software-ul existent pentru utilizare în noul software.

**AAF** = Factor de ajustare a adaptării.

- Se calculează ca **AAF** = 0,40 (**DM**) + 0,30 (**CM**) + 0,30 (**IM**)
- Este posibil ca **DM**, **CM** sau **IM** să depășească 100%, ceea ce indică faptul că efortul de reutilizare a software-ului ar putea fi mai mare decât cel necesar pentru dezvoltarea unui nou program.
  - **DM** = procentul din designul software adaptat care va fi modificat.
  - **CM** = procentul de cod software adaptat modificat.

- **IM** = procentul de efort necesar pentru a integra software-ul adaptat în produsul general și a testa produsul rezultat.
- 

**Fig. 5.5.2.1.e.** Intermediate COCOMO 81. Revizuirea dimensiunii codului

### COCOMO 81 Ieșiri

- Ieșirile modelului intermediar COCOMO sunt următoarele:
  - (1) Valoare efort LOE în MM pentru proiectul care se estimează.
  - (2) Planificarea în luni.
- Rezultatul efortului poate fi ușor convertit într-o **valoare monetară** dacă se cunoaște costul pe MM.
- De asemenea, este posibil să se determine alocarea efortului general către diferite faze ale ciclului de viață CSCl, sau perioade de timp, folosind informațiile prezentate în cartea Dr. Boehm.

### Considerații privind costurile de asistență (suport) COCOMO 81

- Modelul de menenanță COCOMO 81 (COCOMO-M) poate fi utilizat pentru a estima MM-urile anuale necesare pentru a susține un program software.
  - Se calculează un **produs al multiplicatorilor de întreținere** (PM), similar cu P pentru ecuațiile efortului de dezvoltare.
    - PM este calculată în același mod ca și pentru nivelul intermediar al COCOMO-M. Toate intrările COCOMO sunt utilizate, cu excepția faptului că sunt atribuite valori numerice diferite la două atrbute:
      - Fiabilitate (RELY).
      - Utilizarea practicilor moderne de proiectare (MODP).
  - Este necesară, de asemenea, o evaluare a traficului anual de modificări (ACT), care să reflecte fracțiunea de cod care trebuie schimbată pe an.
  - **Lunile anuale om** (MMA) sunt calculate din ecuația prezentată mai jos.
  - **Costurile anuale de suport** pot fi calculate prin înmulțirea numărului de MMA-uri cu costul pe MM (Fig. 5.5.2.1.f.)
- 

**COCOMO 81:** Costurile suport anuale

$$\text{MMA} = (\text{MMNOM}) (\text{ACT}) (\text{PM}),$$

unde

**MMA** = număr de MM-uri pe an.

**MMNOM** = MM-urile nominal intermediate.

**ACT** = Traficul modificărilor anuale (Annual Change Traffic).

**PM** = Produsul multiplicatorilor de menenanță (Product of Maintenance multipliers).

---

**Fig. 5.5.2.1.f.** Intermediate COCOMO 81. Costuri de asistență anuale

### 5.5.2.2 Modelul COCOMO II

- COCOMO II a fost dezvoltat la mijlocul anilor 1990 de un consorțiu de organizații.
  - Autorul a fost dr. Barry Boehm și mai mulți studenți absolvenți de la Universitatea din California de Sud (USC).
  - Prima versiune a modelului COCOMO II a fost lansată la sfârșitul anului 1996.
- **Scopul** COCOMO II a fost „de a dezvolta un model de estimare a costurilor software și a programului adaptat la practicile ciclului de viață din anii 1990 și 2000”.
  - În timp ce COCOMO 81 sa bazat în mare parte pe proiecte dezvoltate folosind metodologia cascadă, COCOMO II este mai compatibil cu noile metodologii de proiectare (de exemplu, tehnici orientate pe obiecte).
  - Modelul actualizat include și o nouă bază de date constând din date transmise de membrii consorțiuului.
- Similar cu COCOMO 81, modelul este nonproprietary și sunt disponibile ediții computerizate.
- **COCOMO II** are o nouă filozofie.
- **COCOMO II** conține **trei stagi** de estimare.
  - (1) **Stagiul 1** numită **Componerea aplicațiilor** (Applications Composition)
    - Sprijină eforturile de creare a prototipurilor sau de compunere a aplicațiilor.
    - Folosește punctele obiect ca măsură de dimensiune.
  - (2) **Stagiul 2** numită **Design timpuriu** (Early Design)

- Suportă estimarea în fazele timpurii de proiectare ale programului.
- Folosește punctele funcționale sau mii de linii sursă de cod (KSLOC), ca măsură de dimensiune.
- În plus, Etapa 2 utilizează 7 atribute ca multiplicatori de efort, care se bazează pe combinații ale celor 17 atribute utilizate în Etapa 3.

(3) **Stagiul 3** numită **Post arhitectură** (Post Arhitectură)

- Suportă estimarea după proiectarea timpurie.
- Etapa 3 folosește 17 atribute ca multiplicatori de efort.
- Etapa 3 este de fapt o modificare a COCOMO 81, așa că descrierile discutate în paragrafele următoare reprezintă diferențele dintre COCOMO 81 și COCOMO II.
- Cele trei etape sunt descrise mai detaliat, în ordine inversă.

### **COCOMO II Prezentare generală Stagiul 3 - Post Arhitectură**

- Ecuațiile COCOMO II sunt revizuiri ale ecuațiilor COCOMO 81.
  - Aceste revizuiri conțin exponenți variabili și tratează diferit reutilizarea software-ului.
  - **Ecuația Efortului** pentru **Stagiul 3** este prezentată în fig. 5.5.2.2.a.
- 

#### **COCOMO II: Ecuația efortului** pentru **Stagiul 3**

$$\mathbf{PM} = [\mathbf{A} \ (\mathbf{Dimensiune'})\mathbf{B} \ (\mathbf{EM})] + \mathbf{PMAT}, \text{ unde}$$

**PM** reprezintă o persoană-luni de efort estimat.

**A** este un coeficient care este setat provizoriu la 2,5 (adică, valoarea implicită), dar ar trebui calibrat pentru a reflecta costurile și cultura organizației specifice.

**Dimensiune'** = **Dimensiune (1 + BRAK/100)** în care **BRAK** este o pierdere sau procentul de cod aruncat din cauza volatilității cerințelor. Mărimea în sine este suma **KSLOC-ului nou și adaptat**.

**B** este un exponent variabil cu ecuația: **B = 0,91 + 0,01 (SF)**,

- SF este suma a cinci factori de scară (fig. 5.5.2.2.b).
- Cei cinci factori de scară, enumerați în figura 5.5.2.2.b sunt:
  - Precedența (**PREC**);
  - Flexibilitatea dezvoltării (**FLEX**);
  - Gradul de rezoluție a riscului (**RESL**);

- Coeziunea echipei (**TEAM**);
  - Maturitatea procesului (**PMAT**).
- La fel ca multiplicatorii de efort din COCOMO 81 și COCOMO II, aceștia sunt evaluati de la VL la XH (foarte scăzut la foarte mare).
- Aceste valori sunt prezentate în figura 5.5.2.2.b. pentru evaluările mai favorabile rezultând valori mai mici și, prin urmare, un exponent de efort mai mic.
  - Pentru PREC, de exemplu, VL corespunde „cu totul fără precedent”, în timp ce XH corespunde cu „foarte familiar”. Figura 5.5.2.2.b arată că valoarea numerică pentru VL (4,05) este mai mare decât valoarea pentru XH (0,00).

**EM** este produsul a 17 multiplicatori de efort prezentat mai târziu în fig. 5.5.2.2.e. Acestea sunt similare cu cele utilizate în COCOMO 81 intermedier.

**PMAT** este efortul pentru componente traduse automat.

---

**Fig. 5.5.2.2.a.** COCOMO II. Stagiul 3. Ecuația de efort

Nr	Factorii de scală (Scale Factors)	Valori (Rating)					
		VL	LO	NM	HI	VH	XH
1	Precedență (Precedendedness) (PREC)	4.05	3.24	2.43	1.62	0.81	0.00
2	Flexibilitatea dezvoltării (Development Flexibility) (FLEX)	6.07	4.86	3.64	2.43	1.21	0.00
3	Arhitectura/Gradul de rezoluție al riscului (Architecture / Risk Resolution) (RESL)	4.22	3.38	2.53	1.69	0.84	0.00

4	Coeziunea echipei (Team Cohesion) (TEAM)	4.94	3.95	2.97	1.98	0.99	0.00
5	Maturitatea procesului (Process Maturity) (PMAT)	4.54	3.64	2.73	1.82	0.91	0.00

**Fig. 5.5.2.2.b.** COCOMO II. Etapa 3. Factori de scală

- Ecuăția de planificare pentru COCOMO II are un exponent variabil și este calculată folosind următoarea ecuație (Fig. 5.5.2.2.c).

#### **COCOMO II: Ecuăția de planificare pentru Stagiul 3**

$$M = [3.0 \times PM^{(0.33 + 0.2(B-0.91))}] (SCED\%/100), \text{ unde}$$

**PM** = *persoane-lună* indicând efortul estimat.

**SCED%** = *procentul de compresie sau de extindere a planificării*. Este similar cu parametrul utilizat în COCOMO 81 dar cu valoare numerică diferită.

**Fig. 5.5.2.2.c.** COCOMO II. Stagiul 3. Ecuăția de planificare

- După cum este menționat în documentație, COCOMO II tratează diferit codul reutilizat.
- Noua **ecuație de reutilizare** ia în considerare faptul că este nevoie de un anumit efort pentru reutilizare, indiferent cât de mult este de fapt modificat codul.
- În acest scop este utilizat un factor **multiplicator de ajustare a adaptării**.
- **Multiplicatorul de ajustare a adaptării** (AAM) este neliniar și este calculat în baza ecuației prezentată în fig. 5.5.2.2.d.

#### **COCOMO II: Multiplicatorul de ajustare a adaptării (The Adaptation Adjustment Multiplier) (AAM) Stagiul 3**

**AAM = (AAF + AA + SU + UNFM)/100**, unde

**AAF** = Factor de ajustare a adaptării (Adaptation Adjustment Factor). Se calculează ca **AAF = 0.40 (DM) + 0.30 (CM) + 0.30 (IM)**. Este posibil ca **DM**, **CM** sau **IM** să depășească 100%, ceea ce indică faptul că efortul de reutilizare a software-ului ar putea fi mai mare decât cel necesar pentru dezvoltarea unui nou program.

**DM** = procentul din designul software adaptat care va fi modificat.

**CM** = procentul de cod software adaptat modificat.

**IM** = procentul de efort necesar pentru a integra software-ul adaptat în produsul general și a testa produsul rezultat.

**AA** = gradul de evaluare și asimilare pentru a determina adevararea reutilizării și pentru a integra descrierea software-ului reutilizat în descrierea produsului.

**SU** = cantitatea de înțelegere necesară a software-ului.

**UNFM** = gradul de nefamiliarizare a programatorului cu software.

---

**Fig. 5.5.2.2.d. COCOMO II. Stagiul 3. Multiplicatorul de ajustare a adaptării**

#### **COCOMO II Intrări Stagiul 3 (Post Architectură)**

- Pentru stagiul 3, ca în COCOMO 81, intrarea principală reprezintă dimensiunea din KSLOC.
- În plus, există **17 multiplicatori de efort**. Majoritatea sunt similari cu cei utilizate în COCOMO Intermediar 81, cu excepția noilor multiplicatori (fig. 5.5.2.2.e.)
  - Cerințe de reutilizare (RUSE)
  - Cerințe de documentare (DOCU)
  - Continuitatea personalului (PCON)
  - Dezvoltare de site-uri multiple (SITE)
- Platform Volatility (PVOL) înlocuiește VIRT (Virtual Machine Volatility) în COCOMO 81
- Platform Experience (PEXP) și Language and Tool Experience (LTEX) înlocuiește LEXP (Experiența în limbajul de programare) și VEXP (Experiența pe mașină virtuală).

- COCOMO II nu utilizează doi dintre multiplicatorii COCOMO 81 originale: MDOP (Utilizarea practicilor moderne de programare) și TURN (Timpul de return al computerului).
- Figura 5.5.2.2.e prezintă cele 17 atribute pentru Etapa 3 împreună cu evaluaările și valorile numerice ale acestora.

Nr	Atribute	Valori (Rating)					
		VL	LO	NM	HI	VH	XH
1	Fiabilitate cerută (Required Reliability) (RELY)	0.75	0.88	1.00	1.15	1.39	
2	Dimensiunea bazei de date (Database Size) (DATA)		0.93	1.00	1.09	1.19	
3	Complexitatea produsului (Product Complexity) (CPLX)	0.75	0.88	1.00	1.15	1.30	1.66
4	Cerințe de reutilizare (Required Reusability) (RUSE)		0.91	1.00	1.14	1.29	1.49
5	Cerințe de documentație (Documentation Required) (DOCU)	0.89	0.95	1.00	1.06	1.13	
6	Constrângeri ale timpului de execuție (Execution Time Constraints) (TIME)			1.00	1.11	1.31	1.67

7	Constrângeri de memorie (Main Storage) Constraint (STOR)			1.00	1.06	1.21	1.57
8	Volatilitatea platformei (Platform Volatility) (PVOL)		0.87	1.00	1.15	1.30	
9	Capabilitățile analiștilor (Analyst Capability) (ACAP)	1.50	1.22	1.00	0.83	0.67	
10	Experiența în aplicații (Applications Experience) (AEXP)	1.22	1.10	1.00	0.89	0.81	
11	Capabilitatea programatorilor (Programmer Capability) (PCAP)	1.37	1.16	1.00	0.87	0.74	
12	Continuitatea personalului (Personnel Continuity) (PCON)	1.24	1.10	1.00	0.92	0.84	
13	Experiența cu platforma (Platform Experience) (PEXP)	1.25	1.12	1.00	0.88	0.81	
14	Experiența cu limbajul și uneltele utilizate	1.22	1.10	1.00	0.91	0.84	

	Language and Tools Experience (LTEX)					
15	Utilizarea uneltelor software Use of Software Tools (TOOL)	1.24	1.12	1.00	0.86	0.72
16	Dezvoltarea în site-uri multiple (Multiple Site Development) (SITE)	1.25	1.10	1.00	0.92	0.84
17	Cerințe de planificare a dezvoltării Required Development Schedule (SCED)	1.29	1.10	1.00	1.00	1.00

**Fig. 5.5.2.2.e. COCOMO II. Stadiul 3. Multiplicatori de efort pentru dezvoltarea de software**

#### **COCOMO II Inputări Stagiul 2 Design timpuriu (Early Design)**

- Pentru Stagiul 2, intrarea principală o constituie dimensiunea exprimată în *puncte funcționale*.
- Modelul convertește punctele funcționale în KSLOC utilizând un tabel de limbaje de programare aşa cum este descris în Manualul de definire a modelului USC COCOMO II.
- Ecuatiile pentru Efort și Planificare sunt destul de asemănătoare cu cele aparținând Stagiului 3 (fig.5.5.2.2.a, 5.5.2.2.c)
- În schimb, factorul **EM** (Effort Multiplier) include 7 atribute care sunt fie replici, fie combinații ale celor 17 multiplicatori ai Stagiului 3.
- Cele șapte atribute ale Stagiului 2 sunt următoarele:
  - (1) *Fiabilitatea și complexitatea produsului* (RCPX): atributul RCPX este o combinație de RELY, CPLX, DATA și DOCU din stagiul 3.

- (2) Necesitatea reutilizării (RUSE): Atributul RUSE este același ca în stagiul 3.
- (3) Dificultatea platformei (PDIF): atributul PDIF este o combinație de TIME, STOR și PVOL din stagiul 3.
- (4) Capacitate de personal (PERS): Atributul PERS este o combinație de ACAP, PCAP și PCON din stagiul 3.
- (5) Experiența personalului (PREX): atributul PREX este o combinație de AEXP, PEEXP și LTEX din stagiul 3.
- (6) Facilități (FCIL): Atributul FCIL este o combinație de TOOL și SITE din stagiul 3.
- (7) Cerințe de planificare a dezvoltării (SCED): Atributul SCED este același ca în stagiul 3.

### **COCOMO II Intrări Stagiul 1 Componerea aplicațiilor (Application Composition)**

- Intrările din stagiul 1 includ noi **puncte de obiect (NOP)** și o evaluare a **productivității (PROD)**.
- **PROD** este o măsură a capacitatii și experienței dezvoltatorului cu instrumentele CASE.
- **PROD** presupune o valoare între 4 și 50, 4 fiind cele mai înalte niveluri de experiență și 50 reflectând cele mai scăzute niveluri de experiență.

### **COCOMO II Procesare (Stagiile 2, 3)**

- Ecuatiile pentru stagiile 2 și 3 au fost discutate anterior.
  - Ca și în COCOMO 81, provocările principale se referă la determinarea dimensiunii și la estimarea evaluărilor adecvate pentru atribute.
  - Pentru Stagiul 1, modelul folosește o ecuație simplă (fig. 5.5.2.2.f.).
- 

### **COCOMO II Procesare (Stadiul 1)**

**PM = NOP / PROD**, where

**NOP** = Puncte obiect noi.

**PROD** = Valoarea Productivității. Determinarea ambilor factori NOP și PROD poate fi provocatoare.

---

**Fig. 5.5.2.2.f. COCOMO II. Stagiul 1. Ecuația de estimare a efortului**

- Până la acest moment doar Stagiul 3 a fost automatizat în modelul computerizat USC COCOMO II.

### **COCOMO II Ieșiri**

- Rezultatele pentru stagiile 2 și 3 furnizate de COCOMO II sunt următoarele:
  - (1) Nivelul de efort (LOE) în luni-persoană pentru proiectul care se estimează.
  - (2) O planificare în luni.
  - (3) Efortul poate fi ușor convertit într-o valoare monetară dacă se cunoaște costul pe PM (lună-persoană).
- Pentru stagiul 1, singurul rezultat furnizat este efortul (adică, nu se furnizează nicio planificare).

### **COCOMO II Considerații privind costurile suport (Support Cost Considerations)**

- Nu există o ecuație a efortului de suport pentru stagiul 1.
  - Stagiile 2 și 3 ale COCOMO II folosesc o variantă de reutilizare a algoritmilor pentru a calcula întreținerea sau efortului de suport.
  - Din 1998, ecuațiile efortului de sprijin nu au fost incluse în ediția computerizată USC a COCOMO II.
  - Ecuația efortului de suport este prezentată în fig. 5.5.2.2.g.
- 

### **COCOMO II: Ecuația efortului de suport (Stagiile 2,3):**

$$PM_m = A(Size_m)^B(EM), \text{ unde}$$

$PM_m$  este efortul estimat pentru menenanță în persoană-luni.

A este un coeficient care este setat provizoriu la 2,5 (adică, valoarea implicită), dar care poate fi calibrat pentru a reflecta costurile și cultura organizației specifice.

$$\text{Size} = \text{Size}_M = (\text{Size Added} + \text{Size Modified})(MAF),$$

- MAF este factorul de ajustare al menenanței care se calculează după formula:

$$MAF = 1 + (UNFM) (SU/100), \text{ unde}$$

- **UNFM** = gradul de nefamiliaritate al programatorului cu software-ul menținut
- **SU** = cantitatea de înțelegere.

**B** este un exponent variabil cu ecuația:  **$B = 0.91 + 0.01 (SF)$** , unde

- **SF** este **suma** celor **5 factori de scală** care iau valori între 0 și 5.
- Cei **5 factori de scară**, enumerați în figura 2.6-7 sunt: PREC, FLEX, RESL, TEAM și PMAT.
- La fel ca multiplicatorii de efort din COCOMO 81 și COCOMO II, aceștia sunt evaluați de la VL la XH (foarte scăzut la foarte mare).
- Aceste valori sunt prezentate în fig.5.5.2.2.b, pentru evaluări mai favorabile rezultând valori mai mici și, prin urmare, un exponent de efort mai mic.
- Pentru PREC, de exemplu, VL corespunde „cu totul fără precedent”, în timp ce XH corespunde cu „foarte familiar”.
  - Figura 5.5.2.2.b. arată că pentru acest factor valoarea numerică pentru VL (4,05) este mai mare decât valoarea pentru XH (0,00).

**EM** este produsul a 17 multiplicatori de efort, aşa cum se arată în figura 5.5.2.2.e. Aceştia sunt similari cu cei utilizati în COCOMO 81 intermedier.

---

**Fig. 5.5.2.2.g.** COCOMO II. Stagiile 2,3. Ecuația de suport a efortului

### 5.5.2.3 Modelul PRICE S

- Modelul comercial PRICE S® a fost dezvoltat de PRICE Systems, LLC pentru a sprijini estimarea costurilor software.
- PRICE Systems, LLC a dezvoltat, de asemenea:
  - (1) Un model hardware, PRICE H®
  - (2) Un model de estimare a costurilor pentru operațiuni hardware și suport, PRICE HL®
  - (3) Un model pentru microcircuie și module electronice, PRICE M®.
- Modelul PRICE S® este parțial proprietar, deoarece toate ecuațiile nu sunt publicate, deși majoritatea sunt descrise în Manualul de referință PRICE S®.
  - Acest model este aplicabil tuturor tipurilor de proiecte software.

- Modelul ia în considerare toate fazele ciclului de viață al software-ului.
- Pe lângă fazele ciclului de viață al software-ului, se ia în considerare și fazele de concept al sistemului și de testare operațională.

## **PRICE S® Intrări**

- Principalele **intrări** pentru PRICE S® sunt grupate în următoarele nouă categorii:

### **(1) Mărimea proiectului (DIMENSIUNE):**

- Reflectă dimensiunea software-ului care urmează să fie dezvoltat sau suportat.
- Mărimea poate fi introdusă ca:
  - SLOC,
  - Puncte funcționale,
  - Puncte obiect predictive.

### **(2) Tipul aplicației proiectului software (APPL):**

- Oferă o măsură a tipului (sau a tipurilor) de software, descris de una dintre cele șapte categorii:
  - (1) Matematică,
  - (2) Manipularea sirurilor,
  - (3) Stocarea și recuperarea datelor,
  - (4) On-line,
  - (5) Timp real,
  - (6) Interactiv,
  - (7) Sistem de operare.

### **(3) Factorul de productivitate (PROFAC):**

- PROFAC este un parametru de calibrare care leagă programul software de productivitatea și eficiența personalului și practicile de management.

### **(4) Inventivitatea proiectării:**

- Se referă la elementele de noutate pe care le necesită proiectul

- Doi parametri indică cantitatea de noutate pentru fiecare tip de software.
  - Design nou (NOU)
  - Cod nou (NEWC)

**(5) Utilizare (UTIL):**

- Reflectă gradul de încărcare a procesorului în raport cu viteza și capacitatea memoriei.
- Valorile peste 50 la sută cresc de obicei efortul.

**(6) Specificațiile clientului și cerințele de fiabilitate:**

- Parametrul platformei (PLTFM) oferă o măsură a nivelului de testare și documentare care va fi necesară.

**(7) Mediul de dezvoltare:**

- Trei parametri de complexitate (CPLX1, CPLX2 și CPLXM) măsoară condițiile unice ale proiectului, cum ar fi dezvoltarea mai multor site-uri, volatilitatea cerințelor, utilizarea instrumentelor (de exemplu, instrumentele CASE) și alți factori.

**(8) Evaluări de dificultate pentru integrarea internă (INTEGI) respectiv integrarea externă (INTEGE).**

**(9) Procesul de dezvoltare:**

- Reflectă procesul de dezvoltare utilizat. Opțiunile includ dezvoltarea în cascadă, spirală, evolutivă și incrementală.
- În plus, există intrări specifice **activităților de suport software**. Acestea sunt discutate mai detaliat în paragrafele următoare.

## PRICE S® Procesare

(1) PRICE S® calculează **un volum de software (VOL)** pe baza **dimensiunii produsului** și a factorului **APPL** (Program Application).

(2) Apoi utilizează **VOL** și **PROFAC** (Factor de Productivitate) pentru a determina o estimare inițială a **efortului de dezvoltare în ore de muncă (LH)**.

- După cum se prezintă în Manualul de referință PRICE S®, această ecuație este:

$$LH = [ePROFAC] [VOLf(PROFAC)] / 1000$$

(3) Modelul ajustează apoi această estimare inițială prin parametrul **PLATFM** (Platform), care are un efect liniar asupra **LH** obținând estimarea de bază.

(4) Celealte intrări sunt utilizate pentru a face **ajustări suplimentare la estimarea de bază a LH** și pentru a construi **planificarea de dezvoltare**.

### **PRICE S® Ieșiri**

(1) PRICE S® calculează o estimare a **efortului** în luni-persoană care poate fi convertită în cost în dolari sau în alte unități valutare.

- **Efortul** este alocat în trei etape de dezvoltare software:
  - (1) Proiectare.
  - (2) Codificare.
  - (3) Testare.
- **Efortul** este, de asemenea, subdivizat în cinci activități:
  - (1) Ingineria sistemelor.
  - (2) Programare.
  - (3) Configurare și control al calității.
  - (4) Documentare.
  - (5) Managementul programului.

(2) PRICE S® calculează, de asemenea, o **planificare de dezvoltare** în **luni**.

- PRICE S® oferă o **opțiune de efecte de planificare** care compară o planificare furnizată ca intrare cu cea calculată de model.
- Această opțiune arată și penalizări pentru comprimarea planificării utilizatorului în comparație cu planificare prezisă de model.

(3) PRICE S® oferă mai multe **ieșiri optionale**, inclusiv:

- Complexitatea resurselor.
- Matrici de sensibilitate de aplicare a instrucțiilor (Instructions-application sensitivity matrices).
- Profiluri de cheltuieli cu resurse.

(4) PRICE S® oferă, de asemenea, o opțiune de ieșire „**la o privire**” pentru a vizualiza rapid efectele modificării parametrilor de intrare selectați.

- Această opțiune este utilă pentru efectuarea de studii comerciale de tip „ce ar fi dacă”.

### **PRICE S® Considerații privind costurile suport**

- Modelul ciclului de viață PRICE S® estimează:
  - Întreținere software;
  - Îmbunătățiri;
  - Creșteri;
  - Modificări ale costurilor provocate de achiziții și desfășurare (deployment).
- Intrările de desfășurare, care sunt intrări unice pentru suport, includ următoarele:
  - Suport de planificare (date de început și de sfârșit).
  - Număr de instalări.
  - Niveluri de creștere și îmbunătățire așteptate.
  - Nivelul de calitate software.
  - Factori de calibrare a productivității pentru întreținere, îmbunătățiri și creștere.
- Modelul PRICE S® oferă ieșiri pentru costurilor în trei categorii de suport:
  - (1) Întreținere (maintenance).
  - (2) Îmbunătățiri (enhancements).
  - (3) Creștere (growth).
- De asemenea, calculează **numărul de defecte livrate** în programul care urmează să fie suportat.
- Modelul alocă **efortul** sau **costurile** între cele **cinci activități** (de exemplu, ingineria sistemelor) descrise anterior.

#### **5.5.2.4 Modelul SEER SEM**

- SEER-SEM® (Software Evaluation and Estimation of Resources - Software Estimating Model) face parte dintr-o familie de instrumente oferite de Galorath Associates.
- Familia SEER mai include:
  - (1) Un model de estimare a costurilor hardware - SEER-H®
  - (2) Un model de ciclu de viață hardware - SEER-HLC®
  - (3) Un model de dimensionare software - SEER-SSM®

- (4) Un model pentru circuite integrate - SEER-IC®
- (5) Un instrument de proiectare pentru fabricație - SEER-DFM®.

- SEER-SEM® este parțial proprietar, deoarece nu toate ecuațiile sunt publicate.
- Cu toate acestea, unele relații sunt descrise în Manualul utilizatorului SEER-SEM®.
- SEER-SEM® este aplicabil la toate tipurile de programe, precum și la majoritatea fazelor ciclului de viață al dezvoltării software.

### **SEER-SEM® Intrări**

- Intrările SEER-SEM® pot fi împărțite în trei categorii:
  - (1) Dimensiune;
  - (2) Intrări pentru baza de cunoștințe;
  - (3) Parametri de intrare.
- **Intrările** SEER-SEM sunt descrise mai detaliat mai jos.
  - (1) **Dimensiune:**
    - Mărimea poate fi introdusă în unul dintre cele trei formate:
      - SLOC;
      - Puncte funcționale;
      - Proxi-uri - proxy-urile permit utilizatorului să-și specifice propria măsură de dimensiune, pe care modelul o convertește ulterior în SLOC.
    - În plus, toate programele software sunt clasificate astfel:
      - "Nou";
      - „Preexistă conceput pentru reutilizare”;
      - „Preexistă dar nu a fost conceput pentru reutilizare”.
    - Pentru software-ul preexistent, utilizatorii trebuie să specifiche cantitatea de software ștearsă, plus procentele de reproiectare, reimplementare și retestare necesare pentru modificarea sau reutilizarea programului pentru aplicația curentă.
    - Deoarece modelul utilizează Tehnica de evaluare și revizuire a programelor (PERT), utilizatorii trebuie să introducă o valoare

„minimă”, „cel mai probabil” și „maximum” pentru toate intrările de dimensiune.

## (2) Intrări pentru baza de cunoștințe:

- SEER-SEM® conține baze de cunoștințe pentru diferite tipuri de software.
- Bazele de cunoștințe atribuie valori implicate parametrilor de intrare descriși mai jos, în funcție de tipul de software selectat.
- Utilizatorii trebuie să abordeze aceste intrări pentru a specifica baza de cunoștințe care va fi utilizată de model:
  - **Platformă**: Mediul de operare al programului (de exemplu, avionică, la sol sau spațiu cu echipaj);
  - **Aplicație**: Funcția globală a software-ului (de exemplu, comandă și control, planificarea misiunii sau testare);
  - **Metoda de achiziție**: metoda prin care software-ul urmează să fie achiziționat (de exemplu, dezvoltare, modificare sau reproiectare);
  - **Metoda de dezvoltare**: Metoda utilizată pentru dezvoltare (de exemplu, cascadă, evolutivă sau spirală);
  - **Standard de dezvoltare**: standardul utilizat în dezvoltare și gradul de croitorie (de exemplu, arme MIL-STD-498, ANSI J-STD 016 complet, ANSI J-STD 016 nominal sau comercial);
  - **Clasă**: Această intrare este în primul rând pentru baze de cunoștințe definite de utilizator;
  - **Tipul de componentă COTS** (Commercial Off-The-Shelf): Tipul de program COTS, dacă există, cum ar fi biblioteca de clase, baza de date sau aplicații.
    - COTS se referă la activități asociate cu încorporarea componentelor software comerciale în activitățile de dezvoltare.
    - Aceasta înseamnă că sistemele software sunt construite utilizând pachete SW comerciale disponibile (COTS) achiziționate de la furnizori și integrându-le în abordarea „cumpărare și integrare” a aplicației.

## (3) Parametri de intrare:

- SEER-SEM® conține peste 30 de parametri de intrare, din care utilizatorii își pot rafina estimările.

- Similar cu COCOMO 81 și COCOMO II, valorile de intrare variază în general de la „foarte scăzut” la „foarte ridicat”.
- Ca și în ceea ce privește dimensiunea, utilizatorii trebuie să specifice o valoare „minim”, „cel mai mare” și „cel mai probabil” pentru fiecare intrare.
- Baza de cunoștințe selectată calculează valorile implicate pentru majoritatea parametrilor de intrare.
- Prin urmare, dacă utilizatorii nu sunt familiarizați cu un anumit parametru, pot utiliza valorile implicate ale bazei de cunoștințe.
- Categoriile primare de parametri de intrare și o scurtă descriere a fiecărui sunt următoarele:
  - (3.1) **Capacitatea și experiența personalului:** Cei 7 parametri din această categorie, similari „atributelor personalului” ale COCOMO 81, măsoară capabilitatea personalului utilizat în proiect. Aceste intrări sunt:
    - (3.1.1) Capacitatea de analist;
    - (3.1.2) Experiență în aplicații;
    - (3.1.3) Capacitate de programator;
    - (3.1.4) Experiență lingvistică;
    - (3.1.5) Experiență în sistem de dezvoltare gazdă;
    - (3.1.6) Experiență în sistemul țintă;
    - (3.1.7) Practici și Metode Experiență.
  - (3.2) **Mediul de sprijin pentru dezvoltare:** Cei 9 parametri din această categorie sunt similari cu atributele proiectului și cu unele dintre atributele computerului din COCOMO 81. Acestea includ:
    - (3.2.1) Utilizarea practicilor moderne de dezvoltare;
    - (3.2.2) Utilizarea instrumentelor automate;
    - (3.2.3) Timp de răspuns;
    - (3.2.4) Timpul de răspuns terminal;
    - (3.2.5) Dezvoltare de site-uri multiple;
    - (3.2.6) Dedicarea resurselor;
    - (3.2.7) Locația resurselor și suportului;

- (3.2.8) Volatilitatea sistemului gazdă;
  - (3.2.9) Volatilitatea sistemului țintă.
- (3.3) **Cerințe de dezvoltare a produsului:** Cei 5 parametri din această categorie includ:
  - (3.3.1) Volatilitatea cerințelor;
  - (3.3.2) Re-gazduire de la dezvoltare la computerul tinta;
  - (3.3.3) Nivel de specificare;
  - (3.3.4) Nivel de testare;
  - (3.3.5) Nivelul de asigurare a calității.
- (3.4) **Cerințe de reutilizare:** Cei 2 parametri din această categorie măsoară:
  - (4.1) Gradul de reutilizare necesar pentru programele viitoare;
  - (4.2) Procentul de software afectat de cerințele de reutilizare.
- (3.5) **Complexitatea mediului de dezvoltare:** Cei 4 parametri din această categorie măsoară:
  - (3.5.1) Complexitatea limbajului;
  - (3.5.2) Complexitatea sistemului de dezvoltare gazdă;
  - (3.5.3) Complexitatea clasei de aplicație;
  - (3.5.4) Impactul îmbunătățirilor procesului.
- (3.6) **Mediul țintă:** Cei 7 parametri din această categorie sunt similari cu unele dintre atrubutele computerului COCOMO 81, dar se concentrează pe computerul țintă. Acestea includ:
  - (3.6.1) Cerințe speciale de afișare;
  - (3.6.2) Constrângerile de memorie;
  - (3.6.3) Constrângerile de timp;
  - (3.6.4) Cod în timp real;

- (3.6.5) Complexitatea sistemului țintă;
- (3.6.6) Volatilitatea sistemului țintă;
- (3.6.7) Securitate (acesta este cel mai sensibil parametru de intrare din model).
- (3.7) Alți parametri de intrare: Există, de asemenea, intrări speciale pentru:
  - (3.7.1) Constrângerile de orar;
  - (3.7.2) Rate de muncă;
  - (3.7.3) Cerințe de integrare;
  - (3.7.4) Cheltuieli de personal;
  - (3.7.5) Metrici;
  - (3.7.6) Suport software.

## **SEER-SEM® Procesare**

- Deși modelul este proprietar, unele dintre ecuațiile modelului SEER-SEM® pot fi găsite în Manualul utilizatorului, precum și în alte articole publicate.
  - (1) Efortul estimat este proporțional cu **dimensiunea** crescută la un **factor de entropie**, care este nominal 1,2 (ca în COCOMO 81 încorporat), dar poate varia în funcție de opțiunile selectate de utilizator.
  - (2) Planificarea este estimată într-un mod similar, dar este mai puțin sensibilă la dimensiune.
- În SEER-SEM® **efortul estimat** repartizat pe programul estimat oferă un profil de personal.
  - Modelul poate găzdui multe profiliuri diferite de personal:
    - Includerea profilului tradițional Rayleigh-Norden;
    - Personal fix;
    - Profiluri mixte.
- **Constrângerile de personal** sunt evaluate în raport cu capacitatea proiectului de a absorbi personal.
- **Ajustările de productivitate** pot fi făcute pentru situații de supraîncărcare și subîncărcare.

- Înainte de a calcula efortul, planificarea și defectele, SEER-SEM® face mai multe calcule intermediare.
- Mărimea efectivă este calculată din codul nou și din cel reutilizat.
- O evaluare efectivă a tehnologiei (effective technology rating) (ETR) este calculată folosind parametrii de tehnologie și mediu, iar ratele de personal sunt determinate de complexitatea aplicației și de profilul de personal selectat.

### **SEER-SEM® ieșiri**

- SEER-SEM® permite utilizatorilor să selecteze o varietate de ieșiri.

(1) Modelul furnizează estimări de efort în categoriile:

- Management;
- Ingineria Sistemelor;
- Proiectare;
- Cod;
- Date;
- Test;
- Managementul configurației;
- Asigurarea calității.

(2) O estimare rapidă oferă o imagine instantanee a:

- Mărimea;
- Efort;
- Program;
- ETR (Effective Technology Rating);
- Alte ieșiri selectate.
  - Estimarea rapidă poate fi furnizată oricând în timpul procesului de estimare.

(3) Ieșirile optionale furnizate includ:

- O estimare de bază;
- Personal pe luni;

- Cost pe luni;
- Costul pe activități;
- Persoană-luni după activități;
- Defecte livrate;
- Un rating de maturitate SEI.

### **SEER-SEM® Considerații privind costurile suport**

- SEER-SEM® poate furniza optional un [raport de ieșire de „întreținere”](#) care furnizează [costul anual și lunile-persoană](#) pentru fiecare an dintr-o planificare specificată de utilizator, în patru categorii:
  - (1) Corecții;
  - (2) Adaptări;
  - (3) Perfective;
  - (4) Îmbunătățiri.
- Utilizatorul poate specifica perioada de timp de asistență dorită, împreună cu câțiva alți parametri unici ai suportului. Acestea includ:
  - Rata anuală de schimbare;
  - Numărul de site-uri de suport;
  - Creșterea preconizată a programului;
  - Diferențele dintre personalul de dezvoltare și de suport și ambient;
  - Constrângeri minime sau maxime de persona
  - Procentul de cod care trebuie menținut;
  - Nivelul de rigoare (nivel de sprijin).

#### **5.5.3 Calibrarea modelelor de estimare a costurilor**

- Prin definiție, [calibrarea ajustează](#) un model de estimare software parametric comercial la mediul unui utilizator specific.
- Cel puțin, calibrarea ar trebui efectuată dacă modelul va fi utilizat pentru a elabora estimări pentru propunerile care vor fi înaintate Guvernului sau unui contractant de nivel superior.

- Un studiu din 1981 al lui Thibodeau a dezvăluit că calibrarea poate îmbunătăți acuratețea modelului cu până la 400%.
- În plus, alte studii au arătat îmbunătășiri ale preciziei în diferite grade atunci când modelele sunt calibrate.
- În plus, majoritatea dezvoltatorilor de modele încurajează calibrarea modelului și de obicei oferă instrucțiuni pentru efectuarea acestui proces în manualele lor de utilizare.
- O prezentare generală a proceselor de calibrare pentru modelele discutate anterior este oferită mai jos.

### **Calibrarea COCOMO**

- Utilizatorii lui COCOMO 81 intermedier pot calibra fie:
  - (1) Doar coeficientul (de exemplu, 2,8 pentru categoria încorporată),
  - (2) Sau atât coeficientul, cât și exponentul (de exemplu, 2,8 și 1,20 pentru categoria încorporată), dacă sunt disponibile suficiente date istorice.
- În plus, COCOMO II are o capacitate de [calibrare on-line](#), aşa cum se discută în Manualul de referință COCOMO II.
  - Acest lucru permite utilizatorilor să calibreze **coeficienții** ecuațiilor de estimare a **efortului** și **planificării**.
  - Capacitatea on-line permite, de asemenea, utilizatorilor să calibreze simultan atât coeficienții de efort și de planificare cât și valorile exponentilor.
  - Cu toate acestea, această capacitate trebuie utilizată cu atenție, deoarece exponentii ecuației COCOMO II sunt variabili și depind de mai mulți factori care pot fi dificil de evaluat din datele istorice.

### **Calibrarea PRICE S®**

- Modelul PRICE S® trebuie calibrat pentru a ajusta parametrii selectați, astfel încât rezultatele generate de model să reflecte mediul utilizatorului.
- Cea mai comună calibrare este cea a PROFAC (Factor de Productivitate).
  - Conform Manualului de Referință PRICE S®, PROFAC trebuie să rămână constant pentru o anumită organizație, în special pe termen scurt.

- Pentru a calibra PROFAC pentru un CSCI, utilizatorul trebuie să introducă costul sau efortul real, programul de dezvoltare, PLTFM, INTEGI, UTIL și complexitatea managementului.
- De asemenea, este posibilă calibrarea platformei, a aplicației și a parametrilor interni selectați.

### **Calibrarea SEER-SEM®**

- SEER-SEM® conține un „mod de calibrare” care compară efortul real cu cel estimat și cifrele de program, apoi calculează patru cifre de calibrare sugerate.
- Utilizatorii trebuie să introducă intrările din baza de cunoștințe, efortul real, programul real, dimensiunea și dacă activitățile de analiză a cerințelor sau de testare a sistemului sunt incluse în efortul sau programul real.
- De asemenea, este de dorit, dar nu obligatoriu, ca utilizatorii să includă categorii de muncă și evaluări pentru oricât de mulți parametri (de exemplu, experiența aplicațiilor) sunt cunoscuți.
- Folosind informațiile introduse, modelul calculează **patru factori de ajustare a calibrării**:
  - (1) Ajustarea efortului;
  - (2) Ajustare planificării;
  - (3) Ajustarea tehnologiei;
  - (4) Ajustarea complexității.
- Factorii de efort și de ajustare a programului sunt multiplicatori liniari care nu modifică nicio intrare, ci scalează efortul de ieșire și programul.
  - De exemplu, un factor de ajustare a efortului de 1,19 va adăuga 19 la sută la efortul computerului.
- Multiplicatorii de tehnologie și complexitate ajustează factorii intermediari care, la rândul lor, ajustează intrările modelului selectat.
- Efectul lor asupra efortului și programului este neliniar.
- Poate fi cel mai bine pentru utilizatorii neexperimentați SEER-SEM® să folosească doar multiplicatori de efort și de programare.

### **5.6 Productivitatea activității software**

- **Productivitatea activității software** este o măsură a vitezei cu care inginerii implicați în activitatea de dezvoltare a unui proiect SW produc software individual și documentația asociată.
- De obicei, productivitatea software este exprimată în:
  - (1) LOC / unitate\_de\_timp
  - (2) Funcționalități / unitate\_de\_timp
- **Productivitatea SW** poate fi:
  - Neorientată spre calitate;
  - Orientată spre calitate.
- De obicei, valorile productivității bazate pe metrii de tip volum / unitate de timp, nu iau în considerare calitatea (QA).
- Productivitatea SW depinde în principal de:
  - (1) Nivelul limbajului de programare utilizat pentru dezvoltarea software-ului (limbaj de asamblare, limbaj de nivel înalt, limbaj specializat)
  - (2) Natura proiectului dezvoltat (sisteme încorporate în timp real, aplicații de sistem, aplicații comerciale)
  - (3) Natura și facilitatile mediului de dezvoltare.
- **Factori** care afectează **productivitatea**:
  - (1) **Experiența în domeniul aplicației.**
    - Cunoașterea domeniului aplicației este esențială pentru dezvoltarea eficientă a software-ului.
    - Inginerii care înțeleg deja un domeniu sunt mai probabil să fie cei mai productivi
  - (2) **Calitatea procesului.**
    - Procesul de dezvoltare poate avea un efect semnificativ asupra productivității.
  - (3) **Dimensiunea proiectului.**
    - Cu cât un proiect este mai mare, cu atât este nevoie de mai mult timp pentru comunicarea în echipă. Este mai puțin timp disponibil pentru dezvoltare, astfel încât productivitatea individuală este redusă
  - (4) **Suportul tehnologic.**

- Tehnologia de suport bună ca instrumente CASE, sistemele de gestionare a configurației de sprijin pot îmbunătăți productivitatea.
- (5) **Mediu de lucru.**
  - Un mediu de lucru liniștit cu zone de lucru private contribuie la îmbunătățirea productivității.
- Exemple de **productivitate software**:
  - **Sisteme RT incorporate**: 40 - 160 LOC/lună
  - **Programe sistem**: 150 - 200 LOC/lună
  - **Aplicații comerciale**: 200 - 800 LOC/luna.
- **Observație**: în acest moment nu se cunoaște o relație între **productivitate și calitate**.

## Capitolul 4. FAZA DE DEFINIRE

### Partea 3

#### 6. Ghid de estimare a proiectelor software

##### 6.1 Pașii de estimare a unui proiect software

- Estimarea dimensiunii jobului de făcut este probabil una dintre cele mai grele sarcini cu care se confruntă un manager.
- Nimici nu a reușit să vină cu o carte de bucate care să facă procesul de estimare mecanic sau automat
  - Dar există câțiva pași care pot fi urmați pentru a aborda problema cu un grad rezonabil de succes.
- Ce este o **estimare**?
  - (1) O **estimare** este aprecierea dezvoltatorului cu privire la **costul unui job** în termeni de **ani-om**, **timp calendaristic**, **ore-mașină** și alte resurse.
  - (2) O **estimare** este o declarație referitoare la modul în care dezvoltatorul **planifică** să **cheltuiască resursele** pentru a finaliza jobul.
  - (3) Când estimarea este transformată în **bani** și **timp calendaristic** și este adoptată pentru un **proiect**, ea se numește **buget**.
- **Planul unui proiect** trebuie să evolueze și să se schimbe ori de câte ori se schimbă condițiile.
  - Același lucru este valabil și pentru **estimarea** sau **bugetul** aferent: dacă au loc modificări care necesită mai multe resurse, proiectul trebuie să reestimat și bugetul modificat .
  - **Estimările** vor fi întotdeauna **imperfecte** și vor necesita să fie **rafinate** pe măsură ce proiectul se derulează.
    - În mod ideal, contractul de dezvoltare ar trebui să fie scris de o asemenea manieră, încât să permită estimarea fiecărei etape noi la sfârșitul celei precedente.
  - Iată o **modalitate de abordare a procesului de estimare**.
    - Deși această secțiune este plasată în mod logic aici, va fi mai semnificativă dacă va fi revizuită după ce ați studiat restul cursului.
- **PASUL 1. Se proiectează sistemul de programe.**
  - Ce înseamnă de fapt, *a proiecta sistemul?*"

- Adevărul este că sistemul poate fi proiectat la un anumit nivel de detaliu.
  - Designul ar putea avea doar patru pagini de diagrame care necesită o zi de efort, dar asta este mult mai bine decât nimic.
- În primul rând, trebuie acordat timpul necesar pentru a considera amploarea problemei de estimat.
  - Este practic imposibil să fie estimat costul dezvoltării a ceva ce nu este complet definit.
- De fapt, **cheia unei estimări de succes este înțelegerea problemei.**
  - Designul care se dezvoltă pentru primul exercițiu de estimare ar trebui să fie dus la un nivel la care există o idee conturată pentru toate problemele tehnice majore cu care se confruntă proiectul.
- Dacă se poate aloca suficient timp pentru a proiecta sistemul de programe în detaliu, este grozav.
  - Mult mai probabil, totuși, designul realizat (cel puțin pentru estimările inițiale) va fi la un nivel mult mai general.
- **PASUL 2. Se estimează dimensiunea totală a sistemului de programe care urmează să fie livrat.**
  - Prin **dimensiune** se înțelege **numărul final de linii de cod (LOC)** care vor exista, inclusiv codul care definește fișierele de date, precum și codul care definește instrucțiunile executabile ale mașinii.
  - În acest scop pot fi folosite diferitele tehnici și metode care au fost discutate.
    - De exemplu, pe baza tehnicii de jos-în-sus, dimensiunea poate fi determinată prin estimarea dimensiunii fiecărui modul din design și apoi adunându-le împreună.
  - Cu cât proiectul este mai detaliat, cu atât probabilitatea ca estimarea totală să fie mai precisă, este mai mare.
    - Dacă se intenționează codificarea în mai multe limbaje de programare (de exemplu, limbaj de asamblare și un limbaj de nivel înalt), trebuie determinat un total separat pentru fiecare.
  - Dacă se modifică un sistem existent, trebuie estimate liniile noi de cod care vor fi scrise.
- **PASUL 3. Se estimează necesarul de forță de programare pentru a produce numărul de linii de cod care a fost derivat (fig.6.a, prima coloană).**
  - Fiecare organizație are propriile reguli generale cu privire la numărul de instrucțiuni finalizate pe zi om sau lună om.
    - În comunitatea de programare, nu există publicate informații despre astfel de numere. Cea mai bună soluție o reprezintă

regulile informale existente în organizația care dezvoltă proiectul.

### ESTIMATING CHECKLIST GUIDE

MANPOWER	EQUIPMENT	MISCELLANEOUS
<ul style="list-style-type: none"> <li><input type="checkbox"/> Programming manpower</li> <li><input type="checkbox"/> Programmers</li> <li><input type="checkbox"/> Programming managers</li> <li><input type="checkbox"/> Programming responsible</li> <li><input type="checkbox"/> support manpower</li> <li><input type="checkbox"/> Analysts</li> <li><input type="checkbox"/> Designers</li> <li><input type="checkbox"/> Testers</li> <li><input type="checkbox"/> Managers</li> <li><input type="checkbox"/> Engineers</li> <li><input type="checkbox"/> Secretaries</li> <li><input type="checkbox"/> Instructors</li> <li><input type="checkbox"/> Administrative assistants</li> <li><input type="checkbox"/> Financial Assistants</li> <li><input type="checkbox"/> Couriers</li> <li><input type="checkbox"/> Consultants</li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Working Points (PC's)</li> <li><input type="checkbox"/> Users <ul style="list-style-type: none"> <li><input type="checkbox"/> Programmers</li> <li><input type="checkbox"/> Analysts and designers</li> <li><input type="checkbox"/> Testers</li> <li><input type="checkbox"/> Management</li> <li><input type="checkbox"/> Maintenance people</li> </ul> </li> <li><input type="checkbox"/> Uses <ul style="list-style-type: none"> <li><input type="checkbox"/> Editing programs and compilation</li> <li><input type="checkbox"/> Module test</li> <li><input type="checkbox"/> Integration test</li> <li><input type="checkbox"/> System test</li> <li><input type="checkbox"/> Acceptance test</li> <li><input type="checkbox"/> Site test</li> <li><input type="checkbox"/> Installation and support programs</li> <li><input type="checkbox"/> Simulation</li> <li><input type="checkbox"/> Program maintenance</li> <li><input type="checkbox"/> Documentation editing</li> <li><input type="checkbox"/> Reports editing</li> <li><input type="checkbox"/> Back-up</li> <li><input type="checkbox"/> Contingency reserve</li> </ul> </li> <li><input type="checkbox"/> Printers</li> <li><input type="checkbox"/> Special configuration (platforms)</li> <li><input type="checkbox"/> Users <ul style="list-style-type: none"> <li><input type="checkbox"/> Programmers</li> <li><input type="checkbox"/> Testers</li> </ul> </li> <li><input type="checkbox"/> Uses <ul style="list-style-type: none"> <li><input type="checkbox"/> Programming</li> <li><input type="checkbox"/> Test</li> </ul> </li> <li><input type="checkbox"/> Other equipment costs <ul style="list-style-type: none"> <li><input type="checkbox"/> Communication</li> <li><input type="checkbox"/> Internet</li> <li><input type="checkbox"/> Specialized Software</li> <li><input type="checkbox"/> SW Licenses</li> <li><input type="checkbox"/> Document reproduction equipment</li> <li><input type="checkbox"/> Streamers</li> </ul> </li> </ul>	<ul style="list-style-type: none"> <li><input type="checkbox"/> Physical facilities <ul style="list-style-type: none"> <li><input type="checkbox"/> General (office space, furniture, etc.)</li> <li><input type="checkbox"/> Special for the project</li> <li><input type="checkbox"/> Document storage</li> <li><input type="checkbox"/> Disk storage</li> <li><input type="checkbox"/> Classified storage</li> <li><input type="checkbox"/> Reproduction equipment area</li> </ul> </li> <li><input type="checkbox"/> Supplies <ul style="list-style-type: none"> <li><input type="checkbox"/> General (paper, pencils, etc.)</li> <li><input type="checkbox"/> Special for the project</li> <li><input type="checkbox"/> Printer paper</li> <li><input type="checkbox"/> Diskettes</li> <li><input type="checkbox"/> Carrying cases</li> </ul> </li> <li><input type="checkbox"/> Relocations</li> <li><input type="checkbox"/> Moving people</li> <li><input type="checkbox"/> Moving equipment and facilities</li> <li><input type="checkbox"/> Trips <ul style="list-style-type: none"> <li><input type="checkbox"/> Reasons <ul style="list-style-type: none"> <li><input type="checkbox"/> For computer time</li> <li><input type="checkbox"/> Visit customer</li> <li><input type="checkbox"/> Visit other contractors</li> <li><input type="checkbox"/> Attend professional meetings, symposium</li> </ul> </li> <li><input type="checkbox"/> Number of trips</li> <li><input type="checkbox"/> Number of people per trip</li> <li><input type="checkbox"/> Duration of trip</li> </ul> </li> <li><input type="checkbox"/> Special cost publication (publications outside the project)</li> <li><input type="checkbox"/> Other <ul style="list-style-type: none"> <li><input type="checkbox"/> Shuttle service</li> <li><input type="checkbox"/> Cars</li> <li><input type="checkbox"/> Drivers</li> <li><input type="checkbox"/> Leased systems</li> <li><input type="checkbox"/> Purchased systems</li> <li><input type="checkbox"/> Shift premium</li> <li><input type="checkbox"/> Overtime payments</li> <li><input type="checkbox"/> Per diem payments</li> <li><input type="checkbox"/> Special training aids</li> </ul> </li> </ul>

Fig. 6.a. Lista de estimări

- Dacă se utilizează numerele furnizate de altcineva, trebuie să lămurite următoarele probleme.
  - De exemplu, dacă cineva spune că o regulă generală bună este **cinci instrucțiuni pe zi om**, trebuie să cunoască câteva lucruri suplimentare despre acel număr:
  - Acoperă doar timpul programatorului?

- Include analiza problemelor și proiectarea de bază?
  - Include testul modulului?
  - Include testul de integrare?
  - Include testul sistem?
  - Include testul de acceptare?
  - Include documentarea?
  - Include timpul de management?
  - Include personalul de sprijin?
  - Ce limbaj este folosit?
- **PASUL 4. Se estimează necesarul de personal de suport.**
    - Elementele forței de muncă suport sunt prezentate în fig.6.a, prima coloană.
    - Nu toate elementele enumerate în această listă și în alte liste de verificare sunt neapărat aplicabile jobului estimat.
    - Mai târziu, pe parcursul cursului se vor oferi și valori numerice statistice referitoare la personalul suport, dar aici, din nou, propria experiență poate valora mult mai mult decât orice numere obținute dintr-o altă sursă.
  - **PASUL 5. Se estimează costurile echipamentelor.**
    - În figura 6.a, coloana doi, sunt enumerate diferitele categorii de echipamente și utilizatori care ar putea fi necesar să fie luate în considerare pentru jobul estimat.
      - Din nou, în capitolele următoare se vor detalia aspecte referitoare la echipamente.
  - **PASUL 6. Se estimează articolele referitoare la dotări fizice, rechizite, relocări, călătorii etc.**
    - Principalele articole din această categorie sunt enumerate în figura 6.a, coloana numărul 3.
    - Desigur, trebuie luate în considerare doar elementele care se aplică jobului estimat.
  - **PASUL 7. Odată dimensionat totul într-o primă aproximare, se revine și se adăugă acele elemente neprevăzute care pot afecta potențial organizația dezvoltatoare.**
    - De exemplu, este posibil să fi rezultat o cerință pentru 500 de luni-om de timp pentru programatori, dar programatorii se îmbolnăvesc, își iau vacanțe, renunță, sunt concediați, intră în serviciul militar și așa mai departe.
    - Aceste elemente pot reduce cu ușurință eficiența medie a personalului la locul de muncă cu 20%. Estimarea utilizării 100% a timpului de lucru a oricui, este o estimare eronată din start.
    - În urma unei analize se decide care ar trebui să fie cifrele pentru organizația dezvoltatoare și vor fi utilizate pentru a modifica

estimările inițiale (fie forța de muncă, fie timpul calendaristic, sau ambele).

- Toți factorii de contingență trebuie consemnați cu atenție, astfel încât cei care verifică ulterior estimarea realizată să fie edificați.

- **Pasul 8. Se consideră factorii agravanți**

- Factorii agravanți sunt listati în fig.6.b.
- Se bifează fiecare element care se presupune că se aplică jobului.
- Pentru fiecare articol avut în vedere se verifică dacă a fost luat deja în considerare în deviz.
  - Dacă nu, se decide dacă fiecare articol analizat are o valoare suficientă pentru a determina mărirea estimării.
- În fiecare caz, există diferite părți ale estimării care pot fi afectate, dar se recomandă limitarea în mod rezonabil la cele două elemente cele mai mari:
  - (1) **Forța de muncă**
  - (2) **Planificarea.**
- Lista este formulată astfel încât orice element verificat ar sugera în mod normal o creștere a estimărilor.
  - Pentru orice element care nu a fost verificat, poate însemna că estimarea nu ar trebui modificată, în ceea ce privește acel punct, sau poate sugera că ponderarea negativă este în regulă - adică ar putea scădea estimarea inițială.
  - Dar trebuie procedat cu mare atenție – estimările în această afacere sunt foarte rar prea mari.

## **PROJECT WEIGHTING FACTORS GUIDE**

- Vague job requirement
- Innovation required
- System will have more than one user
- System will be installed at more than one location
- System is Real-Time
- System is for military environment
- Interfaces with other systems are ill-defined or complex
- The program has interfaces with other programs
- You are to modify someone else's programs
- Your analysts have not worked on a similar application
- Your programmers have not worked on a similar application
- Your managers have not worked on a similar application
- The system is larger than those you have usually worked on
- You must share computer time with other projects
- You do not have the complete control of computer resources
- Customer has control of computer resources
- You are obliged to adhere to other's standards and procedures
- Customer will supply data base
- Customer will supply test data
- Data base is complex or not yet defined
- Data base is classified for security reasons
- The programmers must be trained on a new coding language
- The programmers must be trained on a new programming environment
- You must provide your own support program
- You must test the product under different platforms
- Your effort is split among several locations
- You have a high percentage of new or junior programmers
- Program dimensions are severely limited
- Your designers are not expert programmers
- Your confidence in personnel continuity is low
- You have little or no choice of personnel who work for you
- The customer must sign off on your design
- Other agencies must sign off on your design
- Customer is inexperienced in data processing
- Customer is experienced in data processing
- You must work on customer premises (cuts down on facility costs, constraints, etc)
- You expect much change during development, either in system requirements or in design constraints or in customer personnel
- The system has a large number of features
- The working environment promises many interruptions.

**Fig.6.b. Project Weighting Factors**

- **PASUL 9.** Se transformă totul în bani, aplicând rate salariale medii, rate de timp de mașină etc. pentru organizația dezvoltatoare.
  - Dacă de această sarcină se ocupă specialiștii în stabilirea prețurilor din organizație, trebuie verificat faptul că aceștia înțeleg ceea ce a fost inclus și pe de altă parte să facă cunoscut ceea ce adaugă.
    - De exemplu, ei pot adăuga în mod normal elementele neprevăzute menționate la Pasul 7.
- **PASUL 10.** Estimarea obținută până acum este **costul de bază** pentru lucrare, uneori numit **cost de fabrică** sau **cost direct**.
  - Acum, se adaugă profitul, cheltuielile generale și orice taxe care nu sunt deja incluse.
  - Aceasta este estimarea completă a costurilor finalizate pentru lucrare. Activitatea însă nu se finalizează aici.
- **PASUL 11.** Se consemnează toate ipotezele și presupunerile pe care se bazează estimarea.
  - Nu transmite nimănui estimarea fără a oferi ipotezele și presupunerile avute în vedere.
    - Dacă se consideră oportun, estimarea și ipotezele se capsează împreună.
  - Este posibil să se fi făcut unele presupuneri care nu sunt acceptate de către conducere sau care sunt imposibil de îndeplinit.
  - Estimarea poate fi total inutilă dacă se bazează pe o presupunere proastă.
  - Se recomandă să nu se includă (cu litere foarte mici) o presupunere despre care se știe că este imposibil de îndeplinit pentru a pune la adăpost estimatorii dacă proiectul va avea probleme mai târziu. Pe lângă faptul că este copilăresc și lipsit de etică, acest subrefugiu nu va funcționa.
- **PASUL 12.** Pe măsură ce lucrarea progresează prin ciclul de dezvoltare, se reestimează părți sau jobul în totalitate pe măsură ce se obțin inputuri din ce în ce mai bune.
  - Un punct important pentru reevaluare este la sfârșitul fazei de proiectare.
    - În acel moment, ar trebui să existe nu doar un **design complet al sistemului** ca bază pentru estimarea, ci și un **Plan de proiect solid**.

- Procesul de scriere a **Planului de proiect** va crește substanțial gradul de conștientizare a tuturor elementelor de cost cu care este posibil să vă confruntați.
- Noua estimare poate arăta de pur și simplu o nevoie de redistribuire a resurselor totale (de exemplu, folosind mai puțini oameni, dar mai mult timp mașină); acest lucru este de obicei de competență unui manager de proiect.
  - Dacă, totuși, noua estimare arată o nevoie de resurse suplimentare, este posibil să apară o **problemă**.
- Dacă se pot obține sau nu resurse suplimentare depinde de tipul de contract care s-a semnat și de înțelegerea și credibilitatea existente între dezvoltator și client.
  - În multe cazuri, se poate demonstra că a existat o schimbare în domeniul de activitate, iar clientul va modifica contractul pentru a reflecta modificările.
  - Adesea se poate ajunge la un acord în care cerințele programului vor fi adaptate pentru a se potrivi cu resursele pe care clientul le poate oferi.
- Probabil, la fel de des, se ajunge în situația de a lucra mai mult decât este plătit – un fapt care subliniază incapacitatea de a estima suficient de precis de prima dată.

## 6.2 Iстория проекта

- Probabil că una dintre cele mai neattractive activități în domeniul procesării datelor este aceea de a păstra înregistrări exacte ale estimărilor față de cheltuielile reale.
- Dificultatea de a desfășura această activitate la scară largă este că toată lumea funcționează diferit și este greu să se ajungă de acord asupra oricărei modalități standard de a ține evidență.
- Dar nu există niciun motiv pentru care o anumită organizație nu poate să compileze astfel de date, poate cu ajutorul unui sistem de control automatizat.
- Este extrem de important să se păstreze istoricul proiectelor unei companii care apoi să fie utilizate în planificarea joburilor viitoare.
  - În consecință, merită acordat timpul necesar pentru a defini ce înregistrări trebuie să păstrate ca parte a unui istoric al proiectului.
  - De asemenea, merită desemnată o persoană îsteață ca istoric.
- Fig.6.1.a. prezintă o schiță pentru un istoric al proiectului.

## **ISTORIA PROIECTULUI (MODEL)**

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

### **SECȚIUNEA 1: DOMENIU DE APLICARE**

*O declarație generală precizează intenția acestui document. Este de fapt o înregistrare istorică a evenimentelor importante și a datelor despre proiect pentru a fi utilizate în planificarea și estimarea fazelor ulterioare ale proiectului sau a proiectelor complet noi. Se dorește să fie un rezumat clar, structurat nu o colecție de hârtie.*

### **SECȚIUNEA 2: EVENIMENTE SEMNIFICATIVE**

*O listă cronologică și un rezumat foarte scurt al evenimentelor importante de pe durata desfășurării proiectului, inclusiv repere ratate, estimări noi, modificări de contract, revizuirile proiectului, date de instalare a echipamentelor, acorduri telefonice importante, întâlniri cu clientul, întâlniri cu subcontractanții, membrii echipei, sau vânzători.*

### **SECȚIUNEA 3: SUCCESSE MAJORE**

*Se menționează termenele limită majore respectate, profitabilitatea, satisfacția clienților, orice s-a desfășurat în foarte bune condiții.*

### **SECȚIUNEA 4: PROBLEME MAJORE**

*Se descriu problemele majore: programări ratate, depășiri de buget, probleme de performanță tehnică, de personal, de relații cu clienții, de suport managerial etc.*

### **SECȚIUNEA 5: ISTORICUL FORȚEI DE MUNCĂ**

*Se prezintă grafice care prezintă trei elemente majore:*

*5.1. Forța de muncă totală estimată (în luni om) la începutul contractului în fiecare dintre categoriile enumerate în Figura 6.a.*

*5.2. O înregistrare a modificărilor numerelor estimate prezentate în (5.1) și notării care explică motivele modificărilor.*

*5.3. Forța de muncă totală efectiv utilizată în timpul contractului în fiecare categorie enumerată la (5.1).*

*Totul trebuie prezentat în forma unui ghid pentru estimarea joburilor următoare.*

## **SECTIUNEA 6: ISTORICUL TIMPULUI MAȘINĂ**

*Secțiunea se utilizează doar dacă este necesar. Conține o serie de diagrame similare cu cele ale istoriei forței de muncă. Se realizează câte o diagramă pentru fiecare tip de mașină utilizată. Fiecare diagramă arată:*

**6.1. Total de ore estimate pentru această mașină la începutul contractului în următoarele categorii:**

- testare modul/integrare
- testare sistemului
- testare de acceptanță
- alte.

**6.2. O înregistrare a modificărilor estimărilor prezentate în (6.1) și notații care explică motivele modificărilor, inclusiv diferențele dintre configurațiile așteptate și cele reale.**

**6.3. Timpul total al mașinii utilizat efectiv în timpul contractului în fiecare categorie enumerată la (6.1).**

*O comparație post-mortem între (6.1) și (6.3), luând în considerare (6.2), ar trebui să ajute la estimările viitoare.*

## **SECTIUNEA 7: CONCLUZII**

*Principalele concluzii. Ce ar trebui făcut diferit dacă s-ar repeta același proiect.*

**Fig.6.1.a. Model de Istorie a proiectului**

- În activitatea practică poate fi utilizat acest model sau poate fi creat unul specific proiectului.
  - Este însă pur și simplu de evitat doar strângerea unei multimi de diagrame și documente, adunate într-un dosar și care este etichetat „Istoria proiectului”.
  - Astfel de teancuri de hârtii sunt inutile.
- Pentru ca setul informații din istoricul proiectului să fie de **ajutor** la următorul job, trebuie respectate două cerințe de bază:
  - (1) În primul rând, trebuie adoptat un **ciclu de dezvoltare** care să fie repectat de la un proiect la altul.

- (2) Când se examinează costurile cu forța de muncă sau cu timpul de calculator pentru diverse elemente precum „testul modulului”, „testul de integrare” și „testul sistemului”, acești termeni trebuie să semnifice același lucru pentru **jobul următor** care este estimat, după cum s-a procedat în **joburile anterioare** pentru care s-au păstrat istoricile.

## 7. Planul proiectului

### 7.1 Caracteristicile unui plan bun

- Un **plan** este o **foaie de parcurs** care arată cum poți ajunge de aici până acolo.
  - Asemenea unei foi de parcurs, el indică rute alternative, repere și distanțe.
- Iată câteva dintre cerințele pe care un plan bun trebuie să le îndeplinească :
  - (1) Este **un document scris**, și nu în capul managerului.
  - (2) Trebuie să descrie:
    - **Care** este jobul.
    - **Cum** va fi atacat de către dezvoltator.
    - **Resursele** necesare pentru a face acest lucru.
  - (3) Este **redactat cu grijă**, astfel încât să fie **lizibil**, nu doar o acumulare de hârtii ale căror relații între ele sunt obscure.
    - Trebuie acordată o atenție deosebită **continuității planului**.
  - (4) Permite **situatii neprevăzute**: precizează acțiunile care trebuie întreprinse în cazul în care ceva nu decurge conform planului.
    - Există două tipuri de **situatii de urgență** de luat în considerare:
      - (a) Primul implică probleme specifice, identificabile, care pot apărea.
        - De exemplu, dacă un computer planificat nu este disponibil la data specificată, de unde se va aloca timpul de testare, cu ce costuri suplimentare în timp și bani?
      - (b) Al doilea este mai dur: ce măsuri vor fi luate în cazul în care apar probleme neprevăzute?
        - În mod evident, răspunsurile specifice nu pot fi furnizate deoarece problemele specifice nu pot fi enunțate.
        - Managerul însă se poate pregăti pentru astfel de evenimente fiind realist în planificarea resurselor.

- Trebuie presupus întotdeauna că oamenii se vor îmbolnăvi, vor părăsi compania, vor înțelege greșit o specificație, vor face greșeli.
- Trebuie presupus de asemenea că întotdeauna mașinile se pot defecta.
- Întotdeauna trebuieesc asumate neînțelegeri cu clientul.
- Planificarea în sine este concepută pentru ca totul să funcționeze perfect, dar este evident că acest lucru nu se va întâmpla niciodată.
- În consecință trebuieesc prevăzute în plan resurse suplimentare pentru a acoperi obstacolele neplanificate.
- (5) Este **modular**.
  - Dacă cărțile nu ar fi scrise în părți, capitole și paragrafe, citirea lor ar fi obosităre.
  - Același lucru este valabil pentru orice document, inclusiv pentru un plan.
  - Documentul trebuie să fie subdivizat logic astfel încât să existe un flux logic rezonabil de la o secțiune la alta, dar astfel încât, fiecare secțiune să-și păstreze în continuare identitatea și să fie utilă de sine stătător.
- (6) Este **suficient de scurt** încât să nu-i îndepărteze pe oameni. Teancurile de hârtii pur și simplu nu vor fi citite.
  - Planul poate fi doar câte o pagină pentru fiecare secțiune.
  - Mai probabil, pentru un proiect de dimensiune medie, va necesita un caiet de poate patruzeci sau cincizeci de pagini. Nu este prea mult pentru a cere fiecărui membru al proiectului să îl citească.
  - După ce au citit planul o dată în integralitate, majoritatea membrilor echipei vor trebui să se refere doar la anumite secțiuni ale planului și aceasta din când în când.
  - Cât de mult crește planul proiectului depinde parțial de cât de multe tutoriale sunt incluse în el.
  - Este nevoie de multă abilitate pentru a defini termenii și pentru a ghida cititorul prin plan, dar trebuie reținut că o cantitate prea mare de informație va face ca acesta să rămână necitit și neutilizat.
- (7) Are un **index**.

- Este nevoie doar de puțin timp pentru a construi un **index alfabetic** decent, iar includerea acestuia va spori foarte mult utilitatea documentului.
- Pentru a îndeplini toate cerințele de mai sus, se recomandă să se înceapă cu un **plan model** și element care va garanta un start bun al proiectului.

## 7.2 Redactarea planului proiectului

- (1) În mod normal, **planificarea** nu începe până când nu există **un contract semnat și proiectul este în derulare**.
  - În practică, însă, acest lucru se întâmplă foarte rar.
  - Faza de definire a proiectului, de obicei, a fost precedată de un număr semnificativ de activități de planificare:
    - Studiile proprii și eforturile de propunere.
    - Studii finanțate de clienți.
    - Contracte anterioare conexe.
    - Discuții între dezvoltator și potențialul client.
  - **Activitatea de planificare** poate fi considerată doar un pas într-un **proces evolutiv**, iar această evoluție nu se termină atunci când se încheie Faza de Definire.
  - Indiferent cât de bine a fost conceput un plan, acesta se va schimba de-a lungul vieții proiectului.
    - Chiar și așa, planul pe care este redactat în timpul fazei de definire trebuie să fie cât mai complet și autonom în măsura în care este posibil acest moment.
- (2) **Echipa** care contribuie la redactarea **Planului de proiect** ar trebui să includă:
  - **Analisti, programatori și manageri**, care trebuie să aibă toti continuitate în responsabilitatea asupra proiectului.
  - **Utilizatorul** trebuie de asemenea, fie să participe, fie cel puțin să ofere informații.
    - Dacă se negligează oricare cei de mai sus, planul va fi probabil nerealist.

- Un singur individ arbitrează și are ultimul cuvânt în zonele de dispută: acesta este **managerul de proiect**.
- Dacă managerul de proiect delegă problema și cere grupului de planificare să conceapă un plan pe care ulterior să îl stampileze, este o pierdere de timp pentru toată lumea.
- Dacă managerul nu crede cu adevărat în planificare și nu intenționează să utilizeze planul pentru a vă ghida desfășurarea proiectului, înseamnă că este un manager dinamic bazat pe intuiție. Mult noroc, fără însă a avea șanse reale de a finaliza cu succes proiectul!
- (3) Echipa de planificare ar trebui să fie **cât mai mică posibil** pentru a reduce **interacțiunile** și pentru a produce un **plan unitar**.
  - Planul nu trebuie să apară ca și cum ar fi fost scris de o duzină de indivizi diferiți care nu vorbesc niciodată între ei.
- (4) Pentru a demara activitatea de planificare, mai întâi trebuie stabilit **un cadru general**.
  - Recomandarea este de a porni pur și simplu de la schița planului model prezentat în acest curs și de a o adapta pentru a se potrivi proprietelor idei.
  - În continuare, trebuie estimat cât timp poate fi alocat activității de planificare.
- (5) Pentru redacta planul trebuie luate în considerare **toate persoanele disponibile în organizație**, cu talentele, punctele lor forte și punctele lor slabe și cărora li se atribuie spre redactare părți adecvate ale planului împreună cu termene clare de finalizare.
  - Modelul de plan prezentat în acest curs este împărțit în secțiuni, sau subplanuri, element care oferă o bază utilă pentru această activitate.
- (6) După ce sarcinile de planificare au fost repartizate, se organizează o **întâlnire de lansare** la care participă toți cei care lucrează pentru sau în acest proiect.
  - (a) Cu toții împreună, atât planificatorii, cât și analiștii, în aceeași cameră, în același timp, vor defini și discuta obiectivele proiectului.
  - (b) Apoi vor fi discutate în amănunt orice constrângeri sub care trebuie să se desfășoare proiectul, cum ar fi termenele limită fixe, reperele impuse de client, finanțare și locația de dezvoltare.
  - (c) Se stabilesc și analizează riscurile legate de proiect.

- (d) Odată aceste aspecte elucidate, se procesează la atribuirea secțiunilor Planului de proiect persoanelor fizice.
- (e) Dacă acest lucru a fost făcut de manager înainte de întâlnire, acesta trebuie să se asigure că toată lumea cunoaște nu numai misiunea personală ci și sarcinile celorlalți membri ai proiectului.
- (f) Se consemnează numele, sarcinile și datele stabilite și se oferă tuturor celor prezenți o copie a listei.
- (7) Întâlnirea de lansare trebuie urmată de **întâlniri periodice de lucru** în care planificatorii și analiștii își pot descrie progresul și în care se poate evalua starea generală a planului în raport cu datele țintă asumate.
- (8) **Fazelor de definire și de proiectare împreună**, ar trebui să le fie alocate, de la aproximativ **un sfert până la jumătate** din timpul total al contractului.
  - Planificarea este o activitate majoră în ambele faze. Uneori există tentația de a neglija activitatea de planificare, deoarece se așteaptă cu nerăbdare demararea activității de programare.
  - Dacă însă nu se alocă suficientă atenție și suficient timp pentru a întocmi un plan de calitate, se va plăti mai mult în fazele ulterioare, când lucrurile se pot complica:
    - Testarea va evidenția probleme, programele vor avea erori, bugetele vor fi depășite și foarte probabil calitatea produsului va avea de suferit.
    - Poate că pe manager nu îl deranjează în mod deosebit haosul unui proiect prost planificat, dar trebuie să se gândească și la oamenii lui.
    - Dacă aceștia sunt lipsiți de o experiență de lucru eficientă și placută, sau dacă viața lor privată este grav afectată din cauza unei planificări defectuoase a proiectului, gestionarea jobului este un eșec.

### **7.3 Un model de Plan de proiect**

- Metzger a încercat mai multe structuri diferite de planuri înainte de a se stabili la modelul prezentat în acest curs.
- Dacă pare util, modelul poate fi folosit ca atare, dar el poate fi modificat în orice mod pentru a se potrivi nevoilor sau chiar temperamentului managerului. La urma urmării, totul se reduce la eficiență.
- **Planul de proiect** este împărțit în următoarele secțiuni, care sunt rezumate în paragrafele următoare:
  - (1) **Prezentarea generală a planului**

- (2) Planul de faze
- (3) Planul de organizare
- (4) Planul de testare
- (5) Planul de control a modificărilor
- (6) Planul de documentare
- (7) Planul de formare
- (8) Planul de revizuire și raportare
- (9) Planul de instalare și exploatare
- (10) Planul de resurse și rezultate
- (11) Index

### **7.3.1 Prezentare generală**

- Secțiunea de prezentare generală a planului are câteva scopuri importante:
  - (1) În primul rând, pornind de la premiza că **nu cunoaște nimic despre proiect**, se prezintă **jobul și clientul** implicați.
  - (2) În al doilea rând, se descrie **organizarea generală a planului**.
  - (3) Se prezintă **ipotezele și restricțiile** pe care se bazează planul.
  - (4) Se stabilește o **planificare grosieră** pentru proiect.
  - (5) Se abordează pe scurt, **aspectele tehnice**.
  - (6) Se realizează o **analiză de risc** a proiectului.
  - (7) **Se rezumă** întregul plan oferind o **descriere sintetică** a componentelor planului care vor fi detaliate în secțiunile următoare. (fig.7.3.1.a).

---

## **SECȚIUNEA 1**

### **PREZENTAREA GENERALĂ A PLANULUI**

**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

-----

## **1.1 OBIECTIV**

*Obiectivul acestei secțiuni este de a rezuma întregul Plan al proiectului.*

## **1.2 DISCUȚIE**

- *Se prezintă contextul. Se identifică clientul și experiența acestuia în domeniu. Se descrie într-unul sau două paragrafe jobul de realizat.*
  - *De ce se realizează acest proiect?*
  - *Care sunt livrabilele proiectului?*
  - *Se prezintă toate informațiile necesare pentru o cât mai bună înțelegere a contextului jobului. În continuare se precizează obiectivele proiectului conform contractului.*
- *În continuare se explică cum este organizat planul. Trebuie sugerat cititorului că poate obține o bună înțelegere a planului citind Obiectivele și Discuțiile fiecărei secțiuni și acest lucru trebuie dus la îndeplinire.*
  - *Va fi folosită o abordare de echipă sau una bazată pe sarcini individuale?*
  - *Care vor fi responsabilitățile pentru echipă sau individ?*
  - *Cui îi va raporta echipa sau individul?*
  - *Cum și unde vor fi folosiți subcontractanții?*
  - *Ce hardware și software sunt necesare?*
  - *Ce alte produse sunt implicate?*
  - *La ce alte documente ar trebui să se facă referire?*
- *În continuare se enumera ipotezele și restricțiile pe care se bazează planul. Sunt extrem de importante. Nu trebuie ascunse într-o mulțime de cuvinte. Trebuie enunțate direct, sincer și simplu.*
- *Se prezintă o planificare grosieră pentru proiect. Această planificare ar trebui să arate toate eforturile majore care au legătură cu acest proiect, indiferent dacă sunt sub controlul dezvoltatorului sau nu.*
  - *Ce sarcini vor fi implicate?*
  - *Cine va fi repartizat fiecărei sarcini?*
  - *Când va începe fiecare sarcină?*

- Cât timp va dura fiecare sarcină?
- În ce ordine trebuie efectuate sarcinile?
- Cum va fi actualizată planificarea?

- **Se realizează o referire pe scurt la aspectele tehnice:**

- Ce limbaj de programare va fi folosit?
- Ce platformă va fi folosită?
- Ce metode de dezvoltare, instrumente, tehnici, reguli, practici și convenții vor fi urmate?
- Cine va efectua revizuirea codului?
- Cum vor fi efectuate revizuirile de cod?
- Care sunt cerințele de documentare?

- **La sfârșit, se realizează o analiză de risc a proiectului.**

- • Care sunt riscurile asociate proiectului. De detaliat acolo unde este necesar:
  - Programe și bugete nerealiste
  - Suprainginerie
  - Subcontractanții care nu au gradul necesar de asigurare a calității
  - Deficiențele în sarcinile efectuate de externi. Aici se includ sarcinile software și hardware efectuate de către subcontractanți precum și sarcinile efectuate de către client
  - Deficiențe de personal
  - Personalul este echipat tehnic pentru a îndeplini sarcinile care îi sunt atribuite?
  - Volatilitatea cerințelor - Cerințele pentru sistem se vor schimba pe parcursul proiectului?
  - Probleme de performanță / Probleme de scalare - Este suficient răspunsul în timp real pentru proiectele care implică acces la baze de date mari? Tensiuni referitoare la tehnologia informatică actuală - Proiectul folosește tehnologii neverificate?
- Care este gradul de risc?
- Ce măsuri vor fi luate pentru a reduce riscul?
- Ce abilități trebuie să aibă dezvoltatorii?
- Cum va fi monitorizat progresul proiectului?

### **1.3 DETALII**

*Se oferă o scurtă descriere a obiectivelor fiecareia dintre secțiunile rămase ale Planului de proiect.*

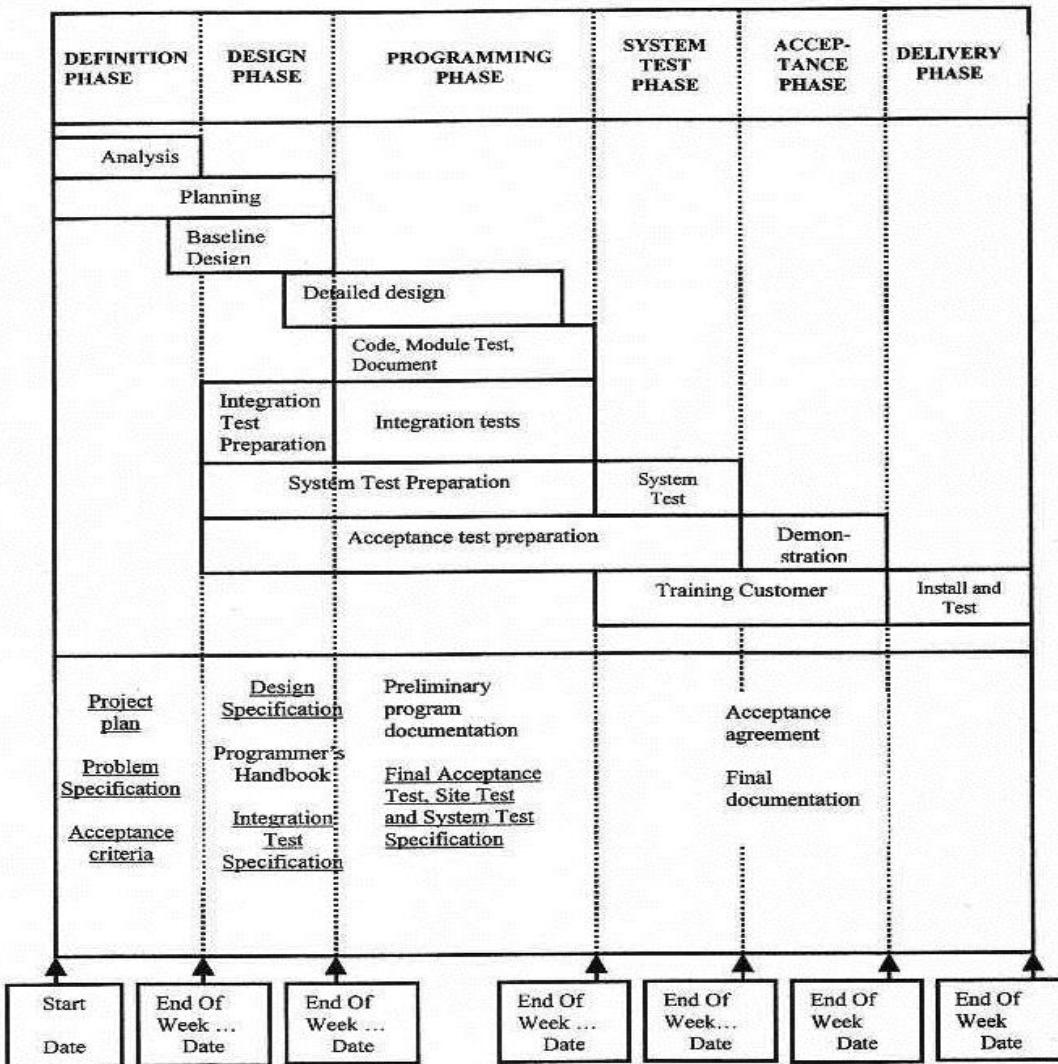
---

**Fig.7.3.1.a.** Model de Prezentare generală a planului

### **7.3.2 Planul de faze**

- Obiectivul acestei secțiuni este acela de a defini **ciclul de dezvoltare a proiectului**.
- Planul de faze servește ca bază pentru elementele ulterioare ale planului.
  - Este foarte recomandat să se adopte un ciclu de dezvoltare precum cel prezentat în acest curs sau un altul.
- Pentru **fiecare fază** a ciclului de viață, se descriu **obiectivele primare** și cele **secundare**.
  - Planul de fază trebuie să se încheie cu un grafic similar cu fig. 7.3.2.a dar cu datele planificate incluse.
- Un model pentru **Planului de faze** este prezentat în fig. 7.3.2.b.
- Planul de faze reprezintă o bază, un punct de referință al întregului proiect
  - De exemplu, când se discută despre faza de testare a sistemului, ar trebui toți cei implicați să discute despre același lucru.
  - Din păcate, acest lucru se întâmplă rar într-un proiect și acesta este un lucru foarte grav. Astfel de situații conduc la multe confuzii și neînțelegeri care altfel ar putea fi ușor evitate.

### The Model of Project Life Cycle



**Fig. 7.3.2.a.** Model de diagramă pentru Ciclul de viață al proiectului

**SECȚIUNEA 2**  
**PLANUL DE FAZE**  
**(MODEL)**

**DEPARTMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

-----

## **2.1 OBIECTIV**

*Obiectivul acestei secțiuni este acela de a defini efortul de dezvoltare a planificării în termenii unei serii de intervale de timp numite „faze”.*

## **2.2 DISCUȚIE**

*Se definește ciclul de dezvoltare și se descrie pe scurt fiecare fază care alcătuiește ciclul. Se include o diagramă ca cea din fig. 7.3.2.a. dar care include și date calendaristice. Se stabilesc definițiile de bază și se subliniază că secțiunile rămase ale planului sunt legate de aceste definiții. Dacă se intenționează planificarea mai multor versiuni, se arată cum sunt ele planificate, eventual într-o serie de cicluri de dezvoltare suprapuse, în esență identice.*

## **2.3 DETALII**

*Pentru fiecare fază se enumera obiectivele primare și secundare și se definește fiecare obiectiv cât mai riguros posibil*

### **2.3.1 Faza de definire**

#### **2.3.1.1 Obiective primare**

- (a) Analiza problemei
- (b) Planificarea detaliată a proiectului
- (c) Definirea criteriilor de acceptare

#### **2.3.1.2 Obiective secundare**

- (a) Găsirea de personal
- (b) Înțelegerea clientului
- (c) Formarea unor idei de proiectare provizorii

### **2.3.2 Faza de proiectare**

#### **2.3.2.1 Obiective primare**

(a) Designul de bază pentru programele operaționale

(b) Designul de bază pentru programele suport

#### **2.3.2.2 Obiective secundare**

(a) Pregătirea pentru testarea de integrare

(b) Configurarea controlului modificărilor

(c) Construirea modelelor de simulare

(d) Planificarea fazelor ulterioare

(e) Pregătirea pentru formarea programatorilor

(f) Publicarea manualului programatorului

(g) Pregătirea inițială pentru testarea sistemului

(h) Pregătirea inițială pentru testul de acceptanță

(i) Pregătirea inițială pentru testarea pe teren

(j) Crearea bibliotecilor proiectului

### **2.3.3 Faza de programare**

#### **2.3.3.1 Obiective primare**

(a) Proiectare detaliată

(b) Codificare

(c) Testul modulului

(d) Test de integrare

(e) Documentația programului

#### **2.3.3.2 Obiective secundare**

(a) Pregătirea detaliată pentru testarea sistemului

(b) Pregătirea detaliată pentru testul de acceptanță

(c) Pregătirea detaliată pentru testarea pe teren

(d) Pregătirea pentru instruirea clienților

### **2.3.4 Faza de testare a sistemului**

#### **2.3.4.1 Obiective primare**

(a) Testarea conform specificației problemei

- (b) Testarea cât mai „în direct” posibil
- (c) Testarea nu este controlată de programatorii

#### 2.3.4.2 Obiective secundare

- (a) Finalizarea pregăririlor pentru testul de acceptanță
- (b) Instruirea clienților
- (c) Corectarea documentației descriptive
- (d) Completarea documentației utilizatorului
- (e) Redistribuirea persoanelor

### 2.3.5 Faza de acceptare

#### 2.3.5.1 Obiective primare

- (a) Executarea și analiza testelor de acceptanță
- (b) Semnarea acordului formal de acceptare

#### 2.3.5.2 Obiective secundare

- (a) Finalizarea instruirii clienților
- (b) Definitivarea documentației

### 2.3.6 Faza de instalare și exploatare

#### 2.3.6.1 Obiective primare

- (a) Asistență la instalarea sistemului
- (b) Asistență pentru începerea funcționării

#### 2.3.6.2 Obiective secundare

- (a) Testare la fața locului
- (b) Întreținerea și reglarea continuă
- (c) Continuarea funcționării
- (d) Evaluarea proiectului

## 2.4 DIAGRAMA SARCINILOR (TASKURILOR)

Aici se introduce o diagramă de sarcini pentru proiect (cum poate fi cea produsă de MS Project sau de Open Project, de exemplu).

---

**Fig. 7.3.2.b.** Model pentru Planul fazelor

### **7.3.3 Planul de organizare**

- **Planului de organizare** trebuie să definească:
  - (1) **Organizarea** pe parcursul diferitelor faze ale proiectului.
  - (2) **Responsabilitățile specifice** fiecărui grup din cadrul organizației.
- Modelul Planului de organizare este prezentat în fig. 7.3.3.a.
  - (1) Se prezintă mai întâi grupurile și responsabilitățile generale ale acestora.
  - (2) Pentru fiecare fază a ciclului de dezvoltare sunt detaliate organizarea specifică și responsabilitățile grupurilor.

<b>SECTIUNEA 3</b>
<b>PLANUL DE ORGANIZARE</b>
<b>(MODEL)</b>
<b>DEPARTAMENT:</b>
<b>PROIECT:</b>
<b>NUMĂR DOCUMENT:</b>
<b>APROBĂRI:</b>
<b>DATA EMITERII:</b>
<b>REALIZAT DE:</b>

#### **3.1 OBIECTIVE**

*Obiectivul acestei secțiuni este acela de a defini organizarea proiectului și alocarea responsabilităților.*

#### **3.2 DISCUȚIE**

*Se precizează rațiunile care stau la baza organizării: claritatea atribuirii sarcinilor, minimizarea interacțiunilor, controlul modificărilor, stabilirea punctelor de*

*responsabilitate și de concentrare. Se schițează fluxul principal de lucru în cadrul organizației, începând cu probleme de analiza și proiectare și trecând prin programare, testare, documentare și livrare.*

### **3.3 DETALII**

*În prima subsecțiune se enumera grupurile care se vor regăsi pe organigramele și responsabilitățile generale ale fiecărui grup. Apoi se prezintă câte o organigramă pentru fiecare fază. Organizarea nu va fi, în general, aceeași pentru toate fazele; de exemplu, în timpul Fazei de Definire nu va exista încă un Grup de Programare.*

#### **3.3.1 Grupuri și responsabilități generale**

##### **3.3.1.1 Grupul de analiză și proiectare**

- (a) Scrierea specificației problemei
- (b) Scrierea specificației de proiectare
- (c) Controlul modificărilor
- (d) Controlul datelor
- (e) Modelare prin simulare
- (f) Inspecții de proiectare și codare
- (g) Redactarea documentației utilizator

##### **3.3.1.2 Grupul de programare**

- (a) Proiectare detaliată
- (b) Codificare
- (c) Testul modulului
- (d) Test de integrare
- (e) Documentație descriptivă

##### **3.3.1.3 Grupul de testare**

- (a) Scrierea specificațiilor de testare a sistemului
- (b) Scrierea specificațiilor de acceptare și de testare site
- (c) Validarea cazurilor de testare
- (d) Colectarea și generarea datelor de testare
- (e) Alegerea și obținerea instrumentelor de testare
- (f) Configurarea bibliotecilor de testare
- (g) Programarea resurselor de testare
- (h) Executarea testelor
- (i) Analiza rezultatelor testelor

(j) Documentarea rezultatelor testelor

#### **3.3.1.4 Grupul de personal (staff)**

- (a) Servicii de bibliotecă
- (b) Controlul timpului computerizat
- (c) Furnizarea de servicii de dactilografiere
- (d) Planificarea și instalarea terminalelor
- (e) Emiterea manualului programatorului
- (f) Instruire
- (g) Misiuni tehnice speciale
- (h) Legătura tehnică
- (i) Controlul documentelor
- (j) Controlul raportărilor
- (k) Controlul modificării contractului
- (l) Furnizarea de sprijin în activități de birou
- (m) Menținerea istoricului proiectului

### **3.3.2 Organizare și Responsabilități:**

#### **Faza de definire**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de definire*

### **3.3.3 Organizare și Responsabilități:**

#### **Faza de design**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de design*

### **3.3.4 Organizare și Responsabilități:**

#### **Faza de programare**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de programare*

### **3.3.5 Organizare și Responsabilități:**

#### **Faza testare sistem**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de testare sistem*

### **3.3.6 Organizare și Responsabilități:**

#### **Faza de acceptanță**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de acceptanță*

### **3.3.7 Organizare și Responsabilități:**

#### **Faza de instalare și operare**

*Se precizează organizarea și responsabilitățile specifice grupurilor de personal în faza de instalare și operare*

**Fig. 7.3.3.a. Model de Plan de organizare**

- Planul de organizare trebuie reorganizat din când în când din următoarele motive:
  - (1) Pe măsură ce proiectul trece de la o fază la alta, accentul se mută de la analiză la proiectare la programare și apoi la testarea sistemului.
    - Organizarea trebuie în principiu, să se schimbe odată cu munca.
    - De exemplu, nu este nevoie de un grup de instalare încă din faza de definire și poate să nu fie nevoie de un grup de analiză a cerințelor în timpul fazei de instalare și exploatare.
  - (2) Organizarea trebuie realizată în concordanță cu personalul la dispoziție.
    - Dacă un nou manager începe să lucreze pentru organizație la jumătatea proiectului și dacă are sentimente puternice cu privire la modul în care ar trebui să fie organizat finalul său de proiect, trebuie ascultat și încercată organizarea potrivit lui.
    - Nu este nimic în neregulă, dacă prin aceasta managerul își crește implicarea sporind eficiența și nu deranjează pe alții.
  - (3) Se recomandă utilizarea tehnologiilor de dezvoltare incrementală și/sau iterativă.
    - În acest caz, este posibil să existe un plan de organizare specific pentru fiecare iterație.
    - Dacă organizarea planificată și care a fost adoptată pur și simplu nu funcționează fără probleme, atunci, cu certitudine, trebuie schimbată.

### **7.3.4 Planul de testare**

- Această secțiune descrie **instrumentele, procedurile și responsabilitățile** pentru efectuarea tuturor nivelurilor de testare a proiectului. (Fig. 7.3.4.a.)
- Planul de testare trebuie să definească în mod clar:
  - (1) **Nivelurile de testare** (de exemplu, „testare modul”, „testare de integrare”, „testare sistem”, „testare de acceptanță”, „testare de site”).
  - (2) **Responsabilitatea** pentru executarea fiecărui nivel.
  - (3) **Suportul mașină** necesar pentru fiecare nivel.
  - (4) **Programele suport sau instrumentele** necesare.
  - (5) **Raportarea** rezultatelor testelor.
- Pentru **fiecare nivel de testare**, Planul de testare trebuie să definească:
  - (1) Obiectivele testului.
  - (2) Responsabilitatea desfășurării testului.
  - (3) Procedurile de testare.
  - (4) Criteriile de intrare în test.
  - (5) Criteriile de ieșire din test.
  - (6) Instrumentele de testare.

---

## **SECTIUNEA 4**

### **PLANUL DE TESTARE**

#### **(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

---

#### **4.1 OBIECTIV**

*Obiectivul acestei secțiuni este acela de a defini instrumentele, procedurile și responsabilitățile pentru efectuarea tuturor nivelurilor de testare a sistemului de programe.*

## **4.2 DISCUȚIE**

*O modalitate convenabilă de a scrie un plan de testare este aceea de a defini fiecare nivel discret de testare (de exemplu, test modul, test de integrare, test de sistem, test de acceptanță, test de site) și apoi de a descrie obiectivele, procedurile, responsabilitățile și instrumentele pentru fiecare nivel. În această subsecțiune, se definesc pe scurt fiecare nivel de testare și precizează modul în care diferențele niveluri se potrivesc într-o ierarhie semnificativă de test. Trebuie subliniată necesitatea de modularizare a procesului de testare și nevoia de certitudine la un nivel înainte de a trece la următorul.*

## **4.3 DETALII**

### **4.3.1 Testarea modul**

*Testarea modul este testarea efectuată pe modulele de program de la cel mai de jos nivel înainte ca acestea să fie integrate cu alte module.*

**4.3.1.1 Obiectivele testului modulului**

**4.3.1.2 Responsabilitatea testării modulului**

**4.3.1.3 Proceduri de testare a modulului**

**4.3.1.4 Criterii de intrare în testul modulului**

**4.3.1.5 Criterii de ieșire a testului modulului**

**4.3.1.6 Instrumente de testare a modulelor**

### **4.3.2 Testarea de integrare**

*Testul de integrare este procesul de combinare a modulelor testate în grupări din ce în ce mai complexe, fie de sus în jos, fie de jos în sus și de testare a acestor grupări până când întregul sistem de programe a fost pus împreună și testat.*

**4.3.2.1 Obiectivele testului de integrare**

**4.3.2.2 Responsabilitatea testului de integrare**

**4.3.2.3 Proceduri de testare a integrării**

**4.3.2.4 Criterii de intrare în testul de integrare**

**4.3.2.5 Criterii de ieșire a testului de integrare**

**4.3.2.6 Instrumente de testare a integrării**

### **4.3.3 Testarea sistem**

*Testarea sistem este re-testarea sistemului de programe finalizat într-un mediu cât mai aproape posibil de cel real de funcționare, de către alte persoane decât cele care au realizat programele.*

- 4.3.3.1 Obiectivele testării sistemului**
- 4.3.3.2 Responsabilitatea testării sistemului**
- 4.3.3.3 Proceduri de testare a sistemului**
- 4.3.3.4 Criterii de intrare în testul sistemului**
- 4.3.3.5 Criterii de ieșire a testării sistemului**
- 4.3.3.6 Instrumente de testare a sistemului**

### **4.3.4 Testarea de acceptanță**

*Testul de acceptanță este exersarea sistemului de programe în condițiile agreate de client pentru a demonstra că sistemul satisface cerințele clientului.*

- 4.3.4.1 Obiectivele testului de acceptare**
- 4.3.4.2 Responsabilitatea testului de acceptare**
- 4.3.4.3 Proceduri de testare de acceptare**
- 4.3.4.4 Criterii de intrare în testul de acceptare**
- 4.3.4.5 Criterii de ieșire la testul de acceptare**
- 4.3.4.6 Instrumente de testare de acceptare**

### **4.3.5 Testarea de site (amplasament)**

*Testarea site-ului este testarea sistemului de programe în mediul său de operare final pentru a asigura că este gata de funcționare.*

- 4.3.5.1 Obiectivele testului de site**
- 4.3.5.2 Responsabilitatea de testare de site**
- 4.3.5.3 Proceduri de testare de site**
- 4.3.5.4 Criterii de intrare în testul de site**
- 4.3.5.5 Criterii de ieșire a testului de site**
- 4.3.5.6 Instrumente de testare a site-ului**

#### **4.3.6 Facilități comune de testare**

*Se descriu facilitățile și instrumentele comune utilizate pentru mai multe sau pentru toate nivelurile de testare.*

4.3.6.1 Biblioteca de sprijin pentru dezvoltare

4.3.6.2 Facilități informaticе

4.3.6.3 Servicii de tastare

4.3.6.4 Sisteme terminale

4.3.6.5 Limbaje speciale

4.3.6.6 Zonele de preluare și lansare de testare (Test Run Pickup and Drop Areas)

#### **4.3.7 Testarea programelor de suport**

*Se descriu elementele specifice referitoare la testarea instrumentelor de testare în sine.*

**Fig. 7.3.4.a. Model de Plan de testare**

### **7.3.5 Planul de control al modificărilor**

- Controlul modificărilor în sistemul de programe în curs de dezvoltare este una dintre cele mai vitale funcții ale managementului.
- Această secțiune definește:
  - (1) **Tipurile de modificări** care trebuie controlate.
  - (2) **Mecanismul** de efectuare a controlului respectiv. (fig. 7.3.5.a.).

## **SECTIUNEA 5**

### **PLANUL DE CONTROL AL MODIFICĂRILOR (MODEL)**

**DEPARTMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

## **5.1 OBIECTIV**

*Obiectivul acestei secțiuni este acela de a defini procedurile care vor fi utilizate pentru controlul modificărilor în sistemul de programe.*

## **5.2 DISCUȚIE**

*Se pornește de la nevoia clientului de a ști că ceea ce este dezvoltat de dezvoltator este ceea ce a fost imaginat atunci când a fost semnat contractul și nevoia dezvoltatorului de a ști că ceea ce produc programatorii este ceea ce a fost intenționat inițial. O soluție la această problemă este să se stabilească o anumită documentație de bază critică acceptabilă atât pentru client, cât și pentru dezvoltator și să se controleze evenimentele întotdeauna în raport cu acele documente de bază. Ori de câte ori se ridică o întrebare, documentele de bază sunt punct de referință. Dacă cineva dorește ceva ce nu este inclus în liniile de bază, aceasta este o modificare și trebuie negociată. Atunci când o modificare este considerată necesară, se estimează costul și impactul acesteia (dacă există vreunul), iar modificarea trebuie să fie consemnată în documentul (documentele) de bază. Un document de bază revizuit devine noua linie de referință.*

## **5.3 DETALII**

### **5.3.1 Documentele de referință**

*Se definesc documentele care vor fi utilizate ca bază pentru proiect.*

5.3.1.1. Specificația problemei

5.3.1.2. Specificația de design

### **5.3.2 Propunerea unei modificări**

5.3.2.1 Cine poate propune o schimbare

- (a) Membrii proiectului
- (b) Clientul
- (c) Alți contractorii

5.3.2.2 Documentul de propunere de modificare

### **5.3.3 Investigarea unei modificări propuse**

### **5.3.3.1 Cine, cum, când?**

### **5.3.3.2 Raportul anchetatorului**

- (a) Rezumatul modificării propuse
- (b) Numele inițiatorului și organizația
- (c) Clasificarea modificării
- (d) Impactul asupra costurilor, planificării și altor programe
- (e) Recomandări

### **5.3.4 Tipuri de modificări**

#### **5.3.4.1 Tipul 1**

*Modificarea afectează o linie de referință sau ar cauza un cost, o planificare sau alt impact.*

#### **5.3.4.2 Tipul 2**

*Modificarea nu afectează niciun document de referință și are un cost, o planificare sau alt impact toate neglijabile.*

### **5.3.5 Bordul de control al modificărilor**

- 5.3.5.1 Membri**
- 5.3.5.2 Când se întâlnește**
- 5.3.5.3 Cum funcționează**

### **5.3.6 Tipuri de recomandări**

- 5.3.6.1 Acceptare**
- 5.3.6.2 Respingere**

### **5.3.7 Implementarea unei modificări**

- 5.3.7.1 Estimarea costului modificării**
- 5.3.7.2 Aprobări**

- (a) Managementul proiectului
- (b) Client

#### **5.3.7.3 Documentarea modificării**

#### **5.3.7.4 Testarea modificării**

*OBS. Ca rezultat al Planului de control al modificărilor, o Procedură de modificare asociată trebuie definită ca o componentă a asigurării calității software (Software Quality Assurance - SQA).*

**Fig. 7.3.5.a. Model de Plan al controlului modificărilor**

- De fapt, **Planul de control al modificărilor** definește:
  - (1) **Documentele de bază** (documentele afectate de modificări).
  - (2) **Procedura de gestionare** a modificărilor constând în:
    - (2.1) Propunerea unei modificări.
    - (2.2) Investigarea unei modificări propuse.
    - (2.3) Tipuri de modificări.
    - (2.4) Bordul de control al modificărilor.
    - (2.5) Tipuri de recomandări.
    - (2.6) Implementarea unei schimbări.
- Când se scrie o procedură de control al schimbărilor, există întotdeauna tentația de a se încerca să se acopere orice fel de schimbare imaginabilă, indiferent cât este de minoră.
  - Aici pândește un mare pericol, deoarece aceste proceduri pot deveni rapid atât de încurate în detaliu încât devin o spaimă administrativă și se prăbușesc de la sine.

### **7.3.6 Planul de documentare**

- Aceasta este o secțiune cheie a planului proiectului, care de obicei lipsește.
- Intenția sa este de a controla șuvoiul de hârtii care însoțește inevitabil majoritatea proiectelor.
- O cauză importantă pentru se ajunge atât de des să fim îngropăți în hârtii este că nu ne facem timp să definim documentele pe care vrem să le folosim în proiect.
- Ca urmare, ori de câte ori un membru al proiectului trebuie să scrie ceva, el își crează propriul format și dintr-o dată apare un nou tip de document de ținut în evidență și de arhivat.
- Planul de documentare este locul în care se centralizează descrierile tuturor documentelor care urmează să fie utilizate în cadrul proiectului. (fig.7.3.6.a).

## **PLANUL DE DOCUMENTARE**

**(MODEL)**

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

### **6.1 OBIECTIV**

*Obiectivul acestei secțiuni este de a defini:*

- Numele și tipul documentelor din proiect
- Structura organizatorică a documentelor proiectului
- Procedura de publicare

### **6.2 DISCUȚIE**

#### **6.2.1 Generalități**

*Documentele standard recomandate pentru dezvoltarea unui proiect sunt prezentate în Rezumatul documentației fig. 7.3.6.b. În funcție de proiect, (dimensiune, importanță, preț), pot fi folosite doar câteva dintre documentele recomandate. De scos în evidență faptul că toate documentele utilizabile în proiect sunt precizate în această secțiune și nu se acceptă noi tipuri de documente. Dacă se propune un nou document, trebuie justificată inadecvanța documentelor existente; în acest caz, un nou document va fi generat și adăugat la plan*

#### **6.2.2 Denumirea și tipul documentelor de proiect**

*Pentru fiecare document al proiectului se descrie numele, conținutul, procedura de pregătire (cine îl scrie, data emiterii), procedura de aprobare (persoană), termenul limită, locația fizică a documentului. Totodată, pentru fiecare document trebuie specificate drepturile de acces (citire (R), citire-scriere (RW), acces parolat) pentru diferitele categorii de persoane. Se include o diagramă precum cea afișată în Rezumatul documentației și se complează cu detalii specifice jobului.*

#### **6.2.3 Numărul unui document**

*Fiecare document primește un număr unic care este înregistrat în Indexul documentației. Acest număr apare în toate locurile în care documentul este marcat.*

### **6.3. DETALII**

#### **6.3.1 Infrastructura documentelor de proiect (Project Documents Infrastructure PDI)**

*Această secțiune descrie filosofia Infrastructurii Documentelor de Proiect (PDI) pentru proiectul curent. PDI constă într-un număr de Arhive de Proiect care conțin documente, tabele de documente și alte informații asociate proiectului, structurate într-o manieră accesibilă. PDI este de fapt o colecție de date structurate care conține toate informațiile legate de proiect (documente, cod, rapoarte etc.). Scopul principal al acestei secțiuni este acela de a descrie această structură. PDI-ul este de obicei distribuit în rețeaua locală. Structura recomandată a PDI constă în următoarele depozite.*

- (1) Index de documentație
- (2) Depozitul planurilor de proiect
- (3) Depozitul de specificații de proiect
- (4) Depozitul de coduri al proiectului
- (5) Depozitul de teste de proiect
- (6) Depozitul de erori de proiect
- (7) Depozitul de modificări de proiect
- (8) Depozitul de documentare
- (9) Depozitul managerului

*În funcție de proiect, (dimensiune, importanță, preț), configurația depozitelor poate fi restructurată. Se include aici și un rezumat PDI care descrie depozitele de proiect și locația lor fizică, așa cum se arată mai jos.*

Nr.crt	Depozit	Locație fizică	Obs
1	Index de documente		
2	Depozitul planurilor de proiect	C:\Prj1\...	
3	Depozitul specificațiilor de proiect	D:\Users\...	
4	Depozitul de coduri al proiectului		
5	Depozitul de teste al proiectului		

6	Depozitul de erori (bugs) al proiectului		
7	Depozitul de modificări al proiectului		
8	Depozitul de documentare		
9	Depozitul managerului		

### 6.3.2 Descrierea depozitelor

#### 6.3.2.1 Indexul documentației

Un fișier care conține o listă a tuturor documentelor proiectului. Este dovada principală a oricărui document de proiect. Pentru orice documente de proiect, în afară de depozitul în care este plasat fizic, există o intrare în acest tabel. Fiecare document are un număr unic de identificare și un nume..

#### 6.3.2.2 Depozitul planului de proiect

##### 6.3.2.2.1 Conținutul depozitului

O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel de depozit al planului de proiect.

##### 6.3.2.2.2 Drepturi de acces

Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.

#### 6.3.2.3 Depozitul de specificații de proiect

##### 6.3.2.3.1 Conținutul depozitului

O descriere a structurii depozitului și o listă de fișiere incluse. Se include un tabel cu conținutul depozitului ca Tabel de depozit cu specificații de proiect.

##### 6.3.2.3.2 Drepturi de acces

Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.

#### 6.3.2.4 Depozitul de coduri al proiectului

##### 6.3.2.4.1 Conținutul depozitului

*O descriere a structurii depozitului și o listă de fișiere incluse. Se include un tabel cu conținutul depozitului ca Tabelul depozitului de coduri. Depozitul de coduri de proiect poate fi implementat folosind instrumente specifice (CVS).*

#### **6.3.2.4.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

#### **6.3.2.5 Depozitul de teste al proiectului**

##### **6.3.2.5.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se includeți un tabel cu conținutul depozitului ca tabel al depozitului de teste de proiect.*

#### **6.3.2.5.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

#### **6.3.2.5.3 Depozitul de teste de integrare**

##### **6.3.2.5.3.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel de conținut al depozitului ca tabel al depozitului de testare de integrare.*

#### **6.3.2.5.3.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

#### **6.3.2.5.4 Depozitul de testare a sistemului**

##### **6.3.2.5.4.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel al depozitului de testare a sistemului.*

#### **6.3.2.5.4.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

#### **6.3.2.5.5 Depozitul de testare de acceptanță**

##### **6.3.2.5.5.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel al depozitului de test de acceptare.*

#### **6.3.2.5.5.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

**6.3.2.5.6 Depozitul de teste ale site-ului**

**6.3.2.5.6.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel al depozitului de testare a site-ului.*

**6.3.2.5.6.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

**6.3.2.6 Depozitul de erori (bugs) de proiect**

**6.3.2.6.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă de fișiere incluse. Se include un tabel cu conținutul depozitului ca tabela de erori a proiectului.*

**6.3.2.6.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

**6.3.2.7 Depozitul de modificări de proiect**

**6.3.2.7.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel de modificare a proiectului.*

**6.3.2.7.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

**6.3.2.8 Depozitul documentației produsului software**

**6.3.2.8.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel de depozit al documentației produsului SW.*

**6.3.2.8.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

**6.3.2.9 Depozitul managerului de proiect**

**6.3.2.9.1 Conținutul depozitului**

*O descriere a structurii depozitului și o listă a fișierelor incluse. Se include un tabel cu conținutul depozitului ca tabel al depozitului managerului.*

#### **6.3.2.9.2 Drepturi de acces**

*Pentru fiecare director și fișier inclus în depozit se specifică drepturile de acces pentru diferitele categorii de persoane.*

### **6.3.3 Proceduri de publicare**

#### **6.3.1.1 Pregătirea și aprobarea**

#### **6.3.1.2 Verificare și editare**

#### **6.3.1.3 Vizibilitate**

*(a) Generală*

*(b) Restricționată*

#### **6.3.1.4 Reproducere**

*(a) Obișnuită*

*(b) Specială*

#### **6.3.1.5 Distribuție**

*(a) În cadrul proiectului*

*(b) Clientilor*

*(c) Alți contractorî*

*(d) Conducerea companiei*

*(e) Stocarea înregistrărilor vitale*

**Fig. 7.3.6.a. Model de Plan de Documentare**

- Când cineva dorește să noteze ceva, ar trebui să fie capabil să găsească un tip adecvat de document clar conturat în plan.
- Dacă s-a ratat ceva din planificarea inițială, nu este motiv de îngrijorare. Se pot adauga noi descrieri de documente ori de câte ori se consideră că sunt necesare.
- Descrierile documentelor trebuie să fie cât mai clare și cât mai flexibile posibil, astfel încât cei care le scriu să aibă libertatea de a se exprima.
- Deoarece mândria de autor este o forță de motivare foarte puternică în majoritatea oamenilor, indiferent dacă o recunosc sau nu, ghidurile de documentare care sunt prea restrictive vor fi ignorate.
- Pe lângă faptul că servește ca index al descrierilor documentelor, Planul de documentare include un rezumat al procedurilor de publicare care se ocupă de pregătirea, aprobarea, reproducerea, distribuirea și depunerea.

Name of Document	Contents	Preparation			Approval		Repository Location
		Who writes	When finished	Formal pro of	Who approves	Dea d line	
<a href="#"><u>PROJECT PLAN</u></a>	The plans associated to the project	Project Manager	End of Definition Phase	Yes			<a href="#"><u>Project Plan Repository</u></a>
<a href="#"><u>PROBLEM SPECIFICATION</u></a>	A description of the requirements of the problem to be fulfilled by the product	Analysts	End of Definition Phase	Yes	Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>DESIGN SPECIFICATION</u></a>	A description of the design to be used by the programmers in producing the product	Analysis and Design Group	End of Design Phase	Yes	Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>CODING SPECIFICATION</u></a>	A detailed description of the modules produced by the programmers; the specifications describe the complete product	Individual programmers	Preliminary version at end of module test Final version at end of acceptance test	Yes	Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>CHANGE PROPOSAL</u></a>	A description of proposed change to the Problem Specification and/or Design Specification	Anyone	Anytime	No			<a href="#"><u>Project Change Repository</u></a>
<a href="#"><u>CHANGE RECOMMENDATION</u></a>	A description of the proposed change, classification of the change, the impact, adoption recommendation	Change Investigator	After a Change Proposal	Yes	Change Control Board		<a href="#"><u>Project Change Repository</u></a>

<a href="#"><u>PROBLEM SPECIFICATION CHANGE NOTICE</u></a>	A description of adopted changes to the current Problem Specification	Contractor, Analysis and Design Group	Anytime	No	Contractor Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>DESIGN SPECIFICATION CHANGE NOTICE</u></a>	A description of adopted changes to the current Design Specification	Contractor, Analysis and Design Group	Anytime	No	Depends on change		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>INTEGRATION TEST SPECIFICATION</u></a>	A description of the philosophy, objectives and procedures involved in integration test; a matrix of test cases	Programmers	End of Design Phase	No	Contractor		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>SYSTEM TEST SPECIFICATION</u></a>	A description of the philosophy, objectives and procedures involved in system test; a matrix of test cases	Test Group	End of Programming Phase	Yes	Contractor		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>ACCEPTANCE TEST SPECIFICATION</u></a>	A description of the philosophy, objectives and procedures involved in acceptance test; acceptance criteria; a matrix of test cases	Test Group	Preliminary version at end of Definition Phase; Final version at end of Programming Phase	Yes	Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>SITE TEST SPECIFICATION</u></a>	A description of the philosophy, objectives and procedures involved in site test; a matrix of test cases	Test Group	End of Programming Phase	Yes	Customer		<a href="#"><u>Project Specification Repository</u></a>
<a href="#"><u>TEST CASE</u></a>	Individual test script and data	Test Group and Programmers	Depends on type of test	No	Depends on type of test		<a href="#"><u>Specific Test Repository</u></a>

<u>TEST REPORT</u>	A report of any problem encountered during any formal test; integration, system, acceptance	Test conductor	After running any test cases	No			Specific <a href="#">Test Repository</a>
<u>TEST SCRIPT</u>	A description of the steps to be followed in test. (Part of the Test Case)	Test Group/ Programmers	Depends on type of test	No	Depends on the type of test		Specific <a href="#">Test Repository</a>
<u>TEST CHECKLIST</u>	A description of the items to be verified. (Part of the Test Case)	Test Group (Programmers )	Depends on type o test	No	Depends on type of test		Specific <a href="#">Test Repository</a>
<u>BUG DESCRIPTION</u>	A bug description for each of the problem encountered during the any test. (Attached to the Test Report).	Tester	After running any test cases				<a href="#">Project Bugs Repository</a>
<u>BUG FIXING NOTE</u>	A description of the bug fixing.	Debugging programmer	After debugging				<a href="#">Project Bugs Repository</a>
<u>TECHNICAL NOTE</u>	Miscellaneous technical correspondence	Anyone	Anytime	No	Depends on the content		<a href="#">Manager Repository</a>
<u>ADMINISTRATIVE NOTE</u>	Miscellaneous administrative correspondence	Anyone	Anytime	No	Depends on the content		<a href="#">Manager Repository</a>
<u>PROGRAMMER'S HANDBOOK</u>	Collection of data needed by the programmer	Technical Staff	First version at the end of Design Phase	No	Contractor		<a href="#">Code Repository</a>
<u>TECHNICAL STATUS REPORT</u>	A single form used in reporting technical status to the next management level within the contractor's organization	Each level	Biweekly or monthly	No			<a href="#">Manager Repository</a>

<u>PROJECT HISTORY</u>	A set of charts showing <ul style="list-style-type: none"> <li>• Significant events</li> <li>• Manpower (estimated/actual)</li> <li>• Machine time (estimated/actual)</li> </ul>	Administrative Staff (Project manager)	At the end of the contract	No	Contractor		<a href="#">Manager Repository</a>
<u>DOCUMENTATION INDEX</u>	A table containing entrances for all current project documents	Administrative Staff (Librarian)	Periodically published	No			<a href="#">Project Documents Infrastructure</a>

**Fig. 7.3.6.b. Documentation Summary**

### 7.3.7 Planul de instruire

- În general, există două categorii de instruire necesare pentru un proiect:
  - (1) **Internă** - antrenarea propriilor oameni.
  - (2) **Externă** - instruirea clientului și a altora.
- De regulă instruirii (formării) îi este adesea acordată puțin sau deloc spațiu într-un plan, însă această omisiune poate fi deosebit de gravă pentru unele joburi.
- Planul de instruire definește toate tipurile de pregătire internă și externă necesare, responsabilitatea pentru fiecare și resursele necesare (fig. 7.3.7.a.).

---

## **SECȚIUNEA 7**

### **PLANUL DE INSTRUIRE**

**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

## **7.1. OBIECTIV**

*Obiectivul acestei secțiuni este acela de a defini responsabilitățile de instruire ale contractantului.*

## **7.2. DISCUȚIE**

*Contractorul de programare este responsabil pentru două categorii generale de instruire: internă (instruirea personalului său intern) și externă (instruirea clientului, contractorului de sistem și altele).*

## **7.3. DETALII**

### **7.3.1. Tipuri de instruire (training)**

#### **7.3.1.1. Instruire internă**

(a) Înțelegerea întregului proiect

(b) Aspecți tehnice

- Limbajul de codare
- Mediul de dezvoltare
- Utilizarea instrumentelor de testare
- Procesarea datelor
- Interfața cu alte sisteme
- Problema de soluționat
- Proiectarea de bază

(c) Aspecte nontehnice

- Tehnici de management
- Procedurilor de control a modificărilor
- Controlul documentației
- Cerințe de raportare
- Proceduri de birou

#### **7.3.1.2. Instruire externă**

(a) Instalarea programului

(b) Utilizarea sistemului

(c) Modificarea sistemului

### **7.3.2. Resurse**

*Pentru fiecare tip de instruire identificat se precizează:*

- Planificările de instruire
- Necessarul de instructori
- Materiale de instruire
- Facilități (săli de clasă, calculatoare etc.)
- Număr de cursanți
- Programe speciale de calculator pentru instruire

**Fig. 7.3.7.a.** Model de Plan de instruire

### **7.3.8 Planul de revizii și raportări**

- Obiectivul acestei secțiuni de plan este de a defini modul în care starea proiectului va fi comunicată prin:
  - Evaluări orale ale proiectului.
  - Rapoarte scrise.
  - Parcurgeri structurate.
  - Inspectii.
- Parcurgerile structurate și inspectiile sunt discutate în detaliu mai târziu.
  - Nu sunt concepute ca un mijloc de raportare a stării către management, dar sunt un mijloc important de a ajuta membrii proiectului să evalueze calitatea produselor pe care le dezvoltă.
- Schema Planului de revizuire și raportare este prezentată în fig. 7.3.8.a.

## **SECȚIUNEA 8**

### **PLANUL DE REVIZII ȘI RAPORTĂRI**

**(MODEL)**

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

## **8.1. OBIECTIV**

*Obiectivul acestei secțiuni este de a descrie mijloacele de revizuire și raportare a progresului.*

## **8.2. DISCUȚIE**

*În cadrul oricărui proiect există un proces de revizuire informală și de raportare mai mult sau mai puțin continuu, la toate nivelurile. Acest plan nu se referă la revizuirea și raportarea informală, ci mai degrabă la aspectele formale ale acestora. Această subsecțiune Discuție trebuie să descrie la modul general structura de raportare. Trebuie în continuare să sublinieze importanța realizării rapoartelor financiare, a celor tehnice și a coeranței dintre ele. Trebuie apoi să descrie sistemul de contabilitate al contractantului la care trebuie să se conformeze rapoartele financiare ale proiectului.*

## **8.3. DETALII**

### **8.3.1. Recenzii**

#### **8.3.1.1. Recenzii interne**

*Activitățile de revizuire interne se vor baza pe modelul de dezvoltare prezentat în Procedura de revizuire. Participanții la fiecare evaluare internă includ membri ai proiectului și evaluatori externi.*

##### **(a) Analiza fazei de definiție**

Când: *Sfârșitul fazei de definire*

Obiective:

- *Să revizuiască specificația problemei și să determine gradul de pregătire pentru faza de proiectare;*
- *Să revizuiască și să evalueze Planul de Proiect;*
- *Să revizuiască criteriile de acceptanță.*

##### **(b) Revizuirea fazei preliminare de proiectare**

Când: *La jumătatea fazei de proiectare*

Obiective:

- *Să revizuiască proiectul de bază, în măsura în care acesta a fost elaborat, pentru a asigura validarea abordării de proiectare.*

##### **(c) Revizuirea fazei de proiectare**

Când: *Sfârșitul fazei de proiectare*

Obiective:

- Să revizuiască specificația de proiectare completată pentru a determina dacă îndeplinește sau nu specificația problemei și este rezonabilă și programabilă;

- Să revizuiască Planul de Proiect. (Se includ recenzenți externi)

(d) Revizuirea fazei de programare

Când: *Sfârșitul fazei de programare*

Obiective:

- Să revizuiască rezultatele integrării programului și să determine gradul de pregătire pentru faza de testare a sistemului;

- Revizuirea documentației programului

(e) Revizuirea fazei de testare a sistemului

Când: *Sfârșitul fazei de testare a sistemului*

Obiective:

- Să revizuiască rezultatele testelor sistemului și să determine gradul de pregătire pentru Faza de Acceptare;

- Să revizuiască documentația programului

(f) Revizuirea finală (post mortem)

Când: *Sfârșitul fazei de acceptare*

Obiective:

- Să revizuiască și să aprobe documentul istoria proiectului.

### 8.3.1.2. Recenzii externe

Participanții la fiecare dintre aceste recenzii includ reprezentanți ai contractorului și ai clientului.

(a) Revizuirea preliminară a proiectului

Când: *La jumătatea fazei de proiectare, după revizuirea internă.*

Obiective:

- Să revizuiască validitatea abordării de proiectare.

(b) Revizuirea fazei de proiectare

Când: *La sfârșitul fazei de proiectare, după revizuire internă.*

Obiective:

- Să revizuiască în detaliu Specificația de proiectare;

- Să revizuiască Planul de proiect al contractorului pentru intrarea în faza de programare.

### (c) Examinarea acceptanței

Când: sfârșitul fazei de acceptanță

Obiective:

- Să revizuiască rezultatele testelor de acceptanță terminate și să determine orice probleme rămase care trebuie corectate înainte ca clientul să accepte oficial programele.

### 8.3.1.3. Parcurgeri structurate

Acestea sunt ținute ori de câte ori există un produs (un design, un cod, un plan de testare, un manual de utilizare, orice) pregătit pentru o scrutare atentă de către alți membri ai proiectului, ghidați („conduși”) de dezvoltatorul produsului.

Obiective:

- Pentru a găsi erori și nu pentru a raporta starea.

## 8.3.2. Rapoarte

### 8.3.2.1. Generate de nonmanageri

- (a) Frecvență: la două săptămâni.
- (b) Către: managerul imediat.
- (c) Format: Raport de stare tehnică.
- (d) Domeniul de aplicare: un raport pentru fiecare sarcină atribuită.

### 8.3.2.2. Generate de manageri

- (a) Frecvență: bi-săptămânal.
- (b) Către: managerul imediat;
- (c) Format: Raport de stare tehnică.
- (d) Domeniul de aplicare: un raport pentru fiecare sarcină de reper.

### 8.3.2.3. Generat de managerul de proiect

- (a) Frecvență: lunar și trimestrial. Un raport trimestrial poate să înlocuiască raportul normal lunar.
- (b) Către: conducerea companiei și clienți.
- (c) Format: Depinde de cerințele companiei și ale clienților.

### 8.3.2.4. Generat de personalul companiei

Descrie feedbackul către managementul companiei. Aceste rapoarte sunt de obicei financiare. Vezi Raportul personalului.

**Fig. 7.3.8.a. Model de Plan de revizii și raportări**

- Planul de revizuire și raportare descrie următoarele activități:

- Evaluări interne.
- Recenzii externe.
- Parcurgeri structurate.
- Rapoarte.
- Se recomandă a se evita transformarea recenziilor proiectelor în spectacole.
  - Pentru ca ele să fie utile, trebuie evitată tentația de a trâmbița succesele și de a acoperi problemele - o sarcină dificilă, indiferent cât de obiectivi și de sinceri se cred că sunt cei care o realizează.
- O modalitate de a ajuta la rezolvarea problemelor este aceea de a include evaluatori externi competenți, adică oameni care nu au nicio implicare personală în proiect.
- Recenziile vor discutate mai târziu în curs, dar în modelul prezentat se sugerează un set de posibile recenzi.
- Rapoartele de proiect scrise, ca și alte documente, au o tendință de a deveni din ce în ce mai voluminoase și din ce în ce mai puțin utile.
- **Planul** trebuie să prezinte **exact**:
  - Ce rapoarte sunt necesare.
  - Organizarea rapoartelor.
  - Frecvența rapoartelor.
  - Responsabilitatea pentru redactarea rapoartelor.
  - Distribuirea rapoartelor.
  - Relaționarea între ele a rapoartelor.

### 7.3.9 Planul de instalare și operare

- Acest plan descrie procedura de instalare și funcționare corectă a sistemului de programe terminat, „acceptat”, în mediul dorit, posibil într-un loc de apărare antirachetă, sau în centrul de calcul de pe hol.
- Modelul acestui plan este prezentat în fig. 7.3.9.a.
- Planul conține două capitole:
  - (1) **Instalare**.
  - (2) **Operare**.

- Chiar și cele mai simple programe pot ridica probleme, cum ar fi modul de conversie (trecere) de la un sistem existent, poate manual, la noul sistem computerizat.

---

## **SECȚIUNEA 9**

### **PLANUL DE INSTALARE ȘI OPERARE (MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

---

#### **9.1. OBIECTIV**

*Obiectivul acestei secțiuni este de a defini responsabilitățile contractantului în instalarea și operarea sistemului de programe acceptat.*

#### **9.2. DISCUȚIE**

*Volumul de participare a unui antreprenor la instalarea și operarea unui sistem pe care l-a livrat, diferă de la un proiect la altul. Subsecțiunea de discuții descrie gradul de implicare pentru proiectul în cauză.*

#### **9.3. DETALII**

##### **9.3.1. Instalare**

###### **9.3.1.1. Responsabilitate**

###### **9.3.1.2. Planificare**

###### **9.3.1.3. Conversie**

(a) Metoda utilizată

*Funcționare în paralel, înlocuire imediată etc.*

(b) Criterii pentru inaugurare

*Cum trebuie luată decizia de a întrerupe vechiul sistem și de a-l utiliza pe cel nou.*

(c) Cine ia decizia de inaugurare

(d) Măsuri de rezervă în cazul în care sistemul se defectează

#### **9.3.1.4. Introducerea datelor**

(a) Cine adună datele

(b) Cine validează datele

#### **9.3.1.5. Considerații privind mai multe site-uri**

(a) Echipa de instalare a site-ului

(b) Coordonarea inter site-uri

### **9.3.2. Operare**

#### **9.3.2.1. Responsabilități pentru exploatare**

#### **9.3.2.2. Responsabilități pentru întreținere și reglaj**

(a) Proceduri de control a modificărilor

(b) Locul de muncă

(c) Finanțarea

#### **9.3.2.3. Durata Responsabilităților**

**Fig. 7.3.9.a.** Model de Plan de instalare și operare

### **7.3.10 Planul de resurse și livrabile**

- Acum acest plan reunește la un singur loc detaliile critice asociate planului:
  - (1) **Forța de muncă.**
  - (2) **Ore calculator.**
  - (3) **Alte resurse.**
  - (4) Planificări de livrare - Un rezumat al tuturor articolelor care trebuie fie livrate conform contractului.
  - (5) **Diagrama de repere** (milestones) - Un rezumat al reperelor proiectului.
  - (6) **Bugetul.**
- Modelul acestui plan este prezentat în fig. 7.3.10.a.

- Aceste date sunt printre cele mai frecvent modificate sau consultate, aşa că trebuie adunate într-un singur loc pentru a le face mai ușor de găsit și mai ușor de schimbat.

---

## SECTIUNEA 10

### PLANUL DE RESURSE ȘI LIVRABILE (MODEL)

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

---

#### **10.1. OBIECTIV**

*Obiectivul acestei secțiuni este de a aduna într-un singur loc un rezumat al tuturor estimărilor de resurse și un program pentru toate livrabilele.*

#### **10.2. DISCUȚIE**

*Diverse resurse, grafice și elemente livrabile sunt menționate sau implicate și în alte secțiuni ale Planului de proiect. Aici sunt toate adunate împreună și făcute explicite.*

#### **10.3. DETALII**

##### **10.3.1. Forța de muncă**

- *O diagramă generală care arată forța de muncă totală planificată pentru proiect pe o bază lunară.*
- *O diagramă principală care prezintă două categorii mari: forță de muncă de programare și non-programare. În prima sunt incluși programatorii și managerii de primul nivel; în a doua sunt incluse toate celelalte feluri de forță de muncă.*
- *Diagramele suport ar trebui să detalieze cele două categorii.*

- Dacă proiectul este mare și dacă există un număr de subsisteme majore de program, se prezintă diagrame separate de forță de muncă pentru fiecare subsistem.
- Dacă proiectul planifică mai multe versiuni ale sistemului de programe, se evidențiază forță de muncă pentru fiecare versiune separat.

### 10.3.2. Ore calculator

- Se prezintă cerințele lunare de timp de computer, defalcate în funcție de versiunea programului, de subsistemul major al programului de versiune (release) și de categoria de utilizare: test de modul/integrare, test de sistem, test de acceptare, test de site.
- Dacă se utilizează mai multe tipuri de computere la instalare (platforme diferite), se realizează estimări separate pentru fiecare.
- Se evidențiază separat timpul computer pentru alte categorii, cum ar fi utilizări administrative.

### 10.3.3. Alte resurse

#### 10.3.3.1. Costuri de publicare

- (a) Rapoarte
- (b) Specificația problemei
- (c) Specificația de proiectare
- (d) Specificații de codificare
- (e) Documente utilizator
- (f) Documente de testare

#### 10.3.3.2. Cheltuieli de călătorie

- (a) Spre locațiile proprii ale contractantului
- (b) Către locațiile clienților
- (c) Spre locațiile altui contractant
- (d) Pentru a testa site-urile

#### 10.3.3.3. Relocarea angajaților și a echipamentelor

#### 10.3.3.4. Echipamente și consumabile

Echipamentele și consumabilele normale de birou plus orice obiecte speciale (dischete suplimentare, hard disk-uri, cd-uri etc.).

#### 10.3.3.5. Achiziții sau închirieri speciale

Obiecte precum spațiu suplimentar pentru birouri sau spații temporare în remorci.

### 10.3.4. Planificări de livrare

- *Grafic(e) care arată datele și livrabilele solicitate în contract sau în orice acord ulterior.*
- *Diagrama trebuie să fie însorită de un set de descrieri narrative capsulare pentru fiecare element afișat pe diagramă.*

#### **10.3.5. Diagrama de repere**

- *O diagramă care arată toate reperele pe baza cărora trebuesc făcute rapoarte către client. O bază bună pentru această diagramă o reprezintă o variație a modelului ciclului de viață. Este util să se afișeze reperele suprapuse pe un ciclu de dezvoltare, astfel încât să se poată relaționa mai bine fiecare etapă de activitățile majore planificate, adică de faze.*
- *Se include o fișă separată care oferă o descriere capsulară pentru fiecare etapă indicată pe diagramă. Consultați Ghidul de repere ale proiectului.*

#### **10.3.6. Buget**

*O copie a bugetului financiar care arată modul în care fondurile sunt alocate fiecărei categorii de costuri prezentate în secțiunile precedente. Pe măsură ce estimările sunt reconsiderate și modificate, bugetul trebuie să se schimbe. Când se întâmplă acest lucru, această subsecțiune trebuie actualizată pentru a reflecta toate modificările.*

---

**Fig. 7.3.10.a. Model de Plan de resurse și livrabile**

#### **7.3.11 Indexul planului**

- Este una dintre cele mai eficiente modalități de a face Planul de proiect mult mai atractiv și mai ușor de utilizat.

## **8 Redactarea criteriilor de acceptanță**

- Discuțiile referitoare la testarea de acceptanță vor fi incluse în faza de acceptare din capitolul 8.
- Însă munca propriu-zisă de pregătire pentru acceptare începe aici, în Faza de definire.
- Ceea ce merită evidențiat în mod cu totul deosebit acum este faptul că aceste criterii de acceptanță trebuie **convenite și scrise din timp**.

- Nu se recomandă să se lucreze la un întreg proiect fără a ști exact ce condiții trebuie să îndeplinească produsul pentru a fi acceptabil pentru client.

### **Exercițiul #6**

1. Care sunt resursele unui proiect SW? Care sunt elementele de cost ale unui proiect SW?
2. Descrieți cele mai cunoscute tehnici de estimare a costurilor SW: descriere, avantaje, limitări.
3. Ce fel de modele de evaluare a costurilor SW bazate pe descompunere cunoașteți? Descrieți principalele lor caracteristici.
4. Ce este COCOMO 81? Explicați-i filozofia și modul de utilizare: niveluri, intrări, procesare, ieșiri, suport.
5. Care sunt îmbunătățirile introduse de COCOMO II? Descrieți principalele etape ale COCOMO II.
6. Descrieți principalele caracteristici ale modelelor parametrice de evaluare a costurilor PRICE S și SEER SEM.
7. Care sunt etapele estimării unui Proiect SW?
8. Ce este un plan de proiect? Care sunt caracteristicile unui plan bun?
9. Descrieți pașii activității de redactare a planului.
10. Care este structura unui plan de proiect SW? Descrieți la nivel detaliat conținutul Secțiunilor Planului.

## **Capitolul 5. FAZA DE PROIECTARE**

### **1. Proiectarea sistemului**

- 1.1 Specificația de proiectare
- 1.2 Designerii
- 1.3 Ambientul de proiectare
- 1.4 Ghid de proiectare
- 1.5 Instrumente de proiectare
  - 1.5.1 Scheme logice
  - 1.5.2 HIPO
  - 1.5.3 Pseudocod
  - 1.5.4 Diagrame structurate
  - 1.5.5 Diagrame de flux de date
  - 1.5.6 Tabele de decizie
  - 1.5.7 UML
  - 1.5.8 Matrici de acoperire
  - 1.5.9 Hărți de stocare
  - 1.5.10 Limbaje de programare
  - 1.5.11 Modele de simulare
- 1.6. Evaluarea calității designului

### **2. Planificarea proiectului în faza de proiectare**

- 2.1 Controlul modificărilor
- 2.2 Pregătirea pentru testare
  - 2.2.1 Definirea ierarhiei testelor
  - 2.2.2 Testarea integrării de sus în jos vs. de jos în sus
  - 2.2.3 Scrierea specificațiilor de testare
  - 2.2.4 Definirea procedurilor de testare
  - 2.2.5 Furnizarea timpului de calculator
  - 2.2.6 Trasarea rezultatelor testelor
- 2.3 Estimarea resurselor
- 2.4 Documentare
  - 2.4.1 Manualul de programare
  - 2.4.2 Biblioteca de proiect
- 2.5 Instruire

### **3. Revizia fazei de proiectare**

- 3.1 Pregătirea reviziei
  - 3.1.1 Programarea persoanelor
  - 3.1.2 Programarea sărilor de întâlnire
  - 3.1.3 Pregătirea suporturilor de prezentare
  - 3.1.4 Pregătirea materialelor necesare
- 3.2 Obiectivele reviziei fazei de proiectare
- 3.3 Rezultate

**Exercițiul #8**

## Capitolul 5. Faza de proiectare

- La acest moment , managementul devine mai dinamic.
- Au trecut săptămâni, poate luni, de când a fost semnat contractul și încă nu există niciun semn de programe (de fapt, puține semne din partea programatorilor).
- Este timpul ca managerul proiectului să demonstreze conducerii că totul sub control conform planificării:
  - (1) Analiza este completă.
  - (2) Planul proiectului.
  - (3) Echipa de proiectare a fost recrutată și lucrează din greu.
- Pe scurt, au fost respectate etapele de referință și se trece la următorul set de obiective:
  - (1) Proiectarea sistemului.
  - (2) Rafinarea Planului de Proiect.
  - (3) Efectuarea unei analize cuprinzătoare a întregului proiect înainte de a începe programarea.

### 1 Proiectarea sistemului

- O ieșire cheie a **fazei de definire** a fost **Specificația problemei**, care definește lucrarea de realizat.
- Următorul document important de scris este **Specificația de proiectare**.
  - Specificația de proiectare este modelul pentru sistemul de programe.
  - Este punctul de plecare pentru programatori.
- Este greu de subliniat suficient, importanța de a avea acest document și de a face din el **punctul focal** al **activității programatorilor**.
  - Este un joc ruinător jucat de manageri care nu înțeleg că trebuie să proiectezi înainte de a codifica.
  - Astfel de manageri confundă mișcarea cu progresul. (Larry Constantine)
- **Specificația de proiectare** stabilește **soluția** la **problema clientului**.
  - (1) Este o soluție aleasă de managementul proiectului dintre alternativele oferite de echipa de proiectare.
  - (2) Designul ales trebuie să fie „cel mai bun” pentru proiect.

- (3) Poate să nu fie cel mai bun din punct de vedere al eleganței, nici cel ales în situația în care ar fi disponibile resurse nelimitate, dar trebuie să fie cel mai bun care poate fi implementat, având în vedere constrângerile, referitoare la timpul, talentele, echipamentele și banii disponibili.
- (4) Indiferent de designul ales, acesta trebuie, desigur, să satisfacă complet cerințele menționate în **Specificația problemei**.

## 1.1 Specificația de proiectare

- **Specificația de proiectare** descrie o soluție acceptabilă de programare pentru problema menționată în **Specificația problemei**.
  - Specificația de proiectare este baza pentru toate proiectele detaliate și pentru toate codurile viitoare.
- O bună specificație de proiectare prezintă soluția în două moduri: în termeni de **funcții** și în termeni de **logică**.
  - (1) **Descrierea funcțională** arată **ce trebuie să facă sistemul**.
  - (2) **Descrierea logică** arată modul în care **sistemul este** de fapt **structurat** pentru a realiza acele funcții.
    - De exemplu: O **descriere funcțională** poate include o casetă care conține cuvintele „Calculează traectoria”, dar este lăsată la latitudinea descrierilor logice să descrie metoda sau procedurile care urmează să fie programate pentru a face calculele reale.
- Managementul și proiectanții trebuie să aleagă **instrumentele adevărate** pentru a le folosi în comunicarea atât a funcționalităților, cât și a logicii.
  - (1) O metodă este de a folosi **diagramele HIPO** pentru a descrie **funcțiile** și **schemele logice** (flow charts) pentru a descrie **logica**.
  - (2) O altă metodă, mai concordantă cu tendințele ce prefigurează programarea structurată, este folosirea **HIPO** pentru **funcții** și **diagrame structurate** sau **pseudocod** pentru **logică**.
  - (3) În ultima vreme, **diagramele UML** sunt din ce în ce mai folosite ca suport pentru proiectarea arhitecturii. Aceste instrumente sunt descrise mai târziu.
- **Documentația de proiectare** trebuie să fie rezultatul unor investigații și a unor decizii motivate.
  - Nu trebuie lăsat nimic să se întâmple din cauza inacțiunii.
  - Nu trebuie lăsat un designer să folosească o metodă și un altul o metodă diferită.

- Trebuie avut în vedere în fiecare moment la modul în care documentele de referință se vor interfața cu cele care le-au precedat (Specificația problemei) și cele ce le vor urma (descrierile detaliate ale programului). Atenție deosebită la coerenta proiectului.
- O tehnică precum **HIP**O, de exemplu, poate fi utilizată de analiști în scrierea Specificației problemei, ușurând astfel tranziția între acel document și Specificația de proiectare.
  - De fapt, **secțiunea funcțională a specificației de proiectare** poate arăta foarte asemănătoare cu **specificarea problemei**.
- **HIP**O poate fi, la rândul său, o parte majoră a documentației detaliate pentru programele individuale, extinzând astfel sentimentul de consistență și continuitate în documentația tehnică a proiectului.
- **Conținutul general** al **Specificației de proiectare** constă în (fig. 5.1.1.a):
  - (1) Conceptul general de proiectare
  - (2) Standarde și convenții
  - (3) Proiectarea programului
  - (4) Designul fișierelor
  - (5) Fluxul de date

---

## **SPECIFICATIA DE PROIECTARE**

**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA EMITERII:**

**REALIZAT DE:**

---

### **SECȚIUNEA 1: SCOP**

*Acest document definește o soluție la problema descrisă în Specificația problemei. Specificația de proiectare este baza pentru implementarea tuturor programelor. Logica de proiectare descrisă aici este suficient de detaliată astfel încât să fie satisfăcute toate cerințele funcționale și să fie definite toate interfețele, fișierele de sistem și logica care conectează*

*toate modulele de program. Proiectarea este realizată suficient de detaliat pentru ca toate problemele logice ale sistemului să fie rezolvate și întregul sistem de programe „să se stea împreună”. Cel mai de jos nivel, respectiv cel al modulului de program, este specificat în tremenii funcțiilor pe care trebuie să le îndeplinească și a interfețelor pe care trebuie să le aibă cu alte module, dar designul intern propriu-zis al acestor module de cel mai jos nivel este lăsat la latitudinea programatorilor care fac implementarea.*

*Dacă proiectul va produce mai mult de un sistem de programe, de exemplu, programe suport pe lângă programele operaționale, vor exista mai multe specificații de proiectare.*

## **SECȚIUNEA 2: DOCUMENTE APLICABILE**

## **SECȚIUNEA 3: CONCEPTUL GENERAL DE PROIECTARE**

*Aceasta este o prezentare generală a întregii ierarhii a sistemului de programe.*

### **3.1. Ierarhia programelor**

*Definirea și descrierea ierarhiei sistemului de programe.*

### **3.2. Ierarhia datelor**

*Definirea și descrierea fișierelor sistem și a interrelațiilor dintre ele, inclusiv imagini simple ale structurilor fișierelor.*

### **3.3. Standarde și convenții**

#### **3.3.1. Standarde și convenții de proiectare**

*Definirea tuturor standardelor și convențiilor adoptate pentru utilizare în acest document de proiectare și care urmează să fie respectate în timpul proiectării detaliate ulterioare.*

##### **3.3.1.1. Standarde UML**

##### **3.3.1.2. Standarde de denumire**

##### **3.3.1.3. Standarde de interfață**

##### **3.3.1.4. Formate de mesaje**

#### **3.3.2. Standarde și convenții de codificare**

*Definirea tuturor standardelor și convențiilor care trebuie respectate în timpul codificării.*

##### **3.3.2.1. Limbaje de programare**

- 3.3.2.2. Practici de codificare interzise
- 3.3.2.3. Practici de codificare obligatorii
- 3.3.2.4. Practici de codificare recomandate

## SECȚIUNEA 4: PROIECTAREA DE BAZĂ

Acesta este punctul central al acestui document. Toată logica programului și fișierelor de sistem este prezentată aici la nivelul de detaliu pe care proiectanții îl consideră necesar înainte de a preda documentul programatorilor pentru implementare.

### 4.1. Proiectarea programului

Printr-o combinație de diagrame, informații narrative și tabelare, această secțiune descrie sistemul programului mai întâi din punct de vedere funcțional. Apoi sistemul este descris în termeni de funcționalități, de structura sa actuală sau de logica.

### 4.2. Designul fișierelor

Prezentări ilustrate ale tuturor fișierelor sistem care descriu toate subdiviziunile fișierelor și a conținutului acestora. De asemenea, o descriere completă a relațiilor dintre diferitele fișiere, inclusiv indicatori utilizați pentru a lege fișiere și matricile de acoperire care arată ce programe accesează fiecare fișier.

## SECȚIUNEA 5: FLUXUL DE DATE

Această secțiune utilizează diagrame de flux și narrative însotitoare pentru a descrie tranzacțiile majore din sistem, indiferent de structura logică reală a sistemului. Intenția este de a oferi o înțelegere a căilor de date și a evenimentelor majore din sistemul operațional, inclusiv toate subsistemele, atât hardware cât și software. Această expunere este utilă ca o introducere în sistem care nu ar trebui să presupună cunoștințe de programare din partea cititorului.

---

**Fig. 5.1.1.a.** Model de Specificație de proiectare

- (1) Conceptul general de proiectare
  - Aceasta este o combinație scurtă de narătive și diagrame care oferă o imagine de ansamblu asupra întregului proiect al sistemului de programe la un nivel înalt.
- (2) Standarde și convenții

- Această secțiune precizează regulile adoptate pentru a fi utilizate în descrierea atât a designului de bază, cât și a designului detaliat care urmează să fie realizat ulterior de către programatori.
  - Ea acoperă articole precum:
    - Scheme logice.
    - Standarde HIPO.
    - Standarde UML.
    - Standarde de denumire.
    - Convenții de interfață.
    - Formate de mesaje.
  - Această secțiune include de asemenea:
    - Standarde și convenții de codare care trebuie respectate în faza de programare.
    - Practici de codare interzise, solicitate și recomandate.
    - Dacă astfel de standarde sunt deja publicate pentru organizație, o simplă referire la aceste standarde aici va fi suficientă.
- **(3) Proiectarea programului**
    - Acesta este **nucleul** documentului.
    - Printr-o combinație de diagrame, informații narative și tabelare, descrie sistemul programului mai întâi din punct de vedere funcțional, apoi în termeni de funcționalități, de structura sa reală sau de logica.
    - Începe cu o privire asupra ierarhiei generale și apoi împarte sistemul în bucăți mai mici pentru o privire mai atentă, pe baza WBS.
      - Nivelul de detaliu trebuie să fie astfel ales încât să nu fie lăsate probleme majore de proiectare în faza de programare.
      - Dar acest design de bază nu ar trebui să fie dus la un nivel foarte ridicat de detaliu. Există două motive pentru aceasta:
        - (1) În primul rând, proiectarea detaliată ar face din acesta un document masiv și va fi practic imposibil de aplicat un control eficient al modificărilor.
          - Controlul modificărilor trebuie să se concentreze pe structura sistemului la un nivel suficient de înalt.
          - Deci, o schimbare a manierei în care este codificat un modul de nivel scăzut nu este supusă controlului formal.
        - (2) În al doilea rând, rolul programatorilor nu trebuie redus la acela de roboți care codifică designul altciva.

- Programatorul individual ar trebui să fie expertul în găsirea acelei soluții (design detaliat și cod) care rezolvă cel mai bine de o anumită problemă.

- **Foarte important:**

- (1) Trebuie insistat pentru a construi un cadru cât mai solid atât pentru programe cât și pentru fișierele de date.
- (2) Programatorii individuali trebuie îndrumați să se concentreze pe conceperea celui mai bun cod posibil pentru a se potrivi în acel cadru.

- (4) **Designul fișierelor**

- Acesta este însoțitorul secțiunii de proiectare a programului.
- Definește în detaliu toate fișierele sistem numite și seturi de date sistem.
  - Acestea sunt fișiere care sunt accesate de mai multe module de program.
  - Definirea amănunțită a acestor fișiere ajută proiectanții să evite o mulțime de probleme mai târziu.
- Multe proiecte au eșuat deoarece programatorii individuali au proiectat fișiere în mod independent și au descoperit ulterior că alții programatori au avut în minte diferite designuri de fișiere și și-au scris programele în consecință.
  - Povestea lui Metzger: Într-un caz memorabil, două echipe au dezvoltat subsisteme majore de programe, fiecare bazat pe conceptul său despre cum urmau să fie fișierele de sistem. După mai mult de un an de muncă, ei au descoperit că cele două subsisteme erau lamentabil de incompatibile, deoarece fișierele pe care fiecare le presupunea erau lumi separate. Un subsistem a fost casat; la fel a fost managerul ei. Evitarea unei astfel de lipse grave de comunicare este una dintre cele mai urgente responsabilități ale managementului.

- (5) **Flux de date**

- Acesta este un fel de [rezumat executiv al designului](#), destinat cu precădere conducerii superioare non-tehnice.
  - Acest subiect va fi abordat mai târziu în cadrul acestui capitol.

## **1.2 Designerii**

- **Designerii:**

- Trebuie să fie, mai presus de orice, **programatori experți**.
- Cel puțin unii dintre ei ar fi trebuit să fi fost puternic implicați în activitatea de analiză a problemei.
- La modul ideal, unii dintre designeri ar trebui să alocați ca programatori sau manageri de nivelul 1 (supervizori) în timpul fazei de programare, pentru a oferi cât mai multă continuitate posibilă.
- Un **designer** bun:
  - (1) Trebuie să ajungă rapid **la miezul problemei** și să nu rămână agățat, rătăcind pe aleile întunecate ale detaliilor.
  - (2) Trebuie să fie **practic**.
  - (3) Trebuie să știe **ce se poate aștepta în mod rezonabil** de la diferitele componente ale întregului sistem: mașini, programe și oameni.
  - (4) Și mai presus de toate, trebuie să **comunice**.
    - Există programatori tehnici superbi care valorează cât zece programatori standard în ceea ce privește abilitățile tehnice, dar nu știu să comunice.
    - Ei sunt mulțumiți și extrem de productivi, dacă li se decupează o porțiune mare din sistem, li se definesc interfețele acesteia cu restul sistemului și sunt lăsați liber să acționeze.
    - Acești oameni pot fi mai folositori în faza de programare decât în faza de proiectare.
- Managerul trebuie să cunoască **înclinațiile tehnice** ale designerilor.
  - Prejudecările puternice ar putea foarte ușor să împiedice evaluarea compromisurilor unui design solid.
  - Un designer principal care înclină întotdeauna spre limbaje de asamblare, de exemplu, nu poate să nu dea niciodată idei relevante în contextul unui limbaj de nivel înalt.
  - Prejudecările sunt inevitabile, dar dacă existența lor este cunoscută, probabil pot fi evitate din timp o multime de probleme.

### **1.3 Ambientul de proiectare**

- Proiectanții necesită condiții speciale pentru a lucra.
  - Dacă managerul este o persoană ordonată, care nu suportă argumentele și conflictele, trebuie să izoleze designerii și să-i lase să lucreze singuri.

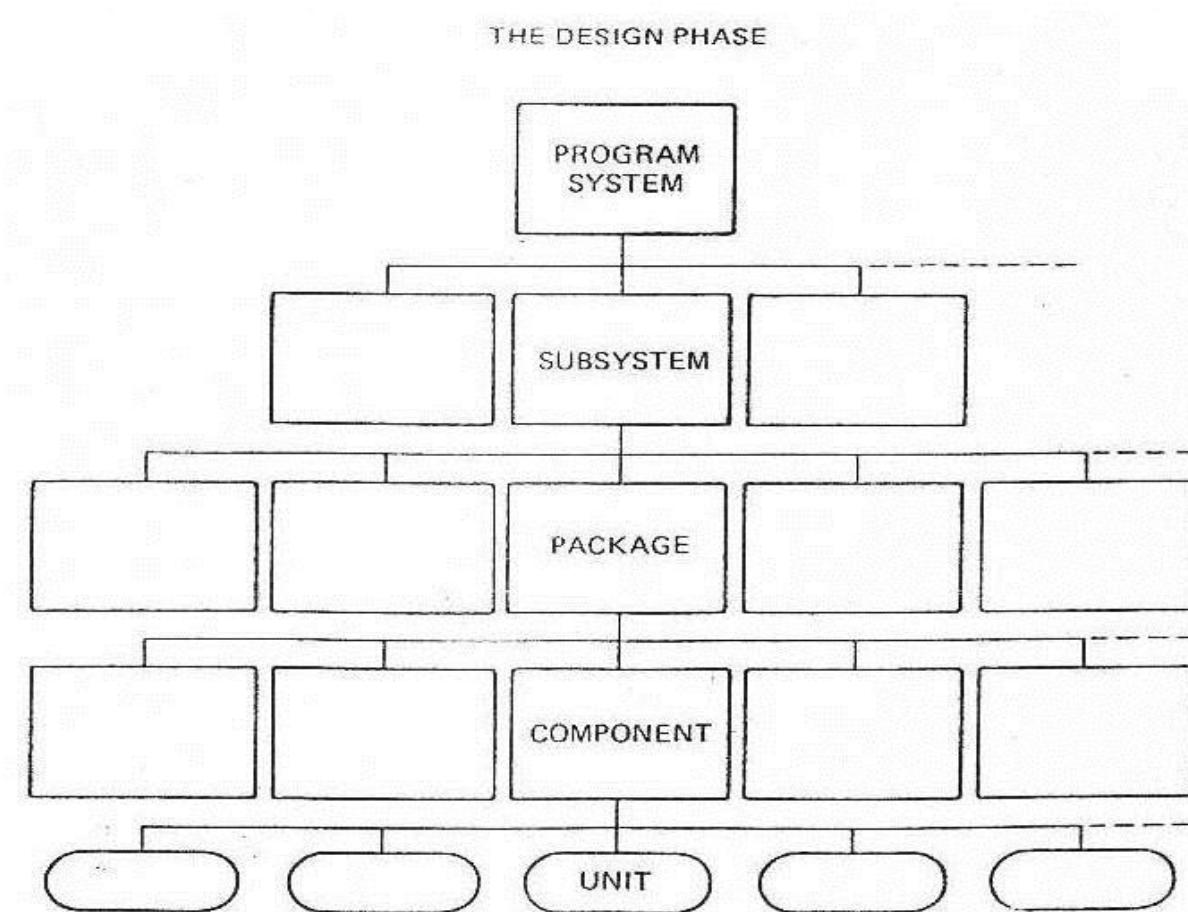
- Dacă există un loc într-un proiect pentru un pic de haos, probabil că este aici în zona de design.
- Designerii nu trebuie deranjați cu întreruperi constante, întâlniri inutile sau treburi care nu au legătură directă cu munca lor de proiectare.
- Managerul de proiect trebuie să se asigure că efortul de proiectare este condus de cineva care are coloana vertebrală și competența tehnică de a rezolva conflictele (uneori în mod arbitrar) pentru a continua cu treaba.

## 1.4 Ghid de proiectare

- Proiectarea unui sistem de programe nu este atât de misterioasă pe cât ar de de înțeles unii designeri.
  - Există câteva linii directoare perfect rezonabile care se aplică pentru proiectarea oricărui sistem de programe și, de altfel, oricărui alt tip de sistem.
- Designerii buni vor respecta aceste linii directoare aproape automat. Care sunt aceste linii directoare:
  - (1) **Integritate conceptuală**.
    - Când designerii sunt lăsați liber să proiecteze un nou sistem, de obicei, aceștia sunt plini de idei noi și de tehnici inteligente și vor avea un impuls aproape irezistibil de a le încorpora în sistem.
      - Dar dacă sunt cu adevărat buni, vor rezista acestui impuls.
    - Un sistem bine conceput ar fi cel mai bine demarat de un număr foarte mic de oameni, astfel încât să prevaleze **o singură filozofie**.
    - •Un design bun al sistemului nu este o mulțime de frunze frumoase lipite împreună pentru a face un copac; mai degrabă, este un trunchi puternic care susține frunzele grățioase.
    - Este de latitudinea proiectantului șef să asigure **integritatea sistemului și uniformitatea concepției**.
      - El sau ea trebuie să păstreze designul orientat tot timpul către cerințele sistemului.
    - Brooks [37] susține că „**integritatea conceptuală este cea mai importantă considerație** în proiectarea sistemului”.
      - Este mai bine ca un sistem să omită anumite caracteristici și îmbunătățiri anormale, dar să reflecte un set coerent de idei de design, decât să conțină multe idei bune, dar independente și necoordonate.”

- (2) **Modularitate.**
  - **Povestea lui Metzger:** „Când eram băiat, am săpat o mulțime de șanțuri. Am făcut acea treabă inspirată într-un mod foarte metodic. Mai întâi am schițat unde trebuia să meargă șanțul, am săpat un strat de două sau trei picioare cube de pământ cu târnăcopul și apoi am îndepărtat cu lopata pământul liber, astfel încât să pot vedea ce am realizat și să atac următoarea bucată de șanț. Întotdeauna am simțit că îmi pot vedea progresul mai bine așa - și acel mic joc probabil m-a ajutat să-mi mențin un grad ridicat de încredere. Alternativa ar fi fost să lovesc cu târnăcopul mai mult timp, îngrämadând o fâșie de pământ mai mult sau mai puțin fărămițat și apoi să dau cu lopata pentru o lungă perioadă de timp, dar astfel ar fi mai puține momente când aș putea privi înapoi ca să văd clar progresul.”
  - O muncă de programare este ca un șanț. Are un început și (uneori) un sfârșit.
    - Îl poți ataca metodic, având întotdeauna sentimentul bun că știi locul în care te află.
    - Sau poți să te arunci înainte fără vreun obiectiv intermedian în minte, cu excepția aceluia de a te îndrepta spre final.
  - În orice caz, te vei lovi de pietre.
    - Domnul Neat, totuși, va putea să curețe locul din jurul pietrei, să o vadă, să o desprindă sau să o ocolească.
    - Piatra domnului Bull va fi ascunsă de tot acel morman de pământ fărămițat.
  - Într-o manieră similară, proiectanții trebuie să conceapă sistemul de programe în **bucăți** sau **module**, nu numai pentru a ajuta procesul de proiectare în sine, ci și pentru a oferi un ajutor important pentru restul proiectului.
  - **Modularitatea** înseamnă subdivizarea unui job de muncă în **compartimente**. Asta are multe avantaje. Le enumerez cu riscul de a afirma ceea ce este evident:
    - (1) Modularitatea oferă **vizibilitate**.
      - Un sistem devine rapid atât de mare și de complicat încât este dificil să se vadă ce se întâmplă, cu excepția cazului în care poate fi privit și înțeles treptat.
    - (2) Modulele forțează **simplitatea** și, ca rezultat, **mai puține erori**.
    - (3) Modulele sunt foarte adesea **reutilizabile**.
      - Cu cât funcția atribuită unui modul este mai restrânsă, cu atât este mai mare probabilitatea ca acesta să poată fi utilizat în altă parte în acest sistem sau în alt sistem de programe.

- (4) Modulele reprezintă o **bază convenabilă** pentru **alocarea de lucru** către programatori.
- (5) Modulele sunt **blocuri utile** care pot fi puse împreună (integrate) într-un mod deliberat și controlat în timpul testării, indiferent dacă se lucrează de sus în jos sau de jos în sus.
- (6) Modulele oferă o **bază convenabilă** pentru **raportarea progresului și păstrarea statisticilor**.
- (7) Modularitatea face **modificările ulterioare mai ușor de efectuat**.
- **Modulul**, aşa cum este folosit în acest curs, este un **termen general** care se aplică unei **părți clar identificate a sistemului de programe la orice nivel din ierarhie**.
- Fig. 5.1.4.a prezintă o ierarhie generală a modulelor care cuprind un sistem de programe.
- Modulele de la fiecare nivel sunt denumite astfel: *unitate, CSCI, componentă, pachet, subsistem, sistem*.



#### **Fig.5.1.4.a. Descompunere WBS a unui sistem (lăzărie)**

- Diagrama din fig. 5.1.4.a spune pur și simplu că sistemul de programe este compus din subsisteme de programe care sunt compuse din pachete de programe care sunt compuse din componente de program care sunt compuse din unități de program.
  - Fiecare casetă din diagramă reprezintă un modul de program.
  - Desigur se poate alege să se numească modulele situate la diferite niveluri cu alte nume, dar indiferent de numele care se aleg, ele trebuie utilizate în mod constant pe tot parcursul proiectului.
  - Este posibil de asemenea, să fie nevoie de mai multe sau mai puține niveluri, funcție de natura sistemului care se construiește.
- Modulul numit **unitate** sau „SU (Software Unit)” este de regulă, cel mai de jos nivel de modul documentat și controlat independent în sistem.
  - O **unitate** care este în general atribuită unui **programator individual**.
  - Programatorul, în codificarea unității, o poate împărți în bucăți mai mici, cum ar fi obiecte, clase, metode, rutine, subroutines, macro-uri sau alte nume exotice, dar atunci când lucrarea este documentată, totul este conținut într-un singur pachet unitar și ordonat numit **unitate**.

- (3) **Definirea interfețelor**

- Deși **proiectanții** trebuie să cheltuiască multă energie în definirea sistemului în termeni de module.
  - Ei trebuie în egală măsură, să acorde o atenție deosebită definirii și documentării **interfețelor dintre module**.
- Specificația de proiectare trebuie să precizeze **modul exact** în care **modulele trebuie să comunice**.
  - Programatorii care scriu unități individuale nu trebuie să aibă niciodată libertatea de a-și combina unitățile în orice mod doresc, respectiv de a-ți crea propriile interfețe.
  - Aceasta nu este locul pentru a acorda libertate programatorului individual.
- Proiectanții au **responsabilitatea** de a include în **documentul de proiectare** **indicații** explicite și detaliante referitoare la următoarele:
  - (1) Cum trebuie să **comunice modulele** cu alte **module**.

- (2) Cum trebuie să comunice modulele cu fișierele de date.
- (3) Cum trebuie să comunice fișierele de date cu alte fișiere de date, inclusiv utilizarea de „indicatoare” care leagă un fișier de altul.
- (4) Cum trebuie să interacționeze operatorii umani cu programele.
- (5) Cum trebuie să transmită programele date, cum ar fi mesajele de eroare, către un operator.
- (6) Modul în care sistemul de programe trebuie să transmită informații către alte sisteme de programe sau către sisteme de echipamente, cum ar fi dispozitivele de afișare.
- (4) **Simplitate.**
  - Domeniul aparte al industriei software încurajează tendința de a folosi construcții complicate și specifice.
    - Dacă poți găsi un designer care să renunțe la astfel de abordări și să-și exprime designul într-un limbaj simplu și ușor de înțeles, ai un adevărat profesionist.
    - „Nu este niciodată neprofesionist... să te faci clar”, spune Robert Gunning [Me81].
  - Pe măsură ce sistemelor de programe li se cere să satisfacă cerințe din ce în ce mai complexe, informaticienii insistă ca proiectarea programelor să devină mai simplă.
    - IBM, lider în căutarea unor tehnici de programare și management mai performante, îndeamnă să fie căutate „*simplățile profunde*” în proiectarea programelor.
    - Se recomandă ca în cadrul unui proiect, modulele să fie concepute atât de simple și explicite în funcția și structura lor, încât să poată fi reutilizate și în alte sisteme.
- (5) **Cuplare simplificată a modulelor.**
  - Nu de puține ori s-au văzut programe în care modulele sunt puternic dependente unele de altele, deoarece funcționarea unui modul depinde de ceea ce se întâmplă într-un alt modul.
    - În cazuri extreme, un modul poate modifica conținutul (uneori instrucțiuni!) în cadrul celuilalt.
    - Se spune că astfel de module posedă cuplare puternică, iar efectul lor este de a complica sistemul.
  - Proiectanții trebuie să urmărească în principiu, cea mai slabă cuplare posibilă – adică cea mai mare independentă – între module.

- **Cuplarea slabă** face fiecare modul:
  - (1) Mai ușor de tratat ca entitate.
  - (2) Mai ușor de proiectat fără a lua în considerare efectul său asupra altor module.
  - (3) Mai ușor de modificat sau de înlocuit ulterior.
- (6) **Implicare minimă**.
  - Larry Constantine a subliniat ideea de „implicare minimă” (angajament minim).
  - Prin aceasta el subliniază ideea că un proiectant trebuie să rezolve problemele de detaliu în sistem doar atât cât este absolut necesar.
    - De exemplu:
      - Când se proiectează spre exemplu, la nivelul de „pachet” al modulului, nu este necesar a se acorda o atenție excesivă unui detaliu care aparține unui modul de nivel inferior, unei „unități”.
      - La orice nivel, trebuie abordat doar atât cât trebuie într-o descriere a designului la acel nivel.
      - Nu trebuie insistat pe detalii și pierdută evidența lucrurilor mari care se întâmplă la acel nivel.
- (7) **Regula cutiilor negre**.
  - Ca un corolar al „implicării minime”, Constantine îi sfătuiește pe designeri să precizeze o funcție cerută ca și o „**cutie neagră**”.
  - O **cutie neagră** presupune definirea atentă a intrărilor și ieșirilor, fără a acorda prea multă atenție structurii interne până la o trecere ulterioară prin proiectare.
    - Joseph Orlicky [6] descrie cutia neagră ca pe un dispozitiv pe care „pur și simplu îl postulăm prin definirea intrărilor și ieșirilor sale”.
    - O cutie neagră, spune el, „face tot ce vrem să facă”.
  - Din nou, se recomandă să se evite implicarea profundă în detalii care într-un design de bază solid nu se văd niciodată.
- (8) **Proiectarea de sus în jos**.
  - Dacă **se proiectează o clădire**:
    - Se începe prin a se lua în considerare ambientul clădirii (de exemplu, unde se va așeza, cât de mare este suprafața de teren, de ce formă).

- Apoi se determină stilul de clădire care s-ar potrivi atât cu destinația sa, cât și cu mediul înconjurător, desigur coroborate cu bugetul clientului și cu gusturile acestuia.
- După aceea, se poate schița structura de ansamblu, fără detalii.
- Având în vedere un acord de principiu cu privire la înălțimea clădirii, numărul de etaje, forma generală, stilul și aşa mai departe, se poate trece la proiectarea părților majore ale structurii — intrările principale, zonele de birouri, zonele de magazine.
- După aceea se trece la proiectarea unor zone specifice, cum ar fi birouri, săli de ședințe, săli de odihnă, magazine, holuri de legătură, casele scărilor, puțuri de lift, etc
- În cele din urmă, se abordează amplasarea ușilor, ferestrelor, luminilor, prizelor, instalațiilor sanitare, decorului și a o mie de alte detalii fără de care clădirea nu ar funcționa.
- Pe tot parcursul procesului se va constata că deciziile cu privire la elementele de nivel inferior au un impact puternic asupra deciziilor luate deja la niveluri superioare.
- Însăși instrumentele și materialele care vor fi utilizate în realizarea lucrării (de exemplu, beton sau sticlă) pot afecta și pot fi afectate de deciziile luate anterior.
- Pe parcursul mai multor iterări ulterioare, deciziile vor fi consolidate și aprobată (uneori în mod arbitrar, pentru a continua cu treaba).
- În final proiectul este gata pentru a fi predat muncitorilor.
- De remarcat, că proiectarea clădirii nu a început prin a se concentra pe dimensiunea toaletelor, după cum proiectele de programe nu încep cu aspectul legate de un anumit modul nesemnificativ.
- În **designul de software** este aceeași situație:
  - Se începe cu cel mai înalt, cel mai grosier și inclusiv nivel de funcții care se perfecționează în pași din ce în ce mai dataliați („rafinare în trepte”) până când toate funcțiile au fost abordate într-o manieră coerentă și sistematică.
    - Aceasta este **designul de sus în jos**.
  - Redactarea proiectului de bază este punctul de plecare pentru întregul proiect detaliat.
- **Documentul de proiectare** de bază trebuie să:
  - (1) Stabilească cadrul general pentru sistemul de programe.

- (2) Stabilească toate convențiile de comunicare.
  - (3) Să rezolve toate problemele de flux și de control.
- Designul modulelor individuale de nivel inferior este însă lăsat la latitudinea programatorilor individuali.
- Proiectanții trebuie să decidă unde să opreasă proiectarea liniei de bază.
  - Nu există nicio modalitate de a preciza exact unde trebuie oprită proiectarea liniei de bază.
  - Acest lucru va fi diferit pentru fiecare proiect și, de fapt, va reflecta experiența, chiar și personalitățile, designerilor, managerilor și a programatorilor.
- De îndată ce **designul de bază** conferă o soluție completă și viabilă la problema menționată în Specificația problemei, el este finalizat.
- (9) **Programe existente. Reutilizabilitatea**
  - Se știe că există foarte puțini **programatori** care nu ar prefera să rescrie de la zero un cod decât să refolosească programele existente.
    - Pe de altă parte, este foarte adesea adevărat că rescrierea unui cod poate fi cursul potrivit.
  - Folosirea unui cod existent poate fi însă o bătaie de cap deoarece:
    - (1) Documentația suport este slabă.
    - (2) Codul nu este exact ceea ce este necesar și trebuie modificat.
    - (3) Modificarea codului altcuiva nu este prea interesantă.
  - Cu toate acestea, designerii sunt neglijenți dacă nu iau în considerare cu sinceritate ce deja există și cum poate fi adaptat noului sistem.
    - Uriile sisteme de operare și bibliotecile de programe suport construite de producătorii de calculatoare și unele case de software costă sute de milioane de dolari; ar putea fi ceva acolo care ar putea fi folosit.
    - Uneori, în loc **să se construiască o soluție**, este posibil **să poată fi cumpărată**.
  - Constantine, Stevens și Meyers [36] susțin un argument puternic pentru construirea (cel puțin în cadrul unei organizații date) de module de program atât de simple și de independente încât să poată fi utilizate pentru nevoi ulterioare, nu doar pentru a satisface cerințele unui sistem sau unui contractat.

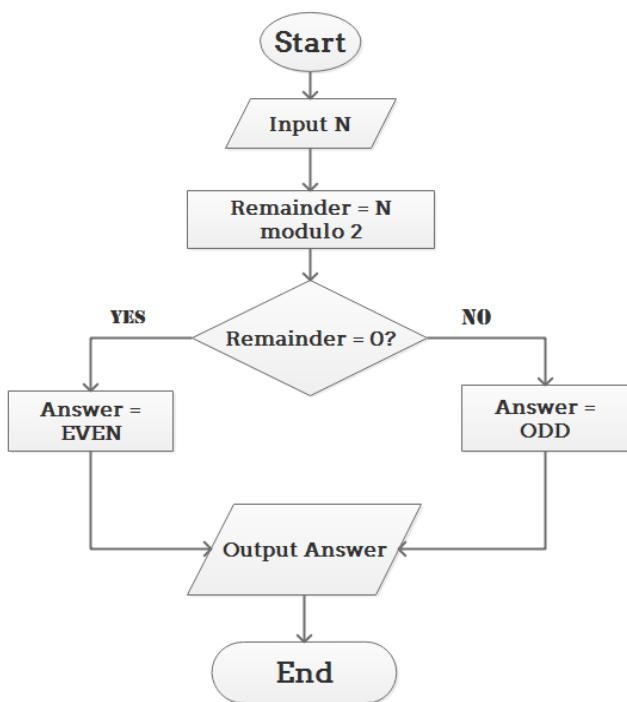
- Cu cât se vor folosi pe scară din ce în ce mai largă designul structurat și tehnologiile orientate pe obiecte, cu atât modulele vor fi tratate ca și „cutii negre”, cu atât mai mult se va ajunge mai aproape de realizarea acestui obiectiv.
  - Câte mii de rutine de „editare” sau „căutare binară” sau „obține” sau „pune” identice au fost construite de-a lungul anilor?
- (10) **Protejarea utilizatorului.**
  - Designerii trebuie să gândească și să acționeze în mod constant având în vedere sistemul în integralitatea sa.
    - Un astfel de sistem include oameni și mașini, nu doar programe.
  - Nu trebuie proiectat ceva care să arate superioritatea intelectuală.
    - Ceea ce se construiește este destinat spre a fi folosit de ființe umane.
    - În plus, de regulă, acei utilizatori umani nu sunt de obicei specialiști în computere.
    - Dacă computerul le facilitează munca, el poate fi acceptat.
    - În caz contrar, îl vor ignora și vor continua să lucreze în manierele lor vechi și confortabile.
    - Dacă computerul și manualele sale complicate de utilizator sunt impuse cu forță, ele pot chiar să saboteze întregul efort depus.
  - **Orlicky** spune despre acest subiect: „*Codifică utilizatorul, nu computerul și reține că scopul principal nu este eficiența sistemului informatic, ci eficiența afacerii*”.
- (11) **Iterații.**
  - Este greșit a se crede că proiectanții au terminat prima dată când au generat o diagramă de flux care pare să includă module pentru a acoperi toate funcțiile necesare.
    - Un design bun este de obicei rezultatul mai multor iterării, uneori chiar al alegerii dintre mai multe alternative posibile.
  - Este foarte posibil ca, lucrurile învățate de designer în timpul proiectării la **nivel de componentă** să îi determine să regândească ceea ce a proiectat la **nivel de pachet**.
    - Acest proces se poate repeta până când modificările sunt minore și proiectantul șef aplică stampila de cauciuc pe care scrie gata.

## 1.5 Instrumente de proiectare

- Majoritatea instrumentelor de proiectare sunt mai utile în **încercarea** sau **documentarea unei idei de design** decât pentru a veni cu ideea în sine.
- Acest fapt este precizat nu cu scopul de a ignora utilizarea unor astfel de ajutoare; este pur și simplu pentru a indica faptul că **conceptualizarea unui design** rămâne în mare măsură un **proces cerebral**.
- În continuare, sunt prezentate câteva dintre **instrumentele de proiectare** cele mai des folosite, aproape toate fiind utile și în timpul analizei problemei.

### **1.5.1 Schemele logice (flow charts)**

- O **schemă logică** (numită și diagramă de flux) este o diagramă care combină **simboluri** și **narațiune prescurtată** pentru a descrie **o secvență de operații** într-un sistem de programe (fig. 5.1.5.1.a).
- **Simbolurile** utilizate includ:
  - (1) Forme geometrice ca dreptunghiuri, paralelograme, cercuri, triunghiuri și romburi.
  - (2) Formele sunt conectate prin linii care indică direcția sau secvența operațiilor.
  - (3) Fiecare simbol are o semnificație specifică:
    - Un dreptunghi poate semnifica un proces de un anume fel,
    - Un romb este de obicei un punct de decizie și așa mai departe.
  - (4) Câteva cuvinte care însotesc fiecare simbol definesc operația reprezentată de simbolul în cauză.



**Fig.5.1.5.1.a.** Exemplu de schemă logică

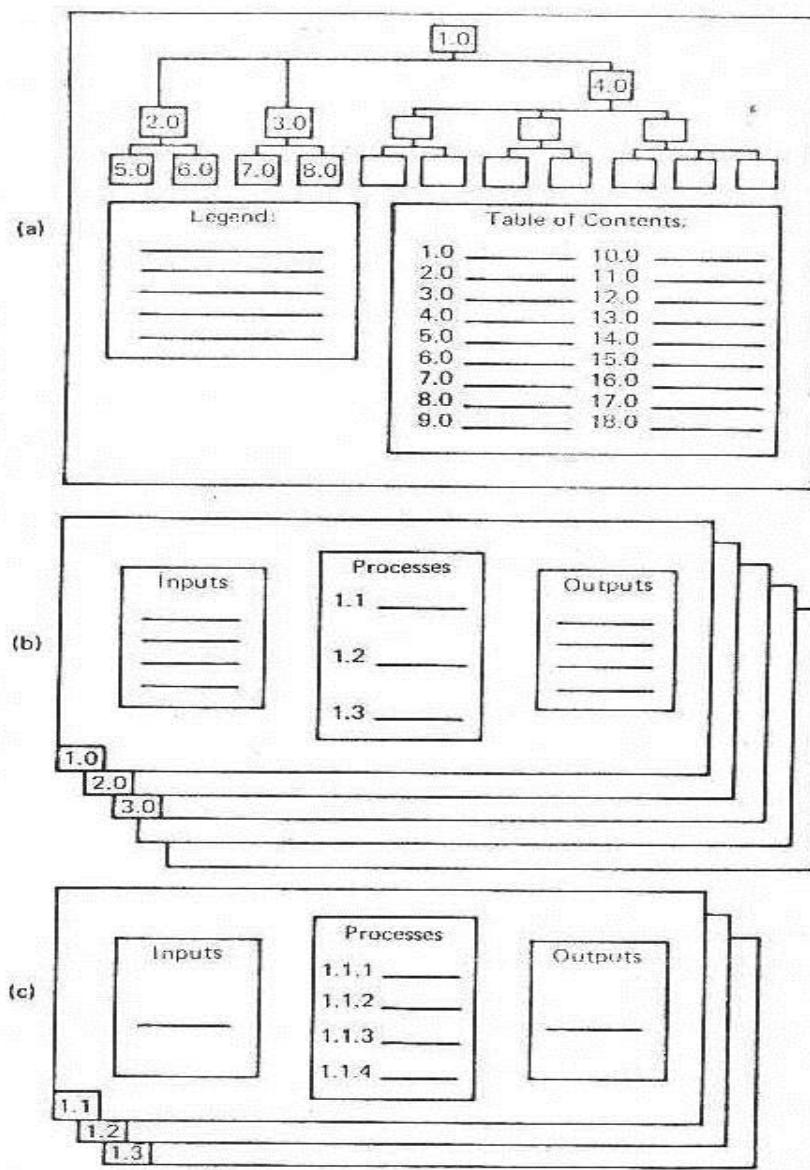
- Aceasta este unul dintre puținele locuri din domeniul programării unde a fost atins un anumit grad de **standardizare**.
  - Organizația Internațională pentru Standardizare (ISO) și Institutul Național American de Standarde (ANSI) au adoptat standarde compatibile pentru schemele logice.
- **Schemele logice** au fost folosite cândva ca formă principală de documentare pentru aproape toate programele.
  - Oricât de utile sunt, nu au o reputație deosebită.
  - Ele sunt de obicei, mult mai utile ca instrument de găndire privată al unui programator decât ca mijloc formal și final de documentare a programelor.
  - Programatorii sunt notoriu de neglijenți în a menține diagramele de flux actualizate și în pas cu codul.
- Mai fructuoasă poate fi utilizarea **altor instrumente** mai ușor de actualizat (de exemplu, diagramele HIPO) și o mai mare încredere în **codul în sine comentat** de o manieră specifică pentru a documenta programul.
- Utilizarea **codului structurat**, descris mai târziu, este următorul pas în această direcție.

- Uneori există presiuni care impun utilizarea în continuare a schemelor logice, chiar dacă există și alte instrumente mai bune.
  - O astfel de presiune poate fi insistența clienților, care sunt obișnuiți cu schemele logice și devin irascibili când cineva zguduie barca sugerând o schimbare.
  - În acest caz, dezvoltatorul este confruntat cu o problemă de educație.

### 1.5.1 HIPO

- În mod tradițional, designerii concep programele propuse folosind diferite combinații de **tabele**, **scheme logice** și **narațiune**.
- Schemele logice au fost foarte adesea considerate o combinație între diagramele funcționale și diagramele logice.
  - Adică, ele descriu de regulă în același set de diagrame:
    - (1) **Functii** pe care programele trebuie să le îndeplinească (**ceea ce**).
    - (2) **Structura sau logica** programelor (**cum**).
- Ceea ce este necesar în timpul **fazei de proiectare** este:
  - (1) Mai întâi o concentrare pe **functii**.
  - (2) Apoi, o **exprimare logică** suficientă pentru a-i asigura pe constructori și pe manageri că toate funcțiile sunt complet determinate.
- Există o modalitate rezonabilă de a construi un set de programe care îndeplinește aceste cerințe.
- Astfel, pentru a satisface **designul funcțional**, a fost concepută o metodă de documentare numită **HIPO** care se bucură de o largă acceptare în rândul designerilor, managerilor și programatorilor.
  - Descrierea logicii, deși este în foarte mare măsură implicită în diagramele HIPO, este lăsată în seama altor forme de documentare care pot descrie logica complet.
- **HIPO** înseamnă **Hierarchy plus Input-Process-Output** (Ierarhie și Intrare-Procesare- ieșire) și reprezintă o **tehnologie de tip WBS** orientată spre design. Se compune din:
  - (1) Un **set de diagrame** care arată **descompunerea funcțională a unui sistem de programe** (sau a oricărui sistem) sub forma unor **diagrame ierarhice tradiționale**.

- (2) Diagrame separate care explodează fiecare casetă din diagrama ierarhică într-un set de trei casețe care arată **intrările**, **procesele** și **ieșirile**.
- Figura 5.1.5.2.a prezintă o modalitate de reprezentare funcțională a unui program folosind HIPO.



**Fig. 5.1.5.2.a. Diagramă HIPO**

- Fiecare organizație care utilizează ideea HIPO poate folosi **formate ușor diferite**, dar **elementele de bază** rămân aceleiași.

- Pentru fiecare **casetă** afişată în prezentarea generală există o diagramă separată care arată:
  - (1) **Intrările** în caseta respectivă.
  - (2) **Procesele** (sau „funcţiile” sau „transformările”) pe care trebuie să le efectueze caseta.
  - (3) **Ieşirile** casetei.
- Schema poate fi extinsă la **orice nivel de detaliu** necesar pentru a contabiliza toate funcţiile de o manieră rezonabilă.
  - În mod normal, fiecare casetă din prezentarea generală (Fig.5.1.5.2.a (a)) va fi afişată în diagrame separate (Fig.5.1.5.2.a (b)), aşa cum este indicat în figură.
  - Oricare sau toate aceste diagrame, la rândul lor, pot fi explodate în continuare, ca în (Fig.5.1.5.2.a (c)), până când toate funcţiile sunt luate în considerare la nivelul de detaliu dorit.
- HIPO poate fi utilizat în mod eficient și de către alți proiectanți decât cei care se ocupă de designul de bază.
  - Poate fi folosit de **analiști** pentru a ajuta la **exprimarea cerințelor sistemului**.
  - HIPO ar putea servi drept **documentație principală** a **analisitorilor**.
  - Utilizarea HIPO pentru a exprima **cerințele sistemului** ar putea ușura modalitatea de utilizare a acestora în descrierea **designului funcțional** al **sistemului**.
  - De fapt, ar putea exista o corelație strânsă, vizuală și de substanță, între **diagramele cerințelor HIPO** și **diagramele de proiectare funcțională HIPO** corespunzătoare.
  - În plus, diagramele HIPO pot fi folosite ca documentație de bază a programului.
  - Atunci când sunt combinate cu cod structurat sau pseudocod, diagramele HIPO pot elimina nevoia de scheme logice ca elemente livrabile clientului.

### 1.5.2 Pseudocodul

- După cum s-a menționat mai devreme, schemele logice nu sunt întotdeauna cel mai potrivit vehicul nici pentru dezvoltarea designului nici pentru documentarea acestuia.
- Odată cu utilizarea crescută a **codului structurat pentru programare**, s-a conturat un nou instrument și anume **pseudocodul**, numit și Program Design Language (PDL)

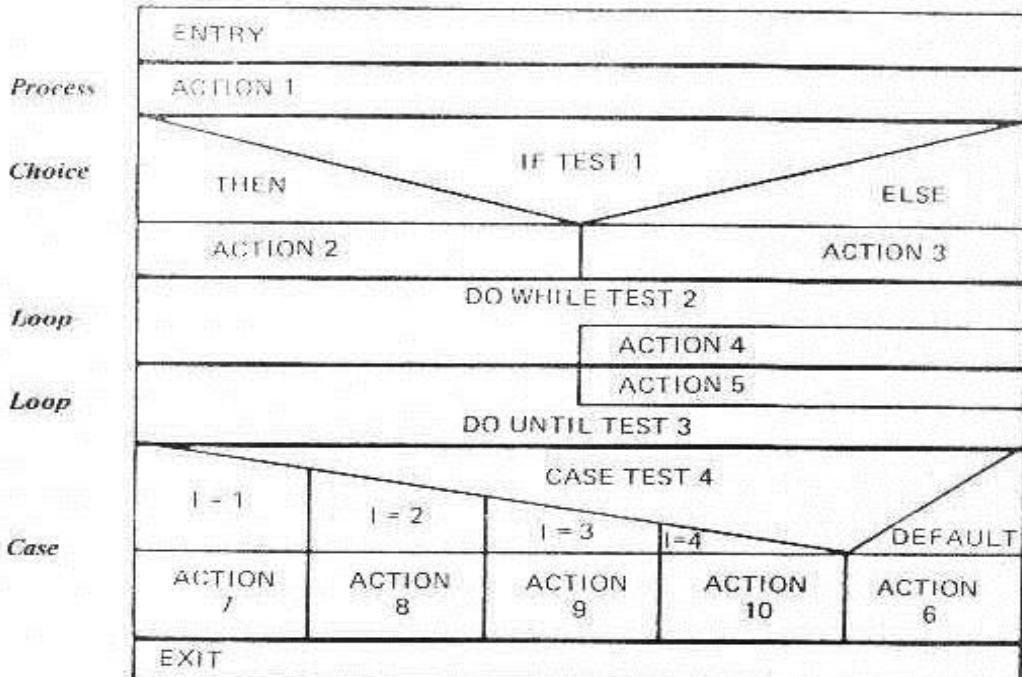
(Limbaj de proiectare a programelor) care este folosit cu succes în mai multe locuri în locul schemelor logice pentru a descrie logica de proiectare.

- **Pseudocodul** este o notație similară ca aspect, formă și semnificație atât cu limbajul vorbit, cât și cu limbajele de programare; este de fapt o punte între cele două.
- **Nu** există un pseudocod unic.
  - Mai degrabă, există o serie de scheme utilizate în diferite organizații, adaptate atât la nevoile respectivei organizații, cât și la limbajul(ele) de codificare utilizat(e).
- De fapt, pseudocodul poate lua orice formă dorește un designer sau programator individual, lucru care, cu siguranță trebuie evitat.
- Pseudocodul trebuie să fie standardizat cel puțin pe parcursul unui proiect dat, dacă nu la nivelul întreagii organizații sau companii.
  - Dacă nu este standardizat, desigur, își va pierde valoarea ca mijloc clar de comunicare.

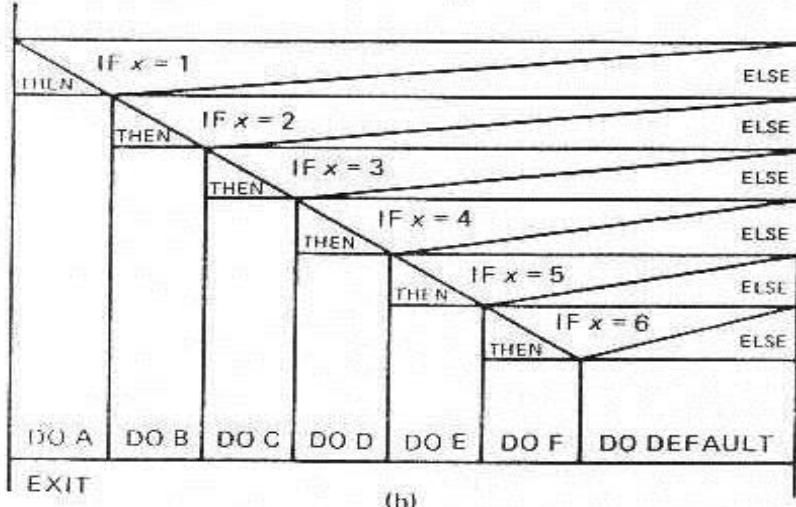
### 1.5.3 Diagrame structurate

- **Diagramele structurate** sunt un mod pictural de exprimare a logicii programului, sau a structurii, într-un mod riguros, lizibil de sus în jos.
- Aceste diagrame se bazează pe **setul restrâns de convenții** utilizate în **programarea structurată**.
- Există o serie de scheme diferite pentru desenarea unor astfel de diagrame.
- Figura 5.1.5.4.a (a) prezintă un astfel de set de convenții pentru reprezentarea fiecărui tip de structură de program.
- Figura 5.1.5.4.a (b) prezintă o porțiune dintr-o diagramă structurată reală în care programul trebuie să facă o serie de alegeri succesive.
- Diagramele ca cele prezentate în aceste figuri sunt adesea numite **diagrame Nassi/Shneiderman** sau **diagrame Chapin**, după autorii lor.

### THE DESIGN PHASE



{a}



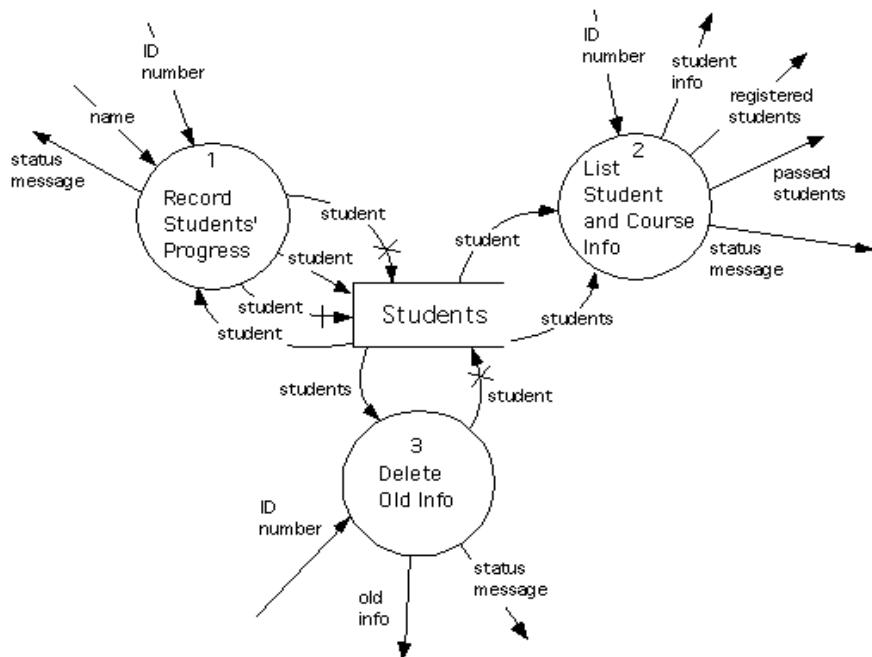
{b}

**Fig. 5.1.5.4.a.** Diagrame structurate

#### 1.5.4 Diagrame de flux de date

- **Diagramele de flux de date** sunt o formă specială de scheme logice (organigramă), utilă atât pentru **analisti**, cât și pentru **proiectanți**.

- Intenția acestei notații, ca și cea a oricărei organigrame, este aceea de a folosi simboluri și text pentru a descrie o secvență de operații (Fig. 5.1.5.5.a)



**Fig. 5.1.5.5.a.** Exemplu de diagramă de flux de date

- Există însă o diferență semnificativă: **o diagramă a fluxului de date** descrie sistemul din **punct de vedere funcțional**, dar ține cont foarte puțin sau deloc de **structura actuală a sistemului**.
  - De exemplu, într-un sistem de procesare a mesajelor, poate fi foarte util să se descrie ce se întâmplă cu un mesaj pe măsură ce acesta trece prin sistem: acceptarea și decodarea mesajului; efectuarea de verificări și corectări de erori; extragerea datelor relevante din mesaj; arhivarea datelor; actualizarea afișajelor afectate de conținutul mesajului; tipărirea rapoartelor rezumative ale tuturor mesajelor primite; și în cele din urmă ștergerea datelor mesajului din sistem.
  - Aceste operațiuni pot fi efectuate de mai multe seturi diferite de programe pe o perioadă lungă de timp (de exemplu, zile), cu implicarea frecventă a operatorilor umani.
  - Schemele logice obișnuite sau alte mijloace de descriere a programelor pot să nu dea o idee clară despre ceea ce se întâmplă în sistem, cu excepția cazului în care cititorul este deja foarte familiarizat cu acesta.

- O diagramă de flux de date poate ajuta la clarificarea procesului:
  - (1) Afisând mai degrabă **evenimente** decât **programe**.
  - (2) Prin **urmărirea traseului unei bucăți majore de date** (fluxului datelor), mai degrabă decât prin **descrierea logicii programului**.
- În continuare se propune un format convenabil pentru o diagramă de flux de date:
  - (1) Se împarte o pagină la mijloc și se afișează simbolurile de flux conectate în jumătatea dreaptă a paginii, cu textul sau comentariul aferent, în partea stângă.
    - (2) Simbolurile și textul lor explicativ trebuie păstrate întotdeauna pe aceeași pagină.
    - În acest caz, textul nu este tipul prescurtat întâlnit de obicei în diagrame de flux.
    - În schimb, este un comentariu simplu care descrie ceea ce se întâmplă în diagrama din partea dreaptă a paginii.
  - (3) Această descriere a fluxului este deosebit de utilă persoanelor al căror interes principal este în prezentarea generală.
- Simbolurile care se utilizează într-o diagramă de flux de date pot fi destul de diferite de cele folosite pentru a descrie logica programului.
  - De exemplu, dacă sistemul dezvoltat implică nave spațiale, radare și dispozitive de afișare, se pot utiliza simboluri care reprezintă acele dispozitive pentru a facilita cititorului înțelegerea diagramei.

### 1.5.5 Tabele de decizie

- Un tabel de decizie este o modalitate simplă și convenabilă de a rezuma o serie de situații „dacă-atunci”.
- Un tabel de decizie arată dintr-o privire ce acțiune trebuie întreprinsă dacă există o anumită condiție sau un set de condiții (fig. 5.1.5.6.a.).

THE DESIGN PHASE

Example (a)	Program A	Program B	Program C	Program D	Program E
Switch 1	X				
Switch 2	X	X	X		
Switch 3	X				X
Switch 4	X				X

Example (b)	1	2	3	4
Request is 1st class	Y	Y		
Request is tourist			Y	Y
1st class available	Y	N		
Tourist available			Y	N
Issue 1st class	X			
Issue tourist			X	
Place on wait list		X		X

**Fig.5.1.5.6.a.** Tabele de decizie

- Exemplul (a) din Figura 5.1.5.6.a arată ce program (numit A, B, C, D și E) sau combinație de programe trebuie apelat și executat ca răspuns la oricare dintre cele patru acțiuni de comutare de către un operator. Dacă comutatorul 1 este pornit, se execută numai programul A; dacă comutatorul 2 este pornit, se execută programele A, B și C; și aşa mai departe.
- Tabela nu spune neapărat nimic despre **combinăriile de comutatoare**, dar ar putea fi ușor construit pentru a face acest lucru. Exemplul este foarte simplu, dar arată câteva informații vitale dintr-o privire.
- Cu cât este mai mare **numărul de comutatoare** și cu cât este mai mare **numărul de programe**, cu atât tabelul devine mai util.
  - În exemplul (b) din Figura 5.1.5.6.a este prezentată o utilizare oarecum diferită a unei tabele de decizie. Aici se folosesc programe pentru a

determină ce fel de bilet să se emită unui pasager al companiei aeriene. În tabel, Y înseamnă da, N înseamnă nu și X înseamnă acțiune. Din nou, exemplul este simplu, dar fără îndoială, acest tabel ar putea fi extins pentru a acoperi un set mai complex de circumstanțe. S-ar putea adăuga spre exemplu un rând intitulat „Este alternativa acceptabilă?”. Apoi, dacă este solicitată clasa întâi, dar indisponibilă, iar turistul este disponibil, acțiunea ar fi eliberarea unui bilet turistic, mai degrabă decât introducerea pasagerului pe o listă de așteptare.

- Practic există un număr nelimitat de utilizări practice ale unor astfel de tabele, deoarece acestea pot fi concepute pentru a afișa într-un singur loc toate deciziile posibile pentru un anumit set de condiții.
  - Diagramalele obișnuite care conțin aceeași logică ar putea acoperi multe foi de hârtie și ar fi mult mai dificil de citit și de înțeles.
- În plus, **tabelele de decizie** sunt ușor de verificat din punctul de vedere al completitudinii prin inspectarea rândurilor versus coloane.
  - Este mult mai dificil să te uiți la o diagramă de flux și să vezi dacă toate combinațiile necesare de condiții au fost sau nu luate în considerare.

### 1.5.6 UML

- (1) **Definiția UML**
  - Unified Modeling Language (UML) este un limbaj pentru specificarea, vizualizarea, construirea și documentarea artefactelor sistemelor software, precum și pentru modelarea afacerilor și alte sisteme non-software.
  - UML reprezintă o colecție a celor mai bune practici de inginerie care s-au dovedit de succes în modelarea sistemelor mari și complexe.
  - Conceptul de artefact joacă un rol central în filosofia UML.
  - Artefactele primare ale UML sunt de fapt definiția UML în sine și modul în care este utilizată pentru a produce artefakte de proiect.
  - Artefactele Proiectului de Dezvoltare sunt **vederi** (views), **modele** și **diagrame** definite de UML.
  - În ceea ce privește **vederile** unui model, UML definește următoarele diagrame grafice:
    - **Diagrame de caz de utilizare** (Use case diagram).
    - **Diagrame de clasă** (Class diagram).
    - **Diagrame de comportament** (behavior diagram):
      - **Diagrama de stare** (statechart diagram).

- Diagrama de activități (activity diagram).
- Diagrame de interacțiune (interaction diagrams):
  - Diagrama de secvențe (sequence diagram).
  - Diagrama de colaborare (collaboration diagram).
- Diagrame de implementare:
  - Diagrama componentelor (component diagram).
  - Diagrama de desfășurare (deployment diagram).
- Deși uneori acestor diagrame le sunt atribuite și alte nume, lista prezentată precizează numele diagramelor canonice.
  - Aceste diagrame oferă perspective multiple ale sistemului aflat în analiză sau dezvoltare.
  - Modelul de bază integrează aceste perspective astfel încât să poată fi analizat și construit un sistem autoconsistent.
  - Aceste diagrame, împreună cu documentația de susținere, sunt artefactele primare pe care le vede un modelator, deși UML și instrumentele de sprijin vor oferi un număr de vederi derivate.
    - O întrebare frecventă a fost: „De ce UML nu acceptă diagramele fluxului de date (DFD)?” Răspunsul: „Fluxul de date și alte tipuri de diagrame care nu au fost incluse în UML nu se potrivesc la fel de clar într-o paradigmă consecventă orientată pe obiecte”.
  - Diagramele de activitate și diagramele de colaborare realizează o mare parte din ceea ce își doresc oamenii de la DFD și apoi ceva.
  - Diagramele de activitate sunt utile și pentru modelarea fluxului de lucru.
- (2) **Motivația definirii UML**
  - De ce **modelăm**.
    - Dezvoltarea unui model pentru un sistem software industrial înainte de construcția sau renovarea acestuia este la fel de esențială ca și a avea un model pentru o clădire mare.
    - Modelele bune sunt esențiale pentru comunicarea între echipele de proiect și pentru asigurarea solidității arhitecturale.
    - Se construiesc modele de sisteme complexe deoarece astfel de sisteme pot fi de regulă înțelese în integralitatea lor.
    - Pe măsură ce complexitatea sistemelor crește, la fel crește și importanța tehnicilor bune de modelare.

- Există mulți factori suplimentari ai succesului unui proiect, dar un standard riguros al limbajului de modelare este un factor esențial.
- Un **limbaj de modelare** trebuie să includă:
  - (1) **Elemente de model** — concepte fundamentale de modelare și semantică.
  - (2) **Notății** — redarea vizuală a elementelor modelului.
  - (3) **Ghiduri** — modalități asistate de utilizare.
- În fața unor sisteme din ce în ce mai complexe, **vizualizarea** și **modelarea** devin esențiale.
  - UML este un răspuns bine definit și larg acceptat la această nevoie.
  - Este limbajul de modelare vizuală preferat pentru construirea de sisteme orientate pe obiecte și bazate pe componente.
- Tendințe în **industria software**
  - (1) Pe măsură ce valoarea strategică a software-ului crește pentru multe companii, industria caută tehnici de automatizare a producției de software.
  - (2) Se caută noi tehnici pentru a îmbunătăți calitatea și a reduce costurile și timpul de lansare pe piață.
    - Aceste tehnici includ:
      - (1) **Tehnologia componentelor**.
      - (2) **Programarea vizuală**.
      - (3) **Utilizarea de modele** (patterns).
      - (4) **Cadre** (frameworks).
    - (3) Se promovează de asemenea, tehnici de gestionare a complexității sistemelor pe măsură ce acestea cresc în domeniul de aplicare și scară.
      - În special, este cunoscută necesitatea de a rezolva probleme arhitecturale recurente, cum ar fi:
        - (1) **Distribuția fizică**.
        - (2) **Concurența**.
        - (3) **Replacarea**.

- (4) **Securitate.**
- (5) **Echilibrarea (balansarea) sarcinii.**
- (6) **Toleranță la erori.**
- (4) Dezvoltarea exponențială a internetului la nivel mondial a simplificat unele lucruri, dar în același timp a exacerbat aceste probleme arhitecturale.
- (5) Complexitatea variază funcție de domeniul de aplicare și de faza procesului de dezvoltare.
  - Una dintre motivațiile cheie din mintea dezvoltatorilor UML a fost accea de a crea un set de semantici și notații care să abordeze într-un mod adecvat toate scările de complexitate arhitecturală, în toate domeniile.
- Prioritatea **convergenței industriale**
  - Înainte de UML, nu exista un limbaj de modelare consacrat. Utilizatorii au trebuit să aleagă dintre multe limbaje de modelare similare cu diferențe minore în puterea expresivă generală.
  - Marea majoritate a limbajelor de modelare împărtășesc de fapt un set de concepe general acceptate dar care sunt exprimate în maniere diferite în diferite limbiaje.
    - Această lipsă de acord a descurajat noii utilizatori să intre pe piața tehnologiei obiectelor și să facă modelare de obiecte, fără a extinde foarte mult puterea modelării.
    - Utilizatorii doreau ca industria să adopte unul sau cât mai puține limbaje de modelare acceptate pe scară largă, potrivite pentru uz general.
    - Unii furnizori au fost descurajați să intre în zona de modelare a obiectelor din cauza necesității de a accepta multe limbaje de modelare similare, dar diferite.
    - În special, oferta de instrumente suplimentare a fost substanțial redusă deoarece furnizorii mici nu își puteau permite să suporte mai multe formate diferite care proveneau de la mai multe instrumente de modelare front-end diferite.
  - Este important pentru întreaga industrie a obiectelor să încurajeze instrumentele și furnizorii de bază, precum și produsele de nișă care răspund nevoilor grupurilor specializate.

- Costul perpetuu al utilizării și susținerii multor limbaje de modelare a motivat multe companii care produc sau folosesc tehnologia obiectelor să susțină și să susțină dezvoltarea UML.
- În timp, deși s-a constatat că UML nu garantează succesul proiectelor, totuși el îmbunătățește multe lucruri.
  - (1) Scade semnificativ costul perpetuu al instruirii și al reorientării la schimbarea între proiecte sau organizații.
  - (2) Oferă oportunitatea unei noi integrări între instrumente, procese și domenii.
  - (3) Permite dezvoltatorilor să se concentreze pe furnizarea de valoare pentru afaceri și le oferă o paradigmă pentru a realiza acest lucru.
- (3) **Obiectivele UML**
  - Obiectivele principale de proiectare ale UML sunt următoarele:
    - (3.1) Să ofere utilizatorilor un limbaj de modelare vizuală expresiv, gata de utilizat, pentru a dezvolta și modifica modele semnificative.
    - (3.2) Furnizarea de mecanisme de extensie și specializare pentru extinderea conceptelor de bază.
    - (3.3) Să suporte specificații care sunt independente de anumite limbaje de programare și procese de dezvoltare.
    - (3.4) Să ofere o bază formală pentru înțelegerea limbajului de modelare.
    - (3.5) Să încurajeze creșterii pieței de instrumente destinate orientării spre obiecte.
    - (3.6) Să sprijie concepte de dezvoltare de nivel superior, cum ar fi componente, colaborări, cadre și modele.
    - (3.7) Să integreze celor mai bune practici.
- (4) **Avantajele UML**
  - **Limbajul de modelare unificat (UML)** este un limbaj pentru **specificarea, construirea, vizualizarea și documentarea artefactelor unui sistem intensiv de software**.
  - (4.1) În primul rând, Unified Modeling Language combină conceptele Booch, OMT și OOSE. Rezultatul este un limbaj de modelare unic, comun și utilizabil pe scară largă pentru utilizatorii acestor și altor metode.

- (4.2) În al doilea rând, Unified Modeling Language împinge limitele a ceea ce se poate face cu metodele existente.
  - Ca exemplu, autorii UML au vizat modelarea sistemelor concurente, distribuite, pentru a se asigura că UML abordează în mod adecvat aceste domenii.
- (4.3) În al treilea rând, Unified Modeling Language se concentrează pe un **limbaj de modelare standard**, nu pe un **proces standard**.
  - Deși UML trebuie aplicat în contextul unui proces, experiența cotidiană evidențiază faptul că diferite organizații și domenii problematice necesită procese diferite.
  - De exemplu, procesul de dezvoltare pentru software-ul shrink-wrapped este unul interesant, dar construirea de software shrink-wrapped este foarte diferită de construirea de sisteme avionice hard real-time de care depind viețile oamenilor.
- (4.4) Prin urmare, eforturile s-au concentrat mai întâi pe un **metamodel comun** (care unifică semantica) și în al doilea rând, pe o **notație comună** (care oferă o redare umană a acestor semantici).
- (4.5) Autorii UML promovează procese de dezvoltare care sunt:
  - **Determinate (driven) de cazuri de utilizare.**
  - **Bazate pe o arhitectură centrică.**
  - **Iterative și incrementale.**
- (4.6) UML specifică un limbaj de modelare care încorporează consensul comunității orientate pe obiecte asupra conceptelor de modelare de bază. Permite exprimarea abaterilor în termenii mecanismelor sale de extindere.
- (4.7) Limbajul de modelare unificat oferă următoarele facilități:
  - Semantici și notații pentru a aborda o mare varietate de probleme contemporane de modelare într-un mod direct și economic.
  - Semantici pentru a aborda anumite probleme viitoare de modelare așteptate, în special legate de tehnologia componentelor, calcul distribuit, cadre sau executabilitate.
  - Mecanisme de extensie astfel încât proiectele individuale să poată extinde metamodelul pentru aplicarea lor la costuri reduse. Nu este de dorit ca utilizatorii să schimbe direct metamodelul UML.

- Semantici pentru a facilita schimbul de modele între o varietate de instrumente.
  - Semantici pentru a specifica **interfața cu depozitele** pentru partajarea și stocarea artefactelor modelului.
- (5) **UML - Prezent și viitor**
  - UML este nonproprietary și deschis tuturor.
  - Se adresează nevoilor utilizatorilor și ale comunităților științifice, stabilită de experiența cu metodele de bază pe care se bazează.
  - Mulți metodologi, organizații și furnizori de instrumente s-au angajat să-l folosească.
  - Deoarece UML se bazează pe semantică și notații similare din Booch, OMT, OOSE și alte metode de vârf și a încorporat contribuții din partea partenerilor UML și feedback din partea publicului larg, adoptarea pe scară largă a UML ar trebui să fie simplă.
  - Există două aspecte de „unificare” pe care le realizează UML:
    - În primul rând, rezolvă efectiv multe dintre diferențele, adesea nesemnificative, dintre limbajele de modelare ale metodelor anterioare.
    - În al doilea rând, și poate mai important, unifică perspectivele dintre multe tipuri diferite de sisteme (afaceri versus software).
      - Fazele de dezvoltare (analiza cerințelor, proiectare și implementare).
      - Concepte interne
- (6) **Standardizarea UML**
  - Multe organizații susțin deja UML ca standard al organizației lor, deoarece se bazează pe limbajele de modelare ale metodelor orientate obiect de vârf. UML este gata pentru utilizare pe scară largă.
  - Peima specificație a Unified Modeling Language v. 1.1 a fost adăugată la lista tehnologiilor adoptate de OMG în decembrie 1997. De atunci, OMG și-a asumat responsabilitatea pentru dezvoltarea ulterioară a standardului UML, dezvoltând o serie de variante. Ultima variantă v.2.5.1 datează din decembrie 2017.
- (7) **Industrializare**
  - Multe organizații și furnizori din întreaga lume au adoptat deja UML. Numărul de organizații care susțin este de așteptat să crească semnificativ în timp.

- Aceste organizații vor continua să încurajeze utilizarea limbajului de modelare unificată, punând la dispoziție definiția și încurajând alți metodologi, furnizori de instrumente, organizații de formare și autori să adopte UML.
- Adevărata măsură a succesului UML rezidă în utilizarea acestuia în proiecte de succes și cererea tot mai mare de instrumente de sprijin, cărți, instruire și mentorat.
- (8) **Evoluția viitoare a UML**
  - Deși UML definește un limbaj precis de modelare, nu este o barieră în calea îmbunătățirilor viitoare ale conceptelor de modelare. A abordat multe din tehniciile de vârf, și este de așteptat ca tehnici suplimentare să influențeze versiunile viitoare ale UML.
  - Multe tehnici avansate pot fi definite folosind UML ca bază.
  - UML-ul poate fi extins fără a redefini nucleul UML.
  - UML, în forma sa actuală, este de așteptat să fie baza pentru multe instrumente, inclusiv cele pentru modelarea vizuală, simulare și medii de dezvoltare.
  - Pe măsură ce sunt dezvoltate integrări interesante de instrumente, standardele de implementare bazate pe UML vor deveni din ce în ce mai disponibile.
  - UML a integrat multe idei disparate, astfel încât această integrare va accelera utilizarea orientării obiectelor.
  - Dintre acestea, **dezvoltarea bazată pe componente** este o abordare extrem de promițătoare, care merită să fie menționată.
    - Este sinergică cu tehniciile tradiționale orientate pe obiecte.
    - În timp ce reutilizarea bazată pe componente devine din ce în ce mai răspândită, aceasta nu înseamnă că tehniciile bazate pe componente vor înlocui tehniciile orientate pe obiecte.
    - Există doar diferențe subtile între semantica componentelor și cea a claselor.

### 1.5.7 Matricile de acoperire (Coverage matrices)

- O **matrice de acoperire** este un modalitate de a arăta relația dintre două tipuri de informații.
  - Un **index** dintr-o carte este o matrice de acoperire pentru că arată ce pagini din carte acoperă care subiecte.

- O matrice de acoperire dintr-un document de proiectare a programului poate arăta funcțiile sistemului versus numele modulelor;
  - Adică, pentru o anumită capabilitate menționată în specificația problemei, ce modul de program din specificația de proiectare oferă acea capabilitate.
- O astfel de matrice este și mai utilă în testare, unde poate fi utilizată pentru a enumera funcțiile de testat față de identificările testelor care urmează să acopere acele funcții.
  - *Exemplu: Un manager IBM a folosit o matrice de acoperire într-un alt mod. El a enumerat toate elementele de lucru specificate în mod special în contract pe o axă și numele persoanelor responsabile pentru acele sarcini de-a lungul celeilalte axe. Prima încercare de utilizare a acestei matrice a arătat câteva sarcini neacoperite de nimeni. Când matricea de acoperire a fost transmisă în rândul lucrătorilor, a scos la lumină și faptul că șeful credea că persoana A era responsabilă pentru o anumită sarcină, iar A credea că această sarcină era gestionată de persoana B.*

### 1.5.8 Hărțile de stocare (Storage maps)

- Hărțile de stocare sunt reprezentări imagistice care descriu modul în care sunt utilizate diferitele dispozitive de stocare (nucleu, disc, bandă, CD-uri etc.).
  - În cele mai multe cazuri, este suficientă o diagramă simplă care arată modul în care sunt utilizate diferite blocuri de stocare (adică de către care programe și în ce scopuri).
- Dacă spațiul de stocare este alocat dinamic în sistem, hărțile de stocare pot fi încă folosite pentru a răspunde la următoarele întrebări de bază:
  - Unde rezidă fiecare tip de program?
  - Unde se află fiecare tip de fișier de date?
  - Unde sunt zonele de depozitare de depășire (overflow), dacă există?
  - Ce prevederi sunt luate în considerare pentru stocarea de rezervă (backup storage)?

### 1.5.9 Limbaje de programare

- Modul de proiectare poate afecta și va fi cu siguranță afectat de limbajul sau limbajele de programare în care se alege să se codifice sistemul.
  - Dacă limbajul ales este „la nivel superior” (de exemplu, FORTRAN, COBOL, PL/I, PASCAL, Basic, Visual Basic, Delphi, C, C++, C#, Java, J+), spre

deosebire de limbajul de asamblare, se poate utiliza limbajul în sine pentru a exprima o parte din design.

- În orice caz, este foarte important să se trateze selecția limbajului de programare drept una dintre deciziile cele mai importante de proiectare.
  - (1) Adesea, o lucrare este făcută în limbajul X, deoarece acesta este limbajul pe care grupul de programatori al dezvoltatorului îl cunoaște cel mai bine - și uneori acest lucru are sens. Această decizie ar trebui însă să fie analizată și să nu fie luată implicit.
  - (2) Chiar dacă programatorii sunt experți în limbaj de asamblare, poate că ar trebui să treacă la un alt limbaj pentru jobul curent.
  - (3) Sau poate că o parte din jobul, cum ar fi un program supervisor, ar trebui să fie codificat în limbaj de asamblare, iar restul într-un alt limbaj.
- În anumite situații, **mixul de limbaje de programare** poate avea sens.
  - Dacă se ia decizia să se amestece limbajele de programare, este extrem de important să existe certitudinea că designerii înțeleg ce probleme de interfață pot să apară între limbalele alese.
- Iată câteva **întrebări** care trebuie luate în considerare înainte de a selecta **un limbaj de programare** pentru un anumit program:
  - (1) Cât de **frecvent** urmează să fie executat programul?
    - Dacă este executat rar, poate fi un candidat pentru un cod mai puțin eficient care de obicei (dar nu întotdeauna) rezultă din utilizarea unui limbaj de nivel înalt.
    - Dacă programul urmează să fie executat frecvent (ca în cazul programelor de supervizare, dispecerelor, planificatoarelor, programelor de intrare-iesire, sau aplicațiilor încorporate), ar putea avea sens codificarea în limbaj de asamblare sau C.
    - În cazul programelor scrise pentru a rezolva probleme specifice de inginerie în care se caută o soluție matematică, alegerea este de obicei clară: se va utiliza limbajul care permite codificarea, testare și obținerea unui răspuns în cel mai scurt timp calendaristic posibil.
      - Eficiența timpului de rulare a programului poate fi ignorată, deoarece are prioritate rezolvarea problemei. Cu alte cuvinte, programul nu va mai fi folosit după ce a dat un răspuns.
      - Aceste programe numite și one-shot, totuși, sunt în general independente și nu fac parte dintr-un sistem de programe.

- Există în discuție și posibilitatea de a folosi un limbaj orientat pe obiecte?
- (2) Este spațiul de stocare intern al computerului o constrângere importantă?
  - Dacă spațiul de stocare este limitat, poate fi necesar codul de asamblare.
  - În general, un anumit program poate fi codificat folosind mai puțină memorie de bază în codul de asamblare decât în codul de nivel înalt, cu condiția să existe experți programatori în utilizarea limbajului de asamblare.
- (3) Este timpul calendaristic critic?
  - Dacă există și dacă trebuie făcută o alegere între programatori competenți în limbaj de asamblare și programatori la fel de competenți în limbaj de nivel înalt, sete foarte probabil că se poate economisi timp calendaristic mergând cu cei din urmă.
  - Codul de nivel înalt poate fi de obicei scris și testat mai rapid decât codul de asamblare, cu mai puține erori, cu alte cuvinte are o productivitate mai ridicată.
- (4) Cât de competenți și ce experiență au programatorii în limbilajele de programare candidate? Răspunsul la aceasta întrebare poate surmonta toate celelalte considerente.
- (5) Procesorarele de limbaj sau controlere avute în vedere, acceptă toate dispozitivele de intrare-iesire necesare pentru acest job? Mare atenție! Este foarte ușor să se facă greșeli cu consecințe neplăcute.
  - *Metzger prezintă o poveste despre un loc de muncă în care contractul specifică utilizarea COBOL, dar managerul de proiect aflat mai târziu în proiect că COBOL-ul nu suportă toate dispozitivele de intrare-iesire preconizate în cerințe. Luni de zile clientul a insistat asupra aderării la contract și a fost de acord să-l schimbe numai după multe întâlniri și compromisuri, consumatoare de timp și jenantă, între contractor și managementul clientilor la cele mai înalte niveluri. Se poate spune că antreprenorul ar fi trebuit să înțeleagă eroarea înainte de semnarea contractului și că clientul nu ar fi trebuit să fie atât de reticent la modificare, dar ideea este că situația s-a întâmplat.*
- (6) Există posibilitatea și timpul necesar, ca programatorii să fie instruiți într-un nou limbaj de programare? (Acest lucru trebuie analizat înainte de a se semna contractul.)
  - Un programator COBOL care participă la un curs de două săptămâni de codare în limbaj de asamblare nu devine cu siguranță un expert în limbaj de asamblare.

- (7) Limbajul de programare selectat este **acceptat** pe deplin pe **toate mașinile pe care vor fi utilizate?**
  - Presupunând că programele sunt dezvoltate pe o mașină și sunt instalate și menținute pe o altă mașină operațională.
  - Cu excepția cazului în care mașina de dezvoltare poate fi reținută pentru a face modificări viitoare la programe, mașina operațională trebuie să includă în repertoriul programului său de suport același asamblor sau aceleași capabilități de procesare a limbajului de nivel înalt ca și cele utilizate pe mașina de dezvoltare.
- (8) Poate avea sens ca în cazul unei aplicații specifice, **să se codifice mai întâi** unele programe **într-un limbaj de nivel înalt** (pentru a începe ceva să ruleze din timp și pentru a testa conceptele de proiectare (proof of concept)) și apoi **mai târziu să se recodifice** modulele critice în **limbaj de asamblare**?
  - Aceasta este însă un lux care se poate aplica doar proiectelor foarte mari, dar poate fi luat în considerare în special pentru proiectele de sisteme incorporate sau timp-real.
- Toate aceste întrebări au scopul de a face dezvoltatorul să cumpănească bine lucrurile și să se gândească la alternative.
  - În final, totuși, trebuie ales acel limbaj (sau limbaje) de programare, care se pare a fi cele mai potrivit pentru jobul curent.
  - Din nou, factori cum ar fi competența și experiența programatorilor sunt determinanți și pot surclasă orice altceva.
- În ultima perioadă, orientarea marilor companii producătoare de software spre un număr redus de limbaje consacrate cum ar fi C și C++, simplifică foarte mult această activitate. Utilizarea altor limbaje de programare are sens doar în contexte limitate și foarte specifice.

### 1.5.10 Modele de simulare

- Dicționarul definește un model ca „**o descriere sau o analogie folosită pentru a ajuta la vizualizarea a ceva care nu poate fi observat direct**”.
- Există multe tipuri de modele, inclusiv modele fizice la scară, modele matematice, planuri și modele ale artiștilor.
  - Tom Humphrey, expert în modelare de simulare, subliniază că chiar și **calendarul de birou** pe care notezi datele și orele pentru întâlniri și ședințe este un model al unei părți din viața ta.

- Tipul de model de interes avut în vedere în acest context este **un model de simulare pe calculator**.
- În acest caz, **sistemul care se modelează** este exprimat într-un limbaj de **programare**, iar calculatorul execută programele rezultate imitând comportamentul sistemului descris.
  - *De exemplu, dacă se dezvoltă un sistem de control automat al traficului de-a lungul străzilor unui oraș, se poate descrie un sistem de control tentativ într-un limbaj de modelare de simulare adecvat, se poate rula modelul pe un computer în diferite condiții de trafic și se pot determina din model, spre exemplu, unde este posibil să apară blocaje. Se pot modifica parametrii (cum ar fi durata luminilor roșii și verzi) și se poate rula din nou modelul pentru a vedea ce efect au modificările asupra fluxului de trafic.*
- Modelarea prin simulare poate fi un instrument puternic pentru proiectant și, de fapt, poate fi extrem de utilă pe tot parcursul proiectului.
- Deși un **sistem de simulare** poate fi foarte costisitor, în funcție de cât de mult se dorește să se simuleze, în multe situații el merită cu prisosință investiția oferind următoarele avantaje:
  - (1) Abordările alternative de proiectare pot fi încercate sub simulare înainte de angajarea proiectului într-o abordare sau alta.
  - (2) Pot fi descoperite blocajele sistemului și pot fi investigate posibilele remedii.
  - (3) Problemele, cum ar fi epuizarea memoriei interne, strangulări (botleneks, deadlines), pot fi anticipate în avans, astfel încât să existe timp pentru a le evita.
  - (4) Construirea unui model de simulare pentru a testa un proiect propus trebuie să forțeze caracterul complet al designului, deoarece modelatorii trebuie să pună întrebări repetitive designerilor până când modelul în sine este complet.
- Noile tehnologii de dezvoltare software se bazează în principal pe diferite tipuri de modele (RUP).

## 1.6 Evaluarea calității designului

- Un test al **calității designului** este, evident, **gradul de succes** al sistemului de program rezultat în îndeplinirea cerințelor clientului.
  - Cu toate acestea, acest lucru nu este suficient.

- Sistemul de programe poate să funcționeze perfect și să facă treaba dorită și totuși să fie inadecvat, deoarece spre exemplu, este dificil de modificat pentru a satisface nevoile viitoare.
- Cu mult înainte de momentul acceptării, de fapt acum, înainte ca programele să fie scrise efectiv, managerii trebuie să fie capabili să evaluateze **calitatea designului de bază**.
- Următorul **set de întrebări** poate fi folosit ca punct de plecare în **evaluarea calității designului**:
- (1) Sunt **toate funcțiile** specificate în specificația **problemei** luate în considerare și deplin abordate în **specificația de proiectare**?
- (2) Au fost proiectate **interfețele** dintre sistemul de programe și operatorii umani sau **mașini** având drept obiectiv principal **ușurința de utilizare**, chiar și cu sacrificarea simplității în programare?
- (3) Sistemul de programe a fost împărțit în **module** suficient de mici pentru a se potrivi pe o singură foaie de imprimare de computer (de exemplu, 40 până la 60 de linii de cod) sau pe un număr redus de ecrane?
- (4) Fiecare **modul** a fost proiectat pentru a îndeplini **o singură funcție specifică** (se spune că astfel de module posedă „**putere funcțională**”)?
- (5) Fiecare **modul** a fost proiectat având în vedere **independența maximă**?
  - Cel mai bun mijloc de comunicare între module este cuplarea datelor, în care modulul A apelează modulul B, trecând la B datele necesare și primind datele înapoi de la B.
  - Alte mijloace de cuplare asigură mai puțină independență de modul; de exemplu, atunci când modulul A se referă direct la conținutul modulului B.
- (6) Fiecare **modul** are **predictibilitate completă**?
  - Modulele al căror comportament depinde de indicatorii de stare autonomi sunt mai puțin previzibile și mai greu de testat.
- (7) Sunt **regulile de comunicare** de la **modul** la **modul** și **de acces la toate datele** clar și complet formulate?
  - Nu trebuie lăsat loc pentru acorduri private între programatori cu privire la comunicarea între module.
- (8) **Setul de documente** care reprezintă proiectul este **clar, complet și gata** pentru rafinare și codificare ulterioară de către programatori?

## 2 Planificarea proiectului în faza de proiectare

- În paralel cu pregătirea efortului de proiectare de bază, în timpul **fazei de proiectare**, Managerul de proiect realizează **planificarea și pregătirea ulterioară**.
- Acestei lucrări se încadrează în următoarele domenii:
  - (1) **Controlul modificărilor**.
  - (2) **Pregătirea pentru testare**.
  - (3) **Instruirea resurselor**.
  - (4) **Documentație**.

## **2.1 Controlul modificărilor**

- Procesul de proiectare **nu** se termină atunci când **specificația de proiectare** este finalizată.
- Pe toată durata de viață a proiectului, și mai ales în timpul Fazei de Programare, vor fi propuse modificări fie la **Specificația problemei**, fie la **Specificația de proiectare**, fie la ambele.
- Mecanismul pentru **gestionarea și controlul schimbărilor** va fi descris în capitolul următor.
  - Acum este momentul ca să se inițializeze mecanismul.
  - La momentul la care apar primele propuneri de schimbare, totul trebuie să fie gata pentru a le rezolva cât mai rapid.

## **2.2 Pregătirea pentru testare**

- La fel ca majoritatea activităților de pe durata unui proiect, **testarea** trebuie pregătită înainte de momentul în care va avea loc efectiv.
- Testarea va începe în timpul **fazei de programare**.
  - Pregătirea pentru aceasta trebuie realizată în timpul fazei de proiectare.
- Discuțiile detaliate despre testare sunt lăsate pentru capitolele ulterioare, dar pe moment trebuie pregătită scena.

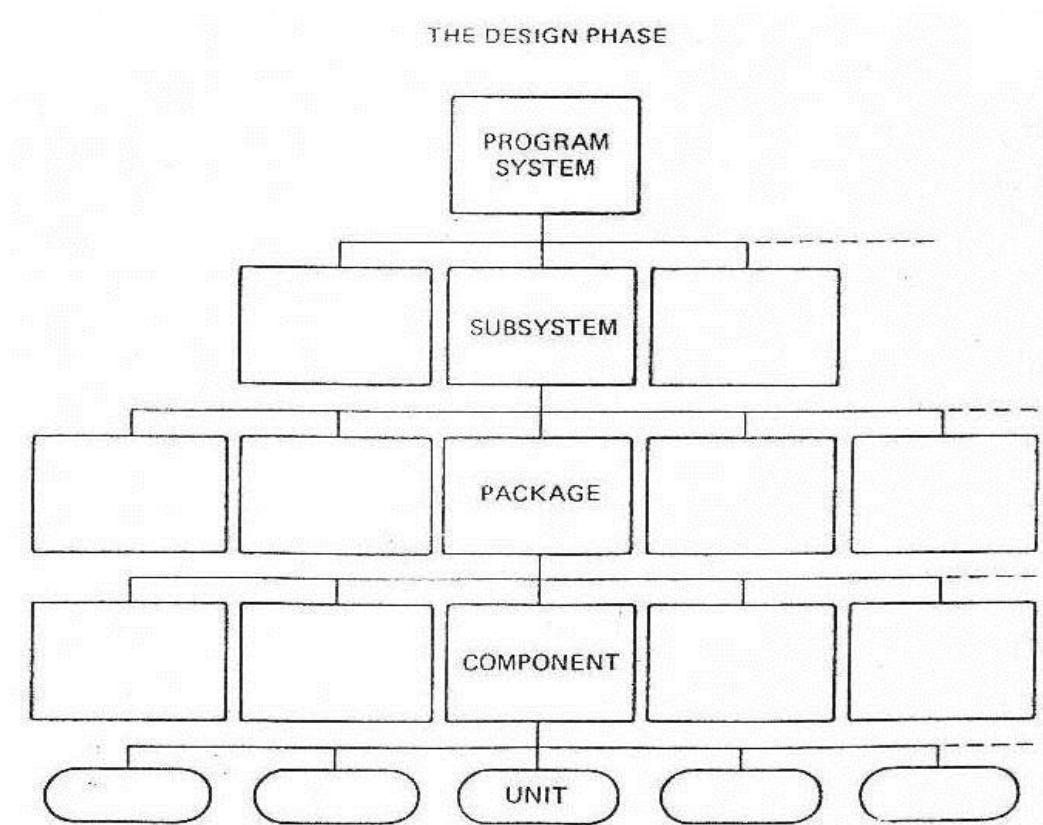
### **2.2.1 Definirea ierarhie de testare**

- Pentru început, trebuie definite tipurile sau nivelurile de testare care urmează să fie efectuate pe proiectul curent.

- Ele trebuie definite, publicate și adoptate.
- Este cunoscut faptul că atunci când sunt folosiți termeni precum „test de integrare” și „test de sistem”, semnificațiile acestora depind de fapt de cine îi folosește.
  - Este foarte important să existe certitudinea că pentru proiectul în dezvoltare există definiții clare, înțelese de toți membrii proiectului, de managementul din afara proiectului și de către client.
- Există **cinci niveluri de testare** ale modulelor care cuprind sistemul de programe.
- Metzger oferă următoarele definiții ale nivelurilor de testare într-o ierarhie de teste (sunt detaliate în continuare în capitolele următoare).
- (1) **Testarea de modul** este testarea efectuată pe orice modul individual înainte de a fi combinat (integrat) cu restul sistemului.
  - Testarea de modul este realizată de către programatori individuali.
- (2) **Testarea de integrare**. Denumit și simplu „integrare”, acesta este procesul de:
  - (2.1) Adăugarea unui nou modul la sistemul în evoluție.
  - (2.2) Testare a acestei noi combinații.
  - (2.3) Repetare a procesului până când în cele din urmă întregul sistem a fost reunit și testat temeinic.
- (3) **Testare sistem**. Sistemul de programe integrat pe care programatorii îl consideră curat este acum rulat printr-o nouă serie de teste.
  - Această serie de teste nu este pregătită sau executată de programatori.
  - Aceste noi teste sunt executate într-un mediu cât mai aproape posibil de ambientul în care ce va funcționa sistemul software.
  - Obiectivul lor principal este acela de a testa programele în raport cu specificația originală a problemei pentru a determina dacă sistemul îndeplinește sau nu funcționalitățile pe care a fost construit să le implementeze.
- (4) **Testarea de acceptanță**. Sistemul de programe este testat în condiții agreate de client, cu scopul de a-i demonstra că sistemul îndeplinește cerințele contractului.
  - În multe cazuri, testul de acceptare este complet controlat de către client sau de către reprezentantul acestuia.
- (5) **Testarea de amplasament (site)**. După instalare în mediul său de operare final, programul este testat din nou pentru a se asigura pregătirea completă pentru funcționare.

## 2.2.2 Testarea de integrare de sus în jos vs. de jos în sus (top-down vs. bottom-up integration testing)

- La fel ca multe alte aspecte ale activității de programare, teoriile și practicile referitoare la testare au suferit multe schimbări de-a lungul timpului.
  - Schimbările au fost atât **filozofice**, cât și **practice**.
- În mod tradițional, un sistem format din mai multe **niveluri de module** sugerate într-un mod generalizat în Figura 5.1.4.a a fost testat „de jos în sus”.



**Fig.5.1.4.a.** System WBS (Hierarchy) (replicate)

- **Testarea de jos în sus** presupune:
  - (1) Modulele de cel mai coborât nivel (în Fig. 5.1.4.a, „unitățile”) sunt codificate și testate mai întâi pe o bază „autonomă”.

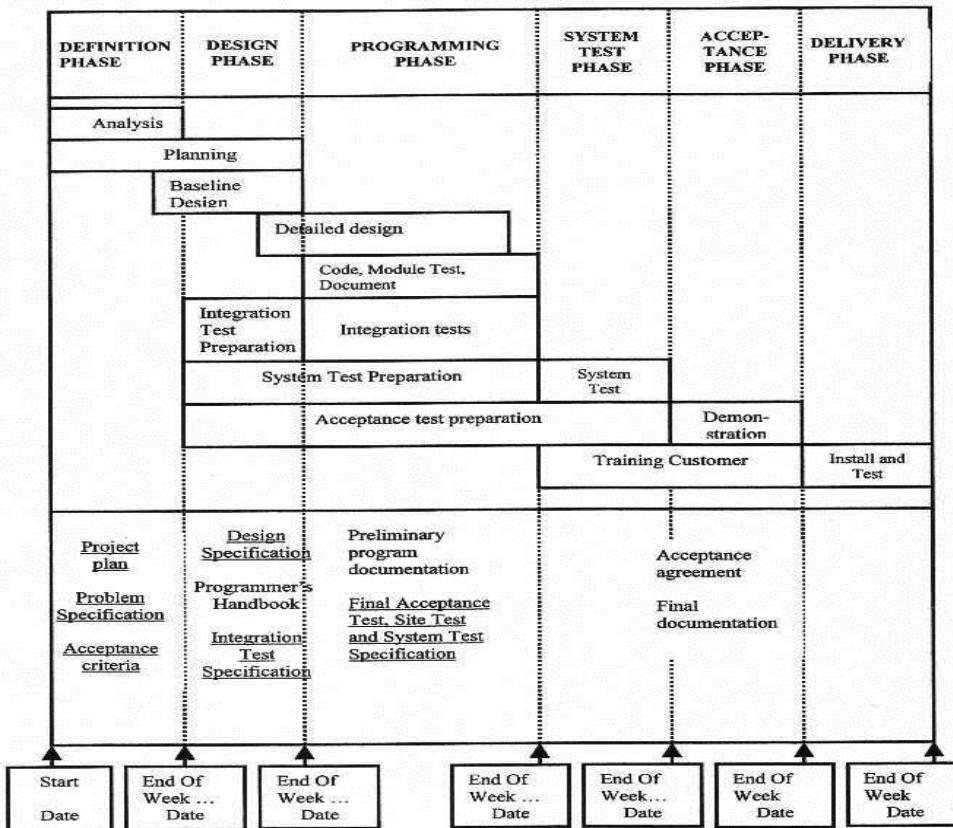
- (2) Când unitățile cuprinse într-un modul de nivel superior („componentă”) sunt gata, acestea urmează să fie combinate și combinația testată până când acea componentă funcționează cu succes.
- (3) Între timp, alte componente sunt pregătite într-o manieră similară și, eventual, testate ca și grupări adecvate de componente.
- (4) Și tot aşa, avansând în sus în ierarhie, până într-o zi — iată! sistemul a fost testat.
- **Testarea de sus în jos** este exact invers, cel puțin din punct de vedere filozofic:
  - (1) Testarea începe cu modulele de nivel superior și continuă în jos coborând în ierarhie.
  - (2) Modulele de cel mai coborât nivel sunt ultimele adăugate la sistem și testate.
- Oricare dintre metodele de testare poate fi aleasă și oricare dintre ele va funcționa, dacă a fost rodul unei activități de planificare și de controlul adecvat.
  - Managerul de proiect, trebuie să înțeleagă ce presupune fiecare metodă și care este miza în momentul în care face alegerea sa.
- Primul lucru de înțeles este că alegerea între testarea de sus în jos și de jos în sus este o alegere între două filozofii sau abordări de bază.
  - Nu sunt neapărat metode de testare care se exclud reciproc, fiecare implicând de fapt și pe celalătă.
- De exemplu:
  - În utilizarea **metodei de jos în sus**, ar fi în mod normal, practic sau necesar **să existe un cadru** în care modulele să poată fi inserate în scopuri de testare.
    - Acest cadru este de obicei o versiune simplă a programului de control al sistemului (un modul de nivel înalt al sistemului).
  - În **testarea de sus în jos**, este adesea necesară codificarea și **testarea foarte devreme a unor module** (cum ar fi un modul de ieșire) care apar la un nivel foarte scăzut în ierarhie.
  - Deci nicio abordare nu este complet de sine stătătoare. În general, va exista întotdeauna un amestec a celor două.
    - Nu este nimic în neregulă cu asta. Ceea ce contează, până la urmă, este ca sistemul să fie bine testat și realizat la timp.

- Ce este însă cu adevărat important este ca o abordare sau alta să fie **selectată ca prioritără** pentru proiectul în dezvoltare.
- Abordarea recomandată de Metzger este **cea de sus în jos**, deși discuțiile ulterioare din acest curs permit utilizarea oricărei abordări.
- Mai jos apare un rezumat al **motivelor** pentru care să se facă această alegere:
  - (1) Metoda de sus în jos este una „naturală”; ea implică însă construirea unui cadru înainte de a se adăuga detaliu.
  - (2) Se armonizează confortabil cu ideile de design de sus în jos și de codificare de sus în jos.
  - (3) Întreaga idee a dezvoltării de sus în jos este aceea a unei progresii naturale. Pe măsură ce noi module din ierarhie sunt adăugate sistemului și testate împreună cu modulele de nivel superior deja testate, sistemul evoluează ca o entitate vie, în creștere, „completă” în orice etapă dată.
  - Metoda de jos în sus este o abordare mai fragmentară, care implică mai multă incertitudine și mai multe surprize atunci când grupuri de module sunt asamblate împreună pentru prima dată.
  - (4) Este mai ușor să fie produse versiuni intermediare ale sistemului final de programe; efectiv, există o versiune intermediară chiar de la început - ceva ce funcționează și arată rezultate. Acest lucru are următoarele avantaje:
    - (a) Se pot arata clientului **rezultate intermediere** mai rapid, evitând astfel șocurile de la sfârșitul proiectului.
    - (b) Se pot furniza **versiuni intermediere**, incomplete ale sistemului.
    - (c) Se poate arăta conducerii că există într-adevăr ceva produs. De regulă, managementul superior este în poziția ingrată de a trebui să accepte și să credă ceea ce i se promite de către dezvoltatori deoarece este atât de greu să vezi cu adevărat acele programe!
  - (5) Integrarea sistemului de programe este realizată în mod continuu pe măsură ce fiecare modul nou este adăugat și testat cu modulele de nivel superior deja existente.
  - (6) În dezvoltarea de sus în jos este nevoie de mult mai puține eșafodaj (cod de suport de testare scris special, sau cum ar fi programele driver fictive).

### 2.2.3 Redactarea specificațiilor de testare

- **Testarea de modul** se face de către programatorul individual fără un document de testare formal.
- Celelalte patru niveluri — **integrare, sistem, acceptanță și testare site** — sunt efectuate conform specificațiilor de testare definite anterior.
- Există un **set separat de specificații de testare pentru fiecare nivel**.
- Multe teste individuale vor servi mai multe categorii de teste.
  - De exemplu, multe, dacă nu toate testele demonstrative de acceptanță pot fi preluate din setul de teste de sistem.
- Figura 5.2.2.3.a. indică în ce moment al ciclului de dezvoltare ar trebui să fie gata fiecare specificație.

**The Model of Project Life Cycle**



**Fig.5.2.2.3.a.** Model de ciclul de viață al unui proiect software

- Este evident faptul că, dacă specificația testului de integrare urmează să fie gata de utilizare când începe testarea integrării, va trebui pregătită în timpul fazei de proiectare.
- În mod similar, specificația de testare a sistemului trebuie scrisă în timpul fazei de programare și aşa mai departe.
- Specificațiile de testare vor fi descrise mai târziu. Aici este suficient să se precizeze că:

- Fiecare **specificație de testare** conține o **definiție** pentru:
  - (1) **Obiectivele testului.**
  - (2) **Criteriile de succes.**
  - (3) **Datele de testare.**
  - (4) **Procedurile de testare.**
- Fiecare specificație este susținută de **cazuri de test.**
  - Un **caz de test** conține:
    - (1) **Informațiile de bază.**
    - (2) **Datele de testare.**
    - (3) **Procedurile detaliate** necesare pentru a executa un set specific de teste.

#### **2.2.4 Definirea procedurilor de testare test**

- Locul în care este permisă creativitatea în testare este atunci când se concep testele și nu atunci când ele se execută.
- În timpul testării vor exista întotdeauna cu siguranță momente de așteptare (mai ales în timpul testării de acceptanță), indiferent cât de bine a fost pregătită testarea; nu trebuie adăugată tensiune prin improvizație.
- Specificațiile de testare trebuie scris în aşa fel încât procedurile, responsabilitățile și rezultatele estimate să fie precizate clar din timp.

### **2.3 Estimarea resurselor**

- Pe măsură ce se apropi sfârșitul fazei de proiectare, ar trebui re-estimate resursele necesare pentru a finaliza lucrarea.
- Presupunerile timpurii despre forța de muncă și timpul de calculator pot fi apreciate acum mai realist, deoarece se cunosc mult mai multe despre jobul de făcut.
- Dacă estimarea nouă este mult mai mare decât cea inițială, aceasta este o problemă pe care dezvoltatorul și clientul trebuie să o rezolveți cumva.
  - Mai bine acum decât la jumătatea fazei de programare.

### **2.4 Documentația**

- Planul de documentare trebuie să fie într-o formă avansată la sfârșitul fazei de proiectare, cu majoritatea documentelor definite și conturate.
- Prima versiune a Manualului Programatorului ar trebui să fie acum gata de distribuție, iar biblioteca de proiect ar trebui configurată.

#### **2.4.1 Manualul programatorului**

- Manualul programatorului este o sinteză care conține informațiile scrise cele mai importante pentru programatori în îndeplinirea atribuțiilor lor.
- Manualul programatorului include (fig. 5.2.4.1.a):
  - (1) Descrierea cerințelor tehnice
  - (2) Designul de bază
  - (3) Software-ul de suport
  - (4) Procedurile de testare
  - (5) Informații hardware
  - (6) Un rezumat al Planului de documentare.

---

### **MANUALUL PROGRAMATORULUI**

**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA REALIZĂRII:**

**REALIZAT DE:**

---

**SECȚIUNEA 1: INTRODUCERE**

#### **1.1 Obiective**

*Manualul se dorește a fi sursa informațiilor tehnice de bază solicitate de toți programatorii din proiect. Informațiile din Manual trebuie considerate „lege” până când o modificare este aprobată și distribuită.*

*Manualul nu este un singur document, ci o colecție de documente pe care fiecare programator din proiect ar trebui să le aibă la indemâna. Este extrem de important ca Personalul Tehnic (responsabil cu emiterea și actualizarea Manualului) să nu permită adăugarea aleatorie a materialelor suplimentare.*

*Manualul este sub formă de caiet de note, pentru a facilita actualizările. Poate fi și în formă electronică pe intranetul companiei. Este împărțit în secțiuni cu filă majoră pentru fiecare componentă și cu sub-file acolo unde este cazul. În forma electronică se adaugă de obicei un mecanism de căutare.*

## **1.2. Scop**

*Manualul ar trebui să se limiteze la subiectele enumerate în această schiță. Există o mulțime de informații relevante pentru proiect (planuri, rapoarte de stare etc.) care nu sunt incluse. Manualul trebuie să fie concis și utilizabil din punctul de vedere al programatorului.*

## **1.3. Publicare**

*Varianta inițială trebuie finalizată aproape de sfârșitul fazei de proiectare. Actualizările ulterioare se fac în două moduri:*

- 1. Actualizare săptămânală de rutină (culoare albastră)**
- 2. Actualizare de urgență 24 de ore (culoare roșie)**

*În ambele cazuri, actualizările sunt gestionate în următoarea manieră:*

- 1. Oricine propune o actualizare, o transmite personalului tehnic.**
- 2. Personalul tehnic primește proiectul, îl analizează, îl corectează, îl aprobă și îl distribuie sau îl publică pe intranet.**

# **SECȚIUNEA 2: PROBLEMA**

## **2.1. Introducere**

*Conține un tutorial care descrie clientului, mediul și jobul care trebuie efectuat (dacă este necesar). Această descriere trebuie să înceapă de la zero și să fie scrisă, astfel încât un nou membru al proiectului să poată să înțeleagă despre ce este vorba. Limita ar trebui să fie de aproximativ 2 pagini.*

## **2.2. Specificatia problemei**

*Include documentul Specificația problemei în integralitatea sa.*

# **SECȚIUNEA 3: TESTAREA**

*Include Planul de testare în integralitatea sa.*

## **SECȚIUNEA 4: PROGRAME SUPORT**

*Descrierea mediilor și instrumentelor de programare disponibile programatorilor și a modului de utilizare a acestora. Fiecare categorie principală de instrumente ar trebui să fie descrisă separat în această secțiune.*

- *Medii de programare*
- *Instrumentul de control al versiunilor*

## **SECȚIUNEA 5: SPECIFICAȚIA DE PROIECTARE**

*Specificația de proiectare este inclusă integral. Acest document conține mai multe subsecțiuni principale, cum ar fi: Conceptul general de proiectare, Standarde și convenții de proiectare, Standarde și convenții de codificare, Proiectarea de bază.*

## **SECȚIUNEA 6: DOCUMENTAȚIA**

### **6.1 Sumarul documentației**

*Aici se include o diagramă cu rezumatul documentației.*

### **6.2. Unelte de documentare**

*Sunt prezentate uneltele utilizate în documentare (dacă există).*

### **6.3. Indexul documentației**

*Include un index de referință detaliat al tuturor documentelor de proiect, actualizat săptămânal (vezi Indexul documentației).*

## **SECȚIUNEA 7: ECHIPMENTE**

*O descriere a hardware-ului operațional și de suport care va fi utilizat în proiect (dacă este necesar). Tipul de informații incluse aici:*

- *Topologia rețelei*
- *Caracteristicile tehnice ale echipamentului*
- *Distribuția geografică a rețelei*

## **SECȚIUNEA 8: GLOSAR**

*Definirea termenilor utilizați în cadrul proiectului, inclusiv denumirile nivelurilor programului, nivelurilor de testare, a jargonul clientilor, nomenclaturii echipamentelor.*

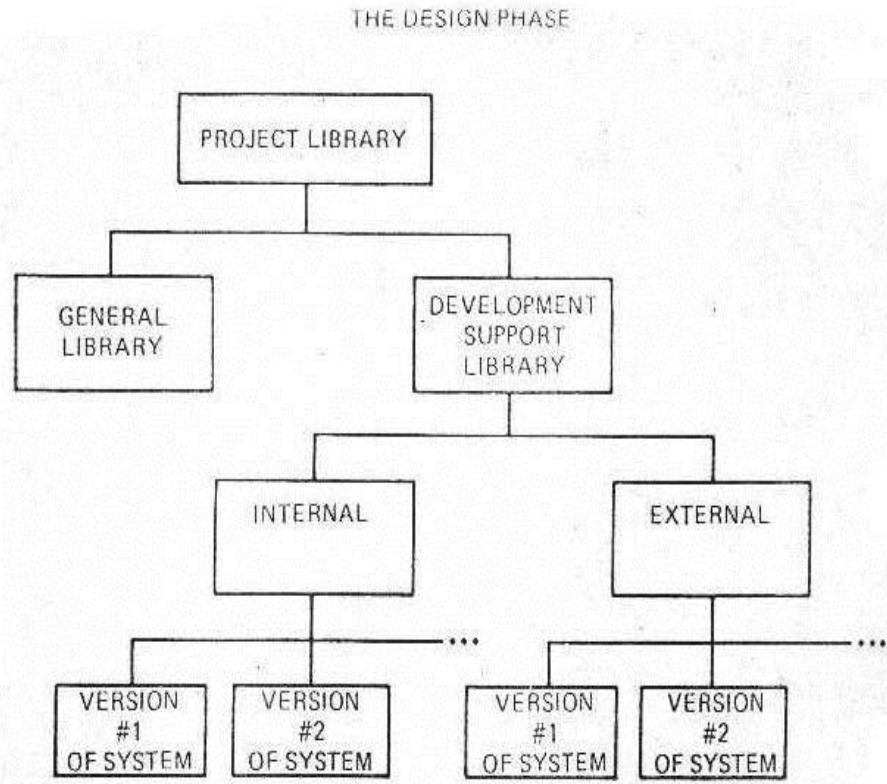
**Fig.5.2.4.1.a.** Model de Manual al programatorului

- **Manualul programatorului** trebuie să fie gata când începe faza de programare.
  - Pregătirea și întreținerea acestuia trebuie încredințate unei **persoane tehnice**, **nu** unui administrator, deoarece este un set important de informații pentru programator.
  - Nu trebuie lăsat să devină prea voluminos, cu alte cuvinte nu este necesar să conțină o secțiune care să permită depunerea oricărui tip de document imaginabil despre proiect.
  - Și, mai presus de toate, nu trebuie incluse în acest manual articole precum cele care descriu detaliat programul.
  - Trebuie limitată nu numai **dimensiunea manualului**, ci și **distribuția** sa.
  - Trebuie păstrat cât mai mic posibil.
    - Cu cât devine mai mare, cu atât este mai puțin probabil ca cineva să-l consulte; iar cu cât lista de distribuție este mai lungă, cu atât actualizarea manualului este mai grea și mai costisitoare.
  - Astăzi, pe baza facilităților intranet, multe proiecte nu mai utilizează manuale. Datele critice sunt stocate în computer și făcute accesibile prin terminale cu ajutorul cărora informațiile pot fi accesate, selectate, pot fi fie tipărite sau afișate pe un ecran, sau ambele.
  - Un mare avantaj al unei astfel de biblioteci centralizată este Acela că poate fi întotdeauna ținută la zi; se evită ciclul adesea lent de publicare și distribuție.

## **2.5. Managementul configurării (Configuration Management)**

- **Managementul configurării** ar trebui să fie pe deplin organizat și să funcționeze la sfârșitul fazei de proiectare.
  - Poate fi implementat în diferite moduri.

- (1) O modalitate de organizare funcțională a managementului configurației o reprezintă **biblioteca proiectului** (Metzger).
  - Acest concept este prezentat în Figura 5.2.4.a și descris mai detaliat în capitolul următor.



**Fig.5.2.4.a.** Model de Bibliotecă a proiectului

- Fiecare articol (document sau modul de program) conținut în bibliotecă trebuie să aibă o **identificare unică**.
- **Nimeni**, în afară de **bibliotecar**, nu trebuie să poată gestiona copia principală a unui document sau a unui modul.
- Aceasta este **un punct de control convenabil** pentru proiect, dar care își pierde o mare parte din valoare dacă membrilor proiectului li se permite accesul liber la bibliotecă.
- Utilizarea bibliotecii va fi discutată în continuare în capitolele următoare.

- (2) O altă modalitate de a implementa managementul configurării este aceea a folosi un instrument specific cum ar fi:
  - ClearQuest (Rational)
  - CVS (sistem de versiuni de control)
  - ClearCase (Rational)
  - Continue
  - CM Sinergy (Telelogic)
  - Etc.,

## 2.6 Instruire

- În timpul fazei de proiectare, ar trebui să se realizeze instruirea programatorilor pentru sarcinile lor în timpul fazei de programare.
- Până la demararea fazei de programare, toți ar trebui să știe:
  - Echipamentul disponibil;
  - Limbajul de programare utilizat;
  - Facilitățile de testare care vor fi utilizate;
  - Definirea problemei;
  - Proiectarea de bază.

## 3 Revizia fazei de proiectare

- Evaluările stării proiectului ar trebui să aibă loc la sfârșitul fiecărei etape din ciclul de dezvoltare, aşa cum a fost stabilit în [Planul de revizuire și raportare](#).
- Deoarece revizia efectuată la **sfârșitul fazei de proiectare** este probabil cea mai critică dintre toate, va fi analizată în detaliu.
  - La sfârșitul fazei de proiectare, proiectul se află aproape de un punct fără întoarcere.
  - Managementul este pe cale să angajeze resursele majore (forța de muncă de programare, forța de muncă de testare și timpul de calculator) și ar fi bine să aibă sentimentul cald că totul este cu adevărat pregătit.

- Odată ce se începe implementarea design-ului de bază, este obositor, costisitor și neadecvat să fie nevoie să oprească totul pentru a face o revizie majoră.
- **Obiectivele** acestei revizii sunt:
  - (1) Să evalueze completitudinea și adecvarea **Proiectului de bază** și a **Planului de proiect**.
  - (2) Să furnizeze conducerii suficiente informații pentru a decide dacă:
    - **Se trece la faza următoare**, sau
    - **Se reiau** anumite părți, sau
    - **Se abandonează proiectul**.

### **3.1 Pregătire reviziei**

- Nu trebuie neglijată pregătire recenziei cu speranța că toată lumea știe despre ce este vorba.
- Trebuie prezentate clar obiectivele reviziei și trebuie acordată integral responsabilitatea completă pentru a face toate aranjamentele necesare unei **persoane desemnate**.
- Aceasta poate fi un job cu normă întreagă pentru câteva zile sau săptămâni.

#### **3.1.1 Planificarea personalului**

- (1) Se includ ca **evaluatori** (revizori):
  - O secțiune transversală a propriilor membri ai proiectului, din toate grupurile proiectului.
  - Reprezentanții conducerii.
  - Invitați evaluatori sau consultanți externi care nu numai că sunt competenți atât în partea tehnică a proiectului, cât și în managementul proiectului, dar care sunt și dezinteresați de proiectul evaluat.
  - Nu trebuie invitați prieteni, în speranța unei evaluări favorabile.
- (2) Trebuie stimulată oportunitatea recenziei de **a descoperi probleme**.
  - Dacă există probleme, acestea vor apărea mai devreme sau mai târziu și, cu cât sunt găsite mai târziu, cu atât vor apărea mai multe dificultăți în remedierea lor.

- De exemplu, o problemă de proiectare poate fi rezolvată în aproximativ o săptămână în timpul fazei de proiectare.
- Aceeași problemă, care nu este surprinsă până când programarea este în curs de desfășurare, poate provoca o mare parte de reproiectare, reprogramare și retestare.
- (3) Trebuie invitat **clientul** la revizie? Metzger sugerează că **nu**.
  - Este posibil să apară mai multe probleme interne în timpul acestei revizii, iar prezența clientului poate fi inhibată.
  - Este mai bine să se rezolve cât mai multe probleme posibil și apoi să fie informat clientul.
  - Aceasta nu este deloc o dublă măsură. De fapt, recenzia internă poate fi tratată ca și o pregătire pentru o evaluare din partea clientului.
  - Intenția nu este aceea de a ascunde ceva clientului ci de a-i prezenta soluții și alternative, nu probleme.
- (4) Se aleg ca **vorbitori de proiect** membrii cei mai competenți în domeniile discutate.
  - Trebuie evitate acele persoane care ar putea fi atât de neglijente încât să treacă peste probleme și să-i liniștească pe ascultători facându-i să credă că lucrurile sunt într-o formă mai bună decât sunt.
  - De cele mai multe ori, dacă o sesiune de revizie începe cu o atitudine de tipul „proiectul-este-în-o formă bună” de regulă ea se termină prost.
- (5) **Cât timp** trebuie alocat reviziei?
  - Asta depinde de:
    - Mărimea și natura proiectului.
    - Complexitatea problemelor tehnice.
    - Dificultățile prefigurate pentru fazele viitoare.
  - Pentru un **proiect de un an**, se pot planifica profitabil **trei până la cinci zile** pentru această revizie.
  - Dacă se anticipatează unele discuții prelungite despre anumite probleme, este necesar ca astfel de situații să fie avute în vedere.

### 3.1.2 Planificarea sălilor de întâlnire

- Întâlnirile trebuie să aibă loc cât mai departe posibil de birou (și telefon).

- Dacă aceasta presupune închirierea unei săli de conferințe de hotel pentru o săptămână, trebuie realizată.
- Este recomandabil să fie planificate pauze frecvente de cafea și, dacă este necesar, mesele de prânz.
  - Trebuie găsit un loc liniștit, cu aer condiționat; este greu să concurezi cu fumul de trabuc sau cu zgomotul citătin.
- În unele cazuri, poate se dorește să se separe oamenii dintr-o sală principală de ședințe pentru a forma „grupuri de lucru” mai mici, pentru a analiza în profunzime o anumită problemă.
  - Dacă da, va trebui aranjat spațiu și pentru astfel de situații.
- Trebuie însă să se reziste tentației de a despărți grupul, cel puțin până când toate prezentările principale sunt finalizate.
- Deoarece această revizuire este menită să arate dacă întregul proiect este sau nu consistent, evaluatorii ar trebui să audă toate prezentările, atât pe cele tehnice, cât și pe cele non-tehnice.

### **3.1.3 Pregătirea suporturilor de prezentare**

- Trebuie stabilit ce suporturi de prezentare (de exemplu, diapositive, prezentări power-point, flip chart, table, tablă electronică, tablă de hârtie) vor fi folosite și trebuie asigurat faptul că pentru fiecare prezentator pentru fiecare prezentator echipamentul adecvat.
- Dacă este desemnat un alt grup să pregătească diagrame și diapositive pentru prezentare, trebuie aflat de cât timp au nevoie pentru aceasta.

### **3.1.4 Pregătirea materialelor necesare**

- Trebuie realizată o selecție foarte atentă a materialelor ce vor fi oferite recenzenților dacă există așteptarea ca ei să le citească.
- Cele două documente pe care ar trebui să nu lipsească cu siguranță sunt [Specificația de proiectare](#) și [Planul proiectului](#). Dincolo de asta, depinde organizatorii.
  - Indiferent însă de ce se distribuie, documentele trebuie să fie curate și lizibile.
  - Documentele trebuie să ajungă la evaluatori cu mult înainte de întâlnirea propriu-zisă.

## **3.2 Obiectivele reviziei fazei de proiectare**

- Obiectivele generale sunt **revizuirea planurilor** și a **proiectării de bază**.
- Figura 5.3.2.a sugerează o schiță a subiectelor de revizuire care ar putea fi luate în considerare.
  - (I) Secțiunea **Context** ar trebui să ofere recenzenților o idee despre mediul în care se lucrează, o idee generală despre problemele tehnice și să schițeze soluția propusă.
  - (II) Sub titlul **Planului proiectului**, se prezintă cel puțin o descriere capsulă a fiecărei secțiuni a planului cu o privire mai detaliată asupra:
    - Planului de faze.
    - Planului de organizare.
    - Planului de testare.
    - Planului de resurse și livrabile.
  - (III) Secțiunea **Proiectarea de bază** ar trebui să descrie la niveluri crescânde de detaliu designul realizat.

Poate fi o idee bună să se ia o pauză de o zi sau cam aşa ceva, odată ce a fost prezentat un prim nivel de detaliu.

    - Acest lucru le va oferi recenzenților șansa de a absorbi ceea ce au auzit, de a căuta în documentul de proiectare și apoi de a reveni mai capabili să audă mai multe detalii.
  - (IV) Prezentarea **Rezumat** ar trebui să dea o imagine sinceră și corectă a punctului în care se apreciază că se află proiectul și a problemelor majore cu care acesta se confruntă.
    - Trebuie făcută o distincție netă între problemele pe care managerul simte că le poate rezolva și problemele care necesită ajutorul conducerii.

## DESIGN PHASE REVIEW OUTLINE

### I. BACKGROUND

- A. The Customer
  - His experience
  - Your prior experience with him
  - His organization
- B. The Job
  - Reason for this project
  - Job environment
  - Overview of requirements
  - Overview of design
- C. The Contract
  - Overall schedule
  - Costs
  - Major constraints

### II. THE PROJECT PLAN

- A. Overview
- B. Phase Plan
- C. Organization Plan
- D. Test Plan
- E. Change Control Plan
- F. Documentation Plan
- G. Training Plan
- H. Review and Reporting Plan
- I. Installation Plan
- J. Resource and Deliverables Plan

### III. THE BASELINE DESIGN

- A. Program Design
- B. File Design

### IV. SUMMARY

- A. Current Status
  - End items delivered
  - End items remaining
  - Confidence
  - Assessment of risk
- B. Problems
  - Technical
  - Management
  - Financial
  - Contractual
  - Legal
  - Personnel
  - Political
  - Customer
  - Other

**Fig. 5.3.2.a.** Schiță a reviziei fazei de proiectare

### 3.3 Rezultate

- Ceea ce se dorește de la această revizie este obținerea permisiunii de a continua cu treaba.

- În majoritatea cazurilor, când se încheie revizuirea, este clar în ce formă se află proiectul.
- În alte cazuri, managerul de proiect sau conducerea companiei va insista ca anumite probleme să fie rezolvate înainte ca faza de programarea să fie demarată.
- Când se planifică revizia, **anumitor participanți** li se cere să fie pregătiți să furnizeze la final în **scris**:
  - (1) Opiniile lor cu privire la stadiul proiectului.
  - (2) Lista lor cu problemele evidente ale proiectului.
  - (3) Orice sugestii pentru rezolvarea acestor probleme.
- Recenzenții trebuie încurajați să evidențieze ceea ce consideră ei că sunt probleme chiar dacă nu au soluții de sugerat pentru rezolvarea lor.
- Se recomandă să se solicite astfel de comentarii scrise de la fiecare evaluator extern și de la membrii selectați ai proiectului.
  - Comentariile lor trebuie transmise conducerii companiei împreună cu recomandările proprii ale managerului de proiect.
- Fiecare revizie de proiect ar trebui să se încheie cu un raport formal scris din partea managerului de proiect.
- La încheierea cu succes a unei revizii, sau după efectuarea modificărilor ca urmare a concluziilor revizuirii, se aprinde lumina verde și începe Faza de Programare.

## **Exercițiul #9**

1. Care sunt obiectivele fazei de proiectare?
2. Care este structura documentului specificației de proiectare? Descrieți secțiunile sale principale.
3. Care sunt regulile de aur ale unui design rezonabil?
4. Ce fel de instrumente de design cunoașteți? Descrieți principalele lor caracteristici și domeniul lor de aplicabilitate.
5. Cum se poate evalua calitatea activității de proiectare?
6. Care sunt principalele activități ale managerului de proiect în faza de proiectare? Detaliați-le.
7. Cum se pregătește activitatea de testare? Care sunt principalele filozofii de testare? Ce fel de niveluri de test cunoașteți?
8. Ce puteți spune despre gestionarea documentației și configurației în faza de proiectare?

9. Care sunt obiectivele și cum trebuie organizată Revizia fazei de proiectare?

## **Capitolul 6. FAZA DE PROGRAMARE**

### **Introducere**

#### **1. Tehnici de programare**

- 1.1 Programare structurată
  - 1.1.1 Obiectivele programării structurate
- 1.2 Programare, proiectare și analiză orientată pe obiecte
  - 1.2.1 Programare orientată pe obiecte
  - 1.2.2 Proiectare orientată pe obiecte
  - 1.2.3 Analiza orientată pe obiecte

#### **2. Modalități de organizare**

- 2.1 Organizarea convențională
  - 2.1.1 Grupul de analiză și proiectare
    - 2.1.1.1 Controlul modificării
    - 2.1.1.2 Controlul datelor
    - 2.1.1.3 Proceduri structurate și inspecții
    - 2.1.1.4 Modelarea prin simulare
    - 2.1.1.5 Documentația utilizatorului
  - 2.1.2 Grupul de programare
    - 2.1.2.1 Proiectare detaliată
    - 2.1.2.2 Codificare
    - 2.1.2.3 Testul modulului
    - 2.1.2.4 Documentație
    - 2.1.2.5 Integrare: „De sus în jos”
    - 2.1.2.6 Integrare: „De jos în sus”
    - 2.1.2.7 Integrare: Specificația Testului
  - 2.1.3 Grup de testare
  - 2.1.4 Grupul de personal
    - 2.1.4.1 Funcțiile personalului tehnic
    - 2.1.4.2 Funcțiile personalului administrativ
  - 2.1.5 Numărul de persoane implicate într-un proiect
- 2.2 Organizarea de tip echipă. Echipa de programatorului șef
  - 2.2.1 Participanți. Mod de lucru
  - 2.2.2 Organizarea proiectului folosind echipa programatorului șef

#### **3. Controlul modificărilor**

- 3.1 Documente de referință
- 3.2 Proceduri de control

#### **4. Instrumente de programare**

- 4.1 Specificații scrise
- 4.2 Executive de testare
- 4.3 Simulatoare de mediu
- 4.4 Medii de programare specializate
- 4.5 Ajutorare automate de documentare
- 4.6 Monitoare software și hardware
- 4.7 Biblioteca de proiect
  - 4.7.1 Biblioteca generală
  - 4.7.2 Biblioteca de sprijin pentru dezvoltare

#### **5. Atribuțiile managerului în faza de programare**

- 5.1 Conducere tehnică
- 5.2 Planificare și control
- 5.3 Comunicarea
- 5.4 Asigurarea condițiilor și instrumentelor de muncă
- 5.5 Atribuirea muncii
  - 5.5.1 Atribuirea muncii pe persoane
  - 5.5.2 Atribuirea muncii pe domenii
  - 5.5.3 Obiectivele atribuirii muncii
- 5.6 Programul de lucru

- 5.6.1 Alocarea orelor normale de lucru
- 5.6.2 Program de lucru suplimentar
- 5.6.3 Tehnica Flexy-Time
- 5.6.4 Tehnica termenului prestabilit (dead-line)
- 5.7 Adăugarea mai multor persoane
- 5.8 Raportarea stării tehnice
  - 5.8.1 Rapoarte scrise
  - 5.8.2 Recenziile orale
- 5.9 Raportarea situației financiare
- 5.10 Instruire
  - 5.10.1 Instruirea personalului tehnic
  - 5.10.2 Formarea managerilor
- 5.11 Evaluare și consiliere
- 5.12 Întreținerea Sanității
- 5.13 Niveluri de management

**Exercițiul #10**

## **6 FAZA DE PROGRAMARE**

- În sfârșit, s-a ajuns la momentul la care se conturează la orizont scrierea de programe și lucrurile încep să se întâmple.
  - Dintr-o dată sunt mai mulți oameni de gestionat;
  - Teancul de documente s-a umflat;
  - Programatorii așteaptă cu nerăbdare să înceapă codificarea;
  - Apar primele defecte apar în designul de bază;
  - Clientul se sprijină pe programatori pentru a modifica cerințele;
  - Contabilitatea spune că bugetul a fost deja depășit;
  - Programatorul Jack este un prost;
  - Programatorul Jill se căsătorește și pleacă;
  - Familiile reclamă supraimplicarea în acest proiect.
- Managerul de proiect va fi recunoscător faptului că a planificat și proiectat totul cu grijă, deoarece începând de la acest moment, va fi preocupat cu rezolvarea problemelor zilnice pe care nicio planificare nu le poate evita.
- Acest capitol se concentrează pe munca de programare și pe cea mai eficientă modalitate de a o duce la bun sfârșit.

### **1 Tehnici de programare**

- În zilele noastre, două tehnici de programare sunt mai folosite:
  - (1) Programare structurată
  - (2) Programare orientată pe obiecte

#### **1.1 Programarea structurată**

- Programarea structurată reprezintă efortul de a stabili ordinea în construirea unui program.
- Există o mare ezitare din partea majorității informaticienilor și programatorilor în definirea programării structurate.
  - Potrivit lui Yourdon, „noțiunea de programare structurată este filozofia scrierii programelor după un set de reguli rigide pentru a reduce problemele de testare, pentru a crește productivitatea și pentru a crește lizibilitatea programului rezultat”

- Hughes și Michton precizează: „..... am putea spune că programarea structurată este proiectarea, scrierea și testarea unui program în baza unui model de organizare prescris”
- Harlan Mills spune „... esența programării structurate este prezența rigoarei și a structurii în programare...”.
- Ideile care sunt comune tuturor definițiilor de programare structurată sunt următoarele:
  - (1) **Ordine**
  - (2) **Claritate**
  - (3) **Lizibilitate**
  - (4) Toate conducând spre scopul unui **cod fără erori**, care poate fi ușor înțeles și de către alții nu numai de autorul programului.
- Zilele codurilor secrete complicate scrise de snobi sau a codurilor dezordonate scrise de programatori slab pregătiți, se apropiu de sfârșit.
  - Există un impuls puternic în rândul liderilor comunității de programare de a pune ordine în afaceri, astfel încât, mai devreme sau mai târziu, întregul complex al activității de programare se va schimba cu siguranță în bine.
- Trecerea de la vechile moduri la cele noi este, desigur, mai dificilă pentru cei obișnuiți cu vechiul.
  - Setarea noilor programatori pe un drum mai clar, înainte ca aceștia să fi învățat obiceiuri proaste, este relativ ușor de realizat.
- În zilele noastre, programarea structurată este considerată o abordare clasică.

### **1.1.1 Scopurile programării structurate**

- • (1) **Corectitudine**.
  - Nimeni nu vrea să structureze programe pur și simplu pentru a le face frumoase.
    - Ceea ce contează în cele din urmă este ca programele să fie corecte – și să-și îndeplinească perfect funcțiile prescrise.
  - Folosind programarea structurată și conceptele aferente, acum pot fi scrise programe complexe care rulează corect prima dată.
    - Prin practicarea principiilor de programare structurată și a matematicii acesteia, programatorii sunt capabili să scrie programe corecte și să convingă și pe alții că sunt corecte prin logică și rațiune, mai degrabă decât prin încercări și erori.
  - Pentru un programator profesionist care aplică principiile programării structurate, erorile în logica programului ar trebui să fie extrem de rare,

deoarece ele pot fi împiedicate să afecteze programele printr-o acțiune pozitivă și conștientă din partea acestuia.

- Programele nu dobândesc bug-uri (erori) în timpul dezvoltării prin contaminare doar fiind în preajma altor programe cu erori, aşa cum fac virușii cu oamenii.
  - Programele dobândesc bug-uri (erori) numai de la autorii lor.
- (2) **Lizibilitate.**
    - În afacerile ce privesc computerele de astăzi nu se mai acceptă programe care nu pot fi citite și înțelese de alții decât de autorii originali.
    - Programele trebuie să fie lizibile de la început, în integralitatea lor, astfel încât să poată fi inspectate de manageri, supraveghetori și alți programatori care verifică logica sau urmăresc problemele din sistem.
    - Programele trebuie să poată fi citite la final, astfel încât să poată fi modificate și întreținute de alții decât programatorii originali.
  - (3) **Testabilitate.**
    - Rezultă în mod evident că un program lizibil, structurat clar poate fi testat mai ușor (în special de altcineva decât autorul original) decât un program misterios.
  - (4) **Productivitate crescută.**
    - Îmbunătățirile aduse de primele trei obiective (corectitudine, lizibilitate, testabilitate) conduc automat la costuri de programare mai mici, deci la o productivitate crescută a activității de programare.

## 1.2 Programarea, designul și analiza orientate pe obiecte

### 1.2.1 Programarea orientată spre obiecte

- Ce este **programarea orientată pe obiecte** (sau POO, aşa cum se scrie uneori)?
- O posibilă definiție:
  - **Programarea orientată pe obiecte** este o metodă de implementare în care programele sunt organizate ca și colecții cooperative de obiecte, fiecare dintre acestea reprezentând o instanță a unei clase și ale căror clase sunt toate membre ale unei ierarhii de clase unite prin relații de moștenire.
- Există trei părți importante în această definiție. În programare orientată pe obiecte:
  - (1) Se utilizează ca blocuri logice fundamentale („părțile” ierarhiei programului) **obiecte, nu algoritmi**.
  - (2) Fiecare **obiect** este o **instanță a unei clase**.
  - (3) **Clasele** sunt legate între ele prin **relații de moștenire** („este un membru” al ierarhiei).

- Un program poate părea orientat pe obiecte, dar dacă lipsește oricare dintre aceste elemente, nu este un program orientat pe obiecte.
- Mai exact, programarea fără moștenire nu este în mod clar orientată pe obiecte.
  - O numim **programare bazată pe tipuri de date abstracte**.
- Conform aceastei definiții, anumite limbaje de programare sunt orientate pe obiecte, iar altele nu.
  - Dintr-o perspectivă teoretică, se poate falsifica programarea orientată pe obiecte în limbaje de programare neorientate pe obiecte.
- **Cardelli și Wegner** spun astfel „că un limbaj este orientat pe obiect dacă și numai dacă îndeplinește următoarele cerințe:

  - (1) Suportă **obiecte** care sunt **abstracții de date** cu o **interfață de operații** numite și o **stare locală** ascunsă.
  - (2) **Obiectele** au un **tip asociat** [clasă].
  - (3) **Tipurile** [clasele] pot moșteni atribute de la **supertipuri** [superclase]"

- Pentru ca un limbaj de programare să susțină moștenirea înseamnă că este posibil să se exprime relația „este un membru” între tipuri.
  - Dacă un limbaj nu oferă suport direct pentru moștenire, atunci el nu este orientat pe obiecte.
- **Cardelli și Wegner** disting astfel de limbaje numindu-le mai degrabă **limbaje bazate pe obiecte** decât **limbaje orientate pe obiecte**. Conform acestei definiții:
  - **Limbaje orientate pe obiecte**: Smalltalk, Object Pascal, C++, Eiffel, CLOS, Java, C#.
  - **Limbaje bazat pe obiecte**: Pascal, Ada.
- Cu toate acestea, deoarece obiectele și clasele sunt elemente ale ambelor tipuri de limbaje, este posibil și foarte de dorit să se folosească metode de proiectare orientate pe obiecte atât pentru limbaje de programare bazate pe obiecte, cât și pentru limbaje de programare orientate pe obiecte.

## 1.2.2 Designul orientat spre obiecte

- Accentul în **metodele de programare** se pune în primul rând pe utilizarea corectă și eficientă a anumitor **mecanisme de limbaj**.
- Prin contrast, **metodele de proiectare** pun accent pe **structurarea corectă și eficientă** a unui sistem complex.
- Ce este de fapt **designul orientat pe obiecte**?
- O posibilă **definiție**:

- **Proiectarea orientată pe obiecte** este o metodă de proiectare care cuprinde procesul de descompunere orientată pe obiecte și o *notație pentru reprezentarea atât a modelelor logice și fizice*, cât și *a modelelor statice și dinamice* ale sistemului proiectat.
- Există două părți importante ale acestei definiții. Designul orientat pe obiecte:
  - (1) Conduce la o **descompunere orientată pe obiecte**.
  - (2) Folosește notații diferite pentru a exprima diferite modele:
    - Modelul logic (structura de clase și obiecte).
    - Proiectarea fizică a unui sistem (module și arhitectura procesului).
    - Aspecte statice ale sistemului.
    - Aspecte dinamice ale sistemului.
- Suportul pentru **descompunerea orientată pe obiecte** este cel care face **proiectarea orientată pe obiecte** destul de **diferită** de proiectarea **structurată**.
  - **Designul orientat pe obiecte** folosește concepții abstracte de **clăsă** și **obiect** pentru a structura logic sistemele.
  - **Designul structurat** utilizează abstracții **algoritmice**.
- Termenul de **proiectare orientată pe obiecte** este folosit pentru a se referi la orice metodă care duce la o **descompunere orientată pe obiecte**.
  - Ocazional, acronimul OOD (Object Oriented Design) este folosit pentru a desemna o anumită metodă de proiectare orientată pe obiecte.

### 1.2.3 Analiza orientată pe obiecte

- **Analiza orientată pe obiecte** (sau **OOA**, așa cum se numește uneori) subliniază construirea de modele din lumea reală, folosind o viziune orientată pe obiect a lumii:
  - Analiza orientată pe obiecte este o metodă de analiză care examinează cerințele din perspectiva claselor și obiectelor identificate în vocabularul domeniului problemei.
- Cum sunt legate **OOA**, **OOD** și **OOP**?
  - Practic, produsele **analizei orientate pe obiecte** pot servi drept **modele** de la care se poate porni un **design orientat pe obiecte**.
  - **Produsele proiectării orientate pe obiecte** pot fi apoi folosite ca **planuri (blueprints)** pentru implementarea completă a unui sistem folosind metode de **programare orientată pe obiecte**.

## 2 Modalități de organizare

- Există o serie de modalități de bază pentru a organiza oamenii cu scopul de a realiza un job:

- (1) Organizarea funcțională.
- (2) Organizarea job-shop.
- (3) Organizarea de tip proiect.
- (1) **Organizare funcțională**
  - Managerul de proiect împrumută oameni din grupuri de specialiști din cadrul companiei.
  - Fiecare specialist este împrumutat către managerul de proiect pentru a-și face partea lui din treabă, iar apoi pleacă fiind probabil împrumutat următorului manager care are nevoie de abilitățile sale.
    - Acest aranjament îi oferă managerului de proiect un control redus, deoarece omul împrumutat este mai probabil să fie mai preocupat de organizația sa de acasă decât de proiectul tău.
    - În mod obișnuit, managerul de proiect are puține sau deloc de spus despre cel pe care îl primește și poate fi frustrat de înlocuirile făcute înainte de finalizarea muncii.
  - Poate mai rău decât atât, este faptul că va exista o continuitate redusă sau deloc a oamenilor la locul de muncă.

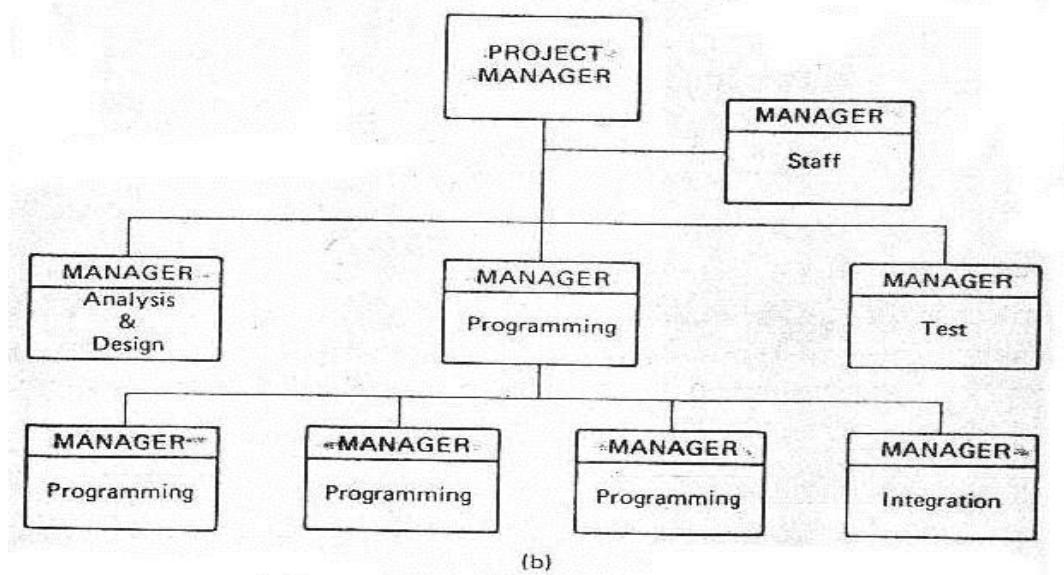
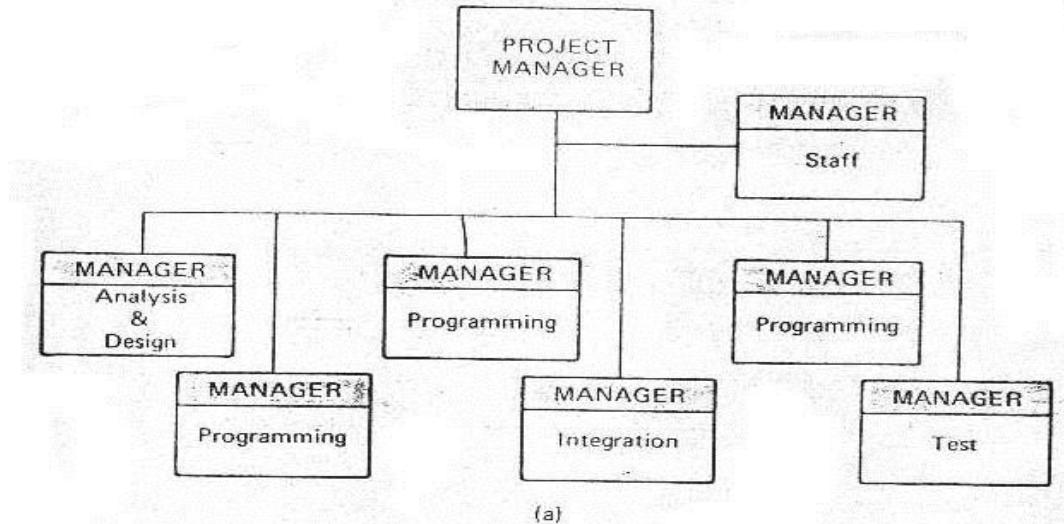
În cel mai rău caz:

  - Analistii vin, analizează și pleacă.
  - Designerii vin, proiectează și pleacă.
  - La fel cu programatorii.
- (2) **Organizarea job-shop (atelier de lucru)**
  - Sistemul de programe este împărțit în mai multe subsisteme majore.
  - Câte un manager și grupul său sunt desemnați cu responsabilitate totală pentru dezvoltarea fiecărui subsistem – analiză, proiectare, programare, implementare.
    - Marea problemă a acestei maniere de lucru este aceea că nimeni nu se preocupă de sistemul ca atare, deoarece managerii sunt preocupăți doar de subsistemele care le-au fost atribuite.
  - Un aranjament job-shop sau atelier de lucru funcționează doar dacă în cadrul proiectului sunt identificate un număr relativ mic de joburi, fără legătură unele cu altele (cu alte cuvinte, proiectul nu este un sistem).
    - Dacă un manager familiarizat cu o organizare de tip atelier este pe cale să gestioneze dezvoltarea unui sistem, acesta trebuie să fie foarte precaut deoarece, cu mare probabilitate, ceea ce a funcționat într-un context s-ar putea să nu funcționeze în altul.
- (3) **Organizarea de tip proiect**

- Nici organizarea funcțională, nici cea de tip atelier nu este adecvată pentru producerea unui sistem.
  - Tipul de organizare necesar aici este **organizarea de tip proiect**.
- Ceea ce este implicat într-un astfel de aranjament este:
  - (1) **Persoanele implicate** își dedică eforturile **unui singur proiect**.
  - (2) Toate sunt **sub controlul unui singur manager de proiect**.
- Organizarea de tip proiect poate lua mai multe forme.
  - În general, fiecare companie are propriile reguli despre **liniile de autoritate, gradul de autonomie, raportarea către managementul extern** și așa mai departe.
  - Ignorând astfel de considerații, referitor la **organizarea de tip proiect**, se pot lua în discuție două abordări destul de diferite:
    - (1) **Organizare convențională**
    - (2) **Organizarea de tip echipă**

## 2.1 Organizarea convențională

- Există două maniere de organizare convențională a unui proiect de dimensiune medie ilustrate în Figura 6.2.1.a.



**Fig. 6.2.1.a.** Modalități de organizare convențională a unui proiect

- Singura **diferență** semnificativă între variantele (a) și (b) este **numărul de niveluri de management** interpuse între managerul de proiect și oamenii care fac munca tehnică.
- Alegerea lui (a) sau (b) depinde de **punctele forte și punctele slabe ale managerului de proiect** și de cele ale **managerilor** care îi stau la dispoziție.
- (1) Dacă **managerul de proiect** este **puternic din punct de vedere tehnic**, capabil să absoarbă multe detalii și să poată gestiona până la **șapte manageri** care îi raportează (un număr oarecare), atunci alegerea ar putea fi varianta (a).
  - În acest caz, pericolul este ca managerul de proiect să devină cufundat în detalii, să piardă din vedere obiectivele mai largi ale proiectului și să piardă în cele din urmă controlul.

- (2) Dacă **managerul de proiect** preferă să delege mai multă responsabilitate pentru a se putea concentra asupra problemelor importante care apar, atunci alegerea ar putea fi varianta (b).
  - În acest caz, managerul de proiect are patru manageri care îi raportează.
- În orice caz, proiectul are mulți manageri (șapte sau opt în afară de managerul de proiect), iar asta poate îngrozi conducerea - „Unde sunt muncitorii!?”.
  - (1) De fapt, acești manageri nu sunt manipulatori de documente.
  - (2) Deoarece sunt foarte implicați în deciziile tehnice, raportul dintre manageri și muncitori nu este atât de rău pe cât pare.
- În cele mai multe situații este recomandat să se aleagă varianta (b) mai degrabă decât (a).
- Importanța reală, însă, nu constă în numărul exact de casete care apar pe organigramă și nici în titlurile atribuite acestora, ci:
  - (1) Managerul de proiect trebuie să răspundă de toate lucrările care trebuesc făcute și să le îndeplinească într-un mod viabil.
  - (2) MP (managerul de proiect) trebuie să se asigure că fiecare membru al organizației cunoaște atât obiectivele sale personale, cât și pe cele ale celorlalți.
- Restul secțiunii descrie funcțiile diferitelor grupuri prezentate în Figura 6.2.1.a. (b) și se încheie prin a comenta numerele tipice de oameni implicați în diferitele roluri.

### **2.1.1 Grupul de analiză și design (A&D Group)**

- Suntem acum în **faza de programare** și, prin urmare, programatorii sunt aceia care sunt în centrul atenției.
- Cu toate acestea, **analiștii și designerii** joacă încă un rol de sprijin foarte puternic.
- Un subset al analiștilor și designerilor originali au următoarele sarcini de îndeplinit:
  - (1) **Controlul modificărilor.**
  - (2) **Controlul datelor.**
  - (3) **Parcurgeri structurate și inspecții.**
  - (4) **Modelare prin simulare.**
  - (5) **Întocmirea documentației utilizatorului.**

#### **2.1.1.1 Controlul modificărilor**

- Funcția cea mai importantă a Grupului de analiză și proiectare în faza de programare este aceea de a efectua procedurile de control al schimbărilor care vor fi descrise mai târziu.
- Acest lucru înseamnă:
  - (1) **Investigarea modificărilor propuse**.
  - (2) **Recomandarea adoptării sau respingerii**.
  - (3) **Documentarea rezultatelor**.
- Grupul acționează ca un filtru.
  - Eliberează pe ceilalți membri ai proiectului, în special pe programatorii, în mare parte de sarcina de a explora o modificare propusă și de a urmări consecințele efectuării schimbării.
  - În cele mai multe proiecte, investigarea unei propunerii de modificare revine programatorului.
    - În acest fel, programatorul este în mod constant distraș de la activitatea lui principală pentru a analiza sau căuta idee sugerată de client sau de cineva din propria organizație.
    - Când o persoană face ceva orientat pe logică precum programarea, activitate care necesită o concentrare maximă, fiecare întrerupere înseamnă o pierdere a eficienței.
      - Când întreruperea se termină, programatorul se întreabă: „Acum, unde am fost?”
      - În plus față de timpul pierdut înapoi, totul s-ar putea termina cu o eroare în punctul de întrerupere.
      - Foarte des frustrarea întreruperii constante îl determină pe programator să dea un răspuns grăbit și să fie de acord cu schimbarea doar pentru a scăpa de problemă, astfel încât să poată continua cu programarea.
  - În concluzie, **un grup** care se ocupă de **propunerile de modificare** și care concentrează o funcție vitală într-un singur loc, mai degrabă decât să o răspândească în întregul proiect, este absolut necesar într-un proiect de anvergură.

### **2.1.1.2 Control datelor**

- Grupul de analiză și proiectare este, de asemenea, implicat în controlul datelor.
  - Aceasta este în realitate o parte a funcției de control al modificărilor, dar necesită o accent suplimentar.
- **Controlul datelor** înseamnă supravegherea tuturor fișierelor sistem, astfel încât structurile acestora să nu fie viciate.
  - Prin **fișiere sistem** se înțeleg acele **organizări de date** care sunt accesate (fie stocate sau preluate din) de mai mult de un singur modul de program.

- Drept urmare, ca parte a designului de bază trebuieesc precizate clar următoarele:
  - (1) Structurile fișierelor sistem.
  - (2) Un dicționar care definește fiecare element de date.
  - (3) Toate regulile de utilizare a fișierelor sistem.
- În multe sisteme de programe, fișierele sistem sunt acelea care **țin sistemul împreună**.
- Așa cum este necesar să se **controlze modificările aduse logicii programului** după ce designul de bază a fost stabilit, tot așa trebuie să fie **controlate și modificările aduse fișierelor sistem**.
  - Programatorii nu trebuie lăsați să încheie acorduri ad-hoc între ei pe măsură ce își desfășoară activitatea.

### 2.1.1.3 Parcurgeri structurate și inspecții

- Grupul de analiză și proiectare este un loc bun pentru a i se atribui responsabilitatea pentru programarea și efectuarea de revizii detaliate continue ale progresului tehnic.
- Există două mijloace strâns legate de efectuarea unor astfel de analize:
  - (1) **Parcurgeri structurate**.
  - (2) **Inspecții**.
- Ambii termeni au intrat în uz în anii 1970; unii fac o distincție între cele două, alții folosesc termenii în mod interschimbabil.
- O **parcuregere structurată** este pur și simplu o revizie organizată (structurată) a muncii unui membru al proiectului de către alți membri ai proiectului.
- În timpul unui **parcurs**:
  - (1) Dezvoltatorul (persoana a cărei lucrare este revizuită) oferă mai întâi o descriere tutorială a proiectului său.
    - Aceasta poate fi un design, un cod, un set de documente, un plan de testare, un artefact sau orice alt element.
  - (2) Apoi membrii proiectului „**parcurg**” verbal produsul, pas cu pas, oferind recenzenților un „**tur ghidat**” și invitându-i să găsească **defecți**.
  - (3) **Vehiculul** folosit de dezvoltator poate fi orice este potrivit pentru produsul său.
    - Dacă este implicat un **design**, atunci **diagramele HIPO** sau alte documente de proiectare pot fi pertinente.
    - Dacă este implicat un **modul de cod**, atunci poate fi folosit **codul real** sau un pseudocod sau ceva similar dacă codul real nu a fost încă scris.

- În cazul în care se revizuește un **plan de testare** sau un **manual de utilizator**, atunci poate fi parcursă fie o schiță detaliată, fie o versiune nefinalizată a acestor documente.
- (4) **Ideea** este ca pentru **fiecare produs parcurs**, trebuie să rezulte **o privire de amănunt**, bine **conturată și foarte specifică** pentru produsul în cauză.
  - Acest proces este mai eficient și mai util, în loc de a face apel spre exemplu, la procesul de testare pentru a identifica probleme sau pur și simplu de a trimite documente unor membri ai echipei în speranța că aceștia le vor revizui și comenta.
- (5) Există, în general, **4-6 participanți** la o parcurgere structurată.
  - Unul dintre ei este întotdeauna moderatorul.
- (6) Se recomandă ca **Grupul de analiză și proiectare** să fie responsabil pentru programarea și desfășurarea demersurilor.
  - În consecință, un membru al aceluia grup ar trebui să acționeze ca moderator.
- (7) **Moderatorul** are ca principale sarcini:
  - (a) Programează întâlnirile și locurile de întâlnire.
  - (b) Ajută la selectarea participanților.
  - (c) Raportează rezultatele imediat după întâlnire.
  - (d) Urmărește pentru a vedea că orice reluare de făcut este făcută și prezentată din nou dacă este necesar.
  - (e) Dar cel mai important lucru, moderatorul trebuie să mențină sesiunile de parcurgere pe direcția obiectivelor lor, fără a fi deturnate și fără a permite ca animozitățile sau ego-urile unor participanți să denatureze eficacitatea reviziei.
- (8) Printre **ceilalți participanți** la parcurgere se includ:
  - Dezvoltatorul a cărui activitate este revizuită.
  - Alți doi până la patru participanți care sunt suficient de competenți pentru a înțelege activitatea analizată și locul ei în sistem.
  - Dacă lucrarea care este revizuită este un modul de cod, unul dintre participanți ar putea fi un programator responsabil pentru cod similar sau un cod care are o interfață directă cu acest modul; altul ar putea fi un programator responsabil de cod dintr-o altă parte a sistemului, spre exemplu, din programul de control.
  - Dacă modulul a fost proiectat de altcineva decât codificatorul, proiectantul original ar trebui să fie și el prezent.
  - În general, alcătuirea grupurilor de revizie are un grad ridicat de flexibilitate.

- (9) Moderatorul trebuie să selecteze cu atenție **recenzenții**.
  - În cele mai multe cazuri, managerii nu trebuie inclusi în demersuri.
  - Aceste sesiuni nu sunt destinate ca vehicule de evaluare a angajaților; prezența unui manager ar pune inevitabil un obstacol enorm asupra procedurilor.
- (10) Scopul esențial al parcurgerii structurate este acela de **a găsi erori, nu de a le corecta**.
  - Corectările sunt presupuse a fi în domeniul și capacitatele dezvoltatorului.
- (11) O sesiune de revizie poate dura între **cincisprezece minute până la două ore**.
  - Dacă este nevoie de mai mult de două ore, se poate programa o a doua ședință după o pauză corespunzătoare (probabil mai târziu în aceeași zi, astfel încât continuitatea să nu se piardă).
- (12) Există câteva **beneficii** extrem de importante ca urmare a efectuării unor parcurgeri serioase și frecvente a tuturor produselor proiectului:
  - (a) În cazul în care produsul este **un design** sau **un cod real**, există o **economie demonstrabilă și semnificativă** atunci când erorile sunt găsite din timp.
    - Cu cât se găsește o **eroare mai târziu** în viața unui proiect, cu atât **costul remedierii erorii este mai mare**.
    - S-ar putea să fie nevoie de o mare parte de **teste de regresie** costisitoare și consumatoare de timp pentru a se asigura că efectuarea unei modificări pentru a remedia o eroare încorporată adânc în sistem nu va afecta negativ alt cod deja testat și presupus curat.
  - (b) Poate există un **beneficiu** enorm în promovarea a ceea ce se numește „**programare fără ego**” sau orice lucru fără ego, de altfel.
    - Într-o carte excelentă, „**Psychology of Computer Programming**” **Gerald Weinberg** susține un argument puternic pentru luarea de măsuri pentru ca programatorul să fie mai puțin defensiv cu privire la erorile din munca sa, promovând ideea ca programatorii să citească codul celorlalți programatori pentru a găsi probleme.
    - El citează dovezi că un număr mare de erori sunt descoperite devreme atunci când se practică citirea codului.
    - Tehnicile specifice utilizate în acest scop sunt cunoscute sub denumirea de **programare peer** sau **peer review**.
  - (c) Avantajele depășesc cu mult astfel de situații.

- Citirea extensivă și regulată a codului oferă o oportunitate frumoasă de a ajuta la **formarea unor oameni noi**.
- Procesul favorizează un **sentiment de deschidere** asupra proiectului, în contrast direct cu situația în care un programator tratează un modul de cod ca pe propria sa proprietate privată.
- Este fără îndoială o **garanție** pentru **calitatea procesului**.
- (d) Când produsul este un document, să zicem un plan de testare sau un manual de utilizare, economiile sunt realizate nu numai prin evitarea erorilor din acele documente care ar putea afecta testarea sau utilizarea sistemului, ci și prin reducerea republicării, a erelor și a costurilor de distribuție.
- (e) Parcurgerile frecvente și productive, odată ce devin un mod acceptat de viață a proiectului, duc în primul rând la **produse mai bune**, deoarece dezvoltatorii nu vor trimite cu bună știință lucrări neglijente știind că vor fi supuse unui astfel de control.
  - Este foarte obișnuit ca un programator sau un scriitor să organizeze un prim draft „rapid și murdar” al unui program sau al unui document, cu intenția de a-l „curăța” mai târziu. Dar adesea, acel mai târziu nu vine niciodată.
  - Parcurgerile structurate pot contribui în mare măsură la eliminarea unor astfel de obiceiuri neglijente și periculoase.
- (f) Există și un **beneficiu educațional** enorm ca rezultat al parcurgerilor.
  - Devine imposibil ca indivizii să lucreze perioade lungi de timp izolați de alți membri ai proiectului, munca lor fiind ascunsă controlului.
  - În general, toată lumea știe ce fac ceilalți.
- Termenul „**inspecție**” este preferat de unii față de „**parcure structurată**” pentru a desemna o activitate similară, dar mult mai riguroasă.
- **Inspețiile** sunt examinări mai intensive ale proiectării detaliate și ale codului, cu mult mai mult accent pe realizarea unor **statistici** privind tipurile de erori găsite pentru a ajuta la ghidarea inspecțiilor ulterioare.
- Rigoarea și evidența atentă a inspecțiilor devin importante pe măsură ce proiectele cresc și pierderea controlului devine o problemă.

#### **2.1.1.4 Modelare, simulare**

- Grupul de analiză și proiectare este responsabil pentru continuarea activităților de modelare și simulare începute în fazele anterioare.
- Efectuează rulări de simulare și evaluează și distribuie rezultatele.
- Poate propune modificări de proiectare ca urmare a unor simulări.
- La un proiect foarte mare în care se realizează multe simulări, poate fi necesar să se formeze un **grup separat** de **modelare prin simulare**.

### **2.1.1.5 Documentația utilizator**

- Există două categorii majore de **documentație**:
  - (1) **Documentația utilizator** include tot ceea ce trebuie precizat pentru a ajuta clientul să utilizeze sistemul.
    - Aceasta este **responsabilitatea grupului A&D**.
  - (2) **Documentația descriptivă** care descrie cum este alcătuit sistemul.
    - Aceasta este treaba **programatorilor**.
- **Documentația utilizator** poate include următoarele subiecte:
  - (1) Instalarea sistemului.
  - (2) Testarea periodică a sistemului după instalare.
  - (3) Proceduri zilnice de pornire.
  - (4) Proceduri zilnice de operare, opțiuni și corectarea erorilor.
  - (5) Pregătirea intrărilor pentru sistem.
  - (6) Analiza ieșirilor din sistem.
    - Aceasta este o activitate care necesită multă asistență din partea programatorilor, dar care trebuie să fie responsabilitatea analiștilor și designerilor, deoarece probabil că au o mai bună înțelegere a punctului de vedere al clientului.
    - La unele proiecte, **utilizatorul** scrie aceste documente cu ajutorul grupului de Analiză și Proiectare.

### **2.1.2 Grupul de programare**

- Programatorii sunt punctul focal al organizării în această fază.
- Munca lor poate fi considerată ca o serie de cinci pași care includ:
  - (1) Proiectare detaliată.
  - (2) Codificare.
  - (3) Testul modul.
  - (4) Documentație.
  - (5) Integrare.
- Programatorul individual este responsabil pentru primii patru pași și cel puțin asistă la al cincilea.

#### **2.1.2.1 Proiectarea detaliată**

- Programatorii moștenesc de la proiectanți documentul numit **Specificația de proiectare**.
- Aceasta este baza pentru întreaga lor activitate.
  - Programele pe care le scriu trebuie să se integreze perfect cu designul de bază; în caz contrar, fie programul, fie designul de bază trebuie să se schimbe.
  - Programatorului individual îi se atribuie o parte din proiectul de bază de către managerul sau supervisorul său. Presupunând că acesta este un singur modul:
    - Prima lui sarcină este să proiecteze modulul în detaliu, respectând toate regulile stabilite în **Specificația de proiectare**.
    - Vehiculul pe care programatorul îl are la dispoziție pentru exprimarea acestui **design detaliat** este documentul numit **Specificația de codificare** (descriș mai târziu).
    - Se așteaptă ca programatorul să elaboreze **cel mai bun design detaliat posibil**, în concordanță cu designul de bază.
    - Încălcarea modelului de referință este o infracțiune capitală. Nu se acceptă sub nici o formă!
- Apare însă o problemă:
  - Unii programatori nu au utilizează documente detaliate de proiectare. Ei preferă să codifice direct din designul de bază și să omită designul detaliat.
  - Alții programatori preferă să codifice mai întâi și să proiecteze mai târziu.
- Ce este de făcut?
  - În primul rând, este o presupunere corectă că într-o zi cineva va trebui să modifice sistemul de programe.
  - Pentru a face acest lucru, persoana respectivă va trebui să o înțeleagă și va solicita o documentație detaliată.
  - Cu excepția cazului în care una dintre aceste două ipoteze este falsă, cu siguranță va fi absolută nevoie de **Specificația de codificare**.
- Următoarea întrebare este: când trebuie produsă specificația de codare?
  - Trebuie făcute înainte sau după codare?
  - În mod clar, dacă singura preocupare este aceea că clientul ar putea modifica ulterior programele, răspunsul este **după codificare**.
- De fapt, tot ceea ce îi pasă clientului este ca documentele detaliate de proiectare să fie livrate împreună cu programele.
  - De fapt clientului nu-i pasă dacă documentele sunt scrise înainte sau după codificare, atâtă timp cât ele sunt exacte.

- Dacă clientului nu îi pasă, cui îi pasă? **Managerul de proiect** are nevoie de **documentație înainte de codificare** din următoarele **motive**:
  - (1) Specificația de codificare este singurul vehicul care poate fi utilizat în **revizuirea activității programatorului** înainte de a ajunge prea departe în codificare și testare.
  - (2) Este singurul document rezonabil de utilizat în continuarea **revizuirii designului**.
  - (3) Redactarea unui document de planificare detaliat forțează un **produs mai bun**.
  - (4) Dacă din orice motiv un programator părăsește proiectul, managementul va fi mult mai confortabil cu o specificație de codare decentă decât cu un modul de program semicodat, nedocumentat.
- În ciuda acestor argumente, se poate permite ocazional ca activitatea de codificare să se facă direct din proiectarea liniei de bază.
  - Este posibil ca unele părți din proiectul de bază să fi fost realizate cu suficient detaliu pentru a permite acest lucru.
- Ca orice altceva din proiect, dacă se merge pe acest principiu, el trebuie să fie rezultatul unei **decizii motivate**.
  - Nu trebuie trecut neobservat și lăsat să se întâmpile.
- Ca **alternativă**, **specificația de codificare** poate fi **înlocuită cu codul însuși**, inserând **comentarii substanțiale** în interiorul codului.
  - (1) Fiecare modul, procedură, funcție, obiect etc. poate fi precedat de o descriere a funcționalităților, intrărilor, ieșirilor, descrierea structurilor de date, relația cu alte module.
  - (2) Codul în sine poate fi bine comentat, inclusiv descrierea filosofiei de codificare.
  - (3) Pentru programatori este mai ușor și în același timp foarte potrivit să-și explice propriile decizii de codare chiar în interiorul codului.
  - (4) Există **instrumente specializate** care pot deriva în mod automat din comentarii documentația codului.

### **2.1.2.2 Codificarea**

- **Codarea** este **traducerea designului detaliat** în **instructiuni de calculator**.
- Pe măsură ce procesul de codificare continuă, **modificările** ale designului detaliat vor fi adesea considerate recomandabile sau necesare.
  - Efectuarea acestor modificări este responsabilitatea programatorului, cu excepția cazului în care proiectarea liniei de bază este afectată.
- Un manager ar trebui să urmărească programatorii care au tendința de a scrie cod inutil de condensat și de complex.

- Deși vor exista momente când va fi necesar să se economisească fiecare bit și fiecare microsecundă posibilă, de obicei există considerații mult mai importante.
- **Codul** trebuie să poată fi citit de un alt **programator competent**.
  - Metzger citează mai multe cazuri când un programator pseudoprofesionist a lăsat în urmă un cod care a funcționat, dar era de neînțeles pentru oricine altcineva în afară de programator.
    - *Într-un caz, programatorul a lăsat un program major minunat de eficient, dar într-o zi a devenit necesară modificarea acelui program. Managerul nebănuind vreo problemă i-a promis clientului că modificările vor fi făcute și livrate în patru săptămâni. Șase luni mai târziu, treaba nu era încă terminată, iar managerul jenat a trebuit în sfârșit să rescrie programul de la zero. Un exemplu extrem? Deloc. Trebuie avut mare grijă în astfel de cazuri.*
- În programare, **simplitatea** dă roade.
- *Dacă vrei să-ți provoci programatorii, provoacă-i să scrie cod eficient pe care chiar și tu, managerul de proiect îl poți înțelege.*

### **2.1.2.3 Testarea modul**

- **Testarea modul** este procesul de **testare a unui modul individual** într-un mediu izolat **înainte** de a-l combina cu alte module testate.
- Termenul **modul** este echivalent în acest context cu: **unități software** (SU), **componente software CSCI** (Elemente de configurare software pentru computer) sau alți termeni specifici.
- În mod obișnuit, unui programator individual i se atribuie ceea ce a fost denumit mai devreme „**unitate**”, adică modulul de cel mai jos nivel din sistem. În consecință ar putea urma, deci, ca această activitate să fie denumită „**testarea unității** (unit test)”.
  - Adesea, totuși, modulul de cel mai jos nivel dintr-o ierarhie ar putea fi situat la un nivel mai înalt decât cel de „unitate”, probabil ceea ce s-a numit „componentă”.
  - În acele cazuri, termenul „**testarea unității**” ar fi incorect.
  - Deci, folosirea „**testare modul**” presupune o vizinătate mai generală.
- **Obiectivul testării** este de a determina dacă, acest modul, atunci când este introdus în sistem, își va face treaba ca o cutie neagră:
  - Cu alte cuvinte, ar trebui să fie capabil să accepte intrările specificate și să producă exact ieșirile potrivite.
- Deși proiectul poate să beneficieze de diverse ajutoare de testare, **testarea modulelor** este **treaba programatorilor**.

- Nu este recomandat să se impună vreo schemă rigidă, formală de testare a modulelor, doar câteva linii directoare generale.
  - (1) Programatorul trebuie să pună pe hârtie, în cuvinte proprii și în format propriu, pașii pe care își propune să îl execute pentru a testa modulul.
  - (2) El ar trebui să discute acest „plan de testare al modulului” informal cu managerul său, să îl supună unei verificări, să îl modifice dacă este necesar și să îl execute.
  - (3) De regulă, testarea modulelor nu implică mai mult decât o verificare minuțioasă și o compilare curată.
  - (4) Un pas mai departe ar fi acela de a „parurge structurat” codul cu un alt programator.
- Multe module vor trebui testate în continuare într-o manieră „autonomă”, de sine stătătoare, adică nu sunt încă combinate cu alte module de sistem.
  - Acest lucru poate fi realizat prin furnizarea de date de testare și de drivere de testare.
  - Driverele de testare sunt programe al căror scop este acela de a furniza un mediu de testare special pentru modul.
- Deciziile privind natura și amploarea testului modul sunt influențate de maniera de testare de integrare adoptată pentru proiect: de sus în jos sau de jos în sus.
- (1) În testarea de jos în sus, sunt necesare aşa numitele drivere de testare de diferite tipuri, care sunt utilizate pentru a reprezenta „partea de sus” a sistemului.
  - Un driver de testare este partea sistemului programului de deasupra modulului testat în ierarhie și responsabil pentru invocarea modulului.
- (2) În testarea de sus în jos, „partea de sus” a sistemului există deja.
  - Modulul poate fi adăugat la sistemul existent.
  - Ceea ce trebuie simulat în acest caz este orice modul relevant situat sub cel testat.
  - În acest caz, programatorul scrie cod numit „stub-uri (modul surrogat)” pentru a înlocui modulele inferioare lipsă.
    - Stub-urile (modulele surrogat) sunt în general mai simple decât driverele sau executivele de testare. Ele pot pur și simplu să înregistreze că au fost apelate și să returneze controlul la modulul de invocare.
    - Stub-urile pot merge mai departe și pot simula acțiunile care vor fi eventual întreprinse de modulele reale pe care le înlocuiesc temporar.
  - Cea mai eficientă modalitate de a furniza stub-uri este construirea întregului sistem de stub-uri în avans, mai degrabă decât introducerea lor de către programatori individuali, atunci când sunt necesare.

- Pe măsură ce module noi sunt terminate și introduse în sistem, stuburile corespunzătoare pot fi șterse.

#### 2.1.2.4 Documentația

- „Documentează pentru alții aşa cum ai vrea ca ei să-ți documenteze ţie!” spun Kreitzberg și Schneiderman.
  - Aici este punctul esențial pentru care un produs de **altfel bun** poate fi **slab reprezentat**.
- Programatorii sunt responsabili pentru **documentele care descriu în detaliu modul în care a fost construit sistemul**.
- Vehiculul pe care programatorii îl folosesc în acest scop este **Specificația de codificare** și, de asemenea, **Planul de documentare**.
- **Specificația de codificare** este același document pe care programatorul l-a folosit pentru a-și arăta designul detaliat înainte ca **modulul să fie codificat** (fig. 6.2.1.2.4.a.)

**SPECIFICAȚIA DE CODIFICARE**  
**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA REALIZĂRII:**

**REALIZAT DE:**

**SECȚIUNEA 1: SCOP**

O declarație standard: *Acest document conține descrierea detaliată a modulului program \_\_\_\_\_.*

**SECȚIUNEA 2: DOCUMENTE APPLICABILE**

O declarație standard care introduce această specificație detaliată în partea corespunzătoare a specificației de proiectare: „*Designul descris în acest document reprezintă partea din proiectul de bază prezentat în specificația de proiectare, numărul documentului \_\_\_\_\_, subsecțiunea \_\_\_\_\_.*”

**SECȚIUNEA 3: DESIGNUL DETALIAT**

### **3.1. Structura programului**

Această secțiune descrie logica modulului de program conform standardelor și convențiilor adoptate și menționate în Specificația de proiectare, subsecțiunea 3.3.

### **3.2. Structura fișierelor**

#### **3.2.1. Fișiere sistem**

Această subsecțiune face referiri explicate la aspectul fișierelor sistem conținute în Specificația de proiectare. Aspectele fișierelor pot fi repetați aici dacă programatorul consideră că acest lucru ar spori claritatea acestui document.

#### **3.2.2. Fișiere locale**

O descriere completă și detaliată a tuturor fișierelor locale. Fișierele locale sunt unice pentru acest modul de program. Ele nu sunt accesate de alte module.

## **SECTION 4: LISTINGURI**

Aceasta este o referință standard la listingurile de instrucțiuni detaliate, produse de mașină, care arată setul complet de coduri obiect pentru acest modul, inclusiv orice fișiere locale.

---

**Fig. 6.2.1.2.4.a. Model de Specificație de codificare**

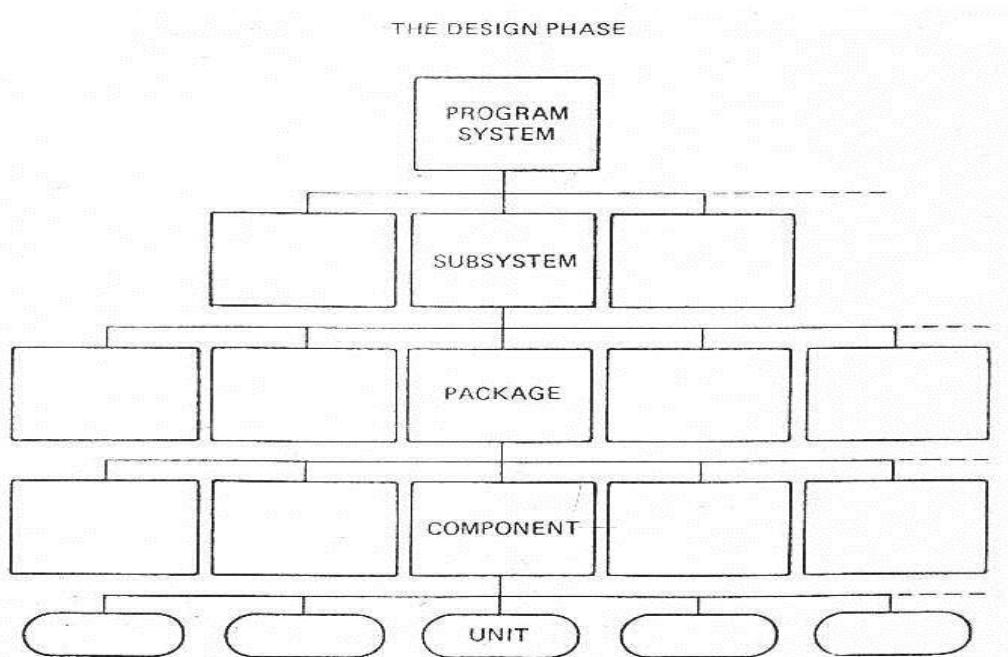
- Când modulul a fost testat, specificația sa de codificare trebuie corectată și completată prin adăugarea listingului modulului codificat real.
  - Astfel, specificația de codificare originală a arătat intenția de proiectare detaliată, la final, documentul completat arată proiectul final împreună cu codul rezultat.
- Logica și codul descrise în Specificația de codificare trebuie să fie exacte și consecvente.
- Țesutul conectiv care leagă **toate specificațiile individuale** de codare împreună este **specificarea de proiectare**.
  - Această combinație ar trebui să descrie complet și adekvat structura sistemului de programe.
- După cum s-a menționat, **specificarea de codificare** poate fi **inclusă** ca și **comentarii în cod**.
  - În acest caz, documentul **Specificarea de codificare** poate lipsi sau poate fi mai formal, inclusiv considerente generale.

## **2.1.2.5 Integrarea de sus în jos**

- **Integrarea**, sau **testarea de integrare**, este procesul de adăugare treptată de noi module la sistemul în evoluție și de testare aferentă pentru a se asigura că noul modul și sistemul funcționează corect.
- Presupunând că s-a ales testarea de integrării de sus în jos ca abordare în cadrul proiectului, în concordanță cu utilizarea designului de sus în jos și a programării de sus în jos. Cum se procedează? Există mai multe căi:
  - (1) Toate **testele de integrare** ar putea fi efectuate de către **un grup separat denumit Echipa de testare a integrării** (vezi Fig. 6.2.1.a) a cărui unică funcție este doar aceasta.
    - Membrii grupului ca atare, nu au scris ei însăși niciun program.
    - Modulele testate individual vor fi predate acestei echipe de testare a integrării.
    - Echipa va adăuga fiecare modul la sistemul de dezvoltare și îl va testa conform unui plan de testare de integrare predeterminat.
  - (2) Toate **testele de integrare** ar putea fi făcute de către **programatorii individuali** (se elimină grupul „**Integrare**” din fig.6.2.1.a.).
    - Fiecare programator este responsabil pentru adăugarea modulelor sale și pentru rularea testelor conform planului de testare.
  - (3) **Testarea de integrare** ar putea fi gestionată de **un grup care are și responsabilități de programare**.
    - În mod logic, acesta ar fi grupul însărcinat cu scrierea modulelor de nivel superior în ierarhia programelor – programul „executiv”, programul „de control” sau orice numim setul de cod care servește drept cadru de sistem.
- **Metzger** sugerează că cea de-a treia alternativă (3) este în general cea mai bună.
  - Această alegere garantează că persoanele responsabile de integrare au cele mai intime cunoștințe despre întregul sistem.
- Alternativa (1) este realizabilă, poate chiar cea mai bună, pentru proiecte mari, în care sunt atât de multe programe implicate încât integrarea este o sarcină uriașă. Dar de reținut:
  - Un grup separat care nu participă la programarea efectivă poate fi prea îndepărtat de sistem.
  - Membrii grupului sunt într-o poziție mai puțin favorabilă pentru a identifica problemele și pentru a găsi soluții.
  - S-ar putea să fie mai puțin motivați, deoarece nu gestionează codul propriu.
  - Un contraargument puternic este acela că, un astfel de grup separat ar putea fi mai obiectiv, tocmai din motivul că propriul cod nu este pus sub semnul întrebării.
- Alternativa (2) invită la haos.

### 2.1.2.6 Integratare de jos în sus

- Pe măsură ce modulele testate devin disponibile de la programatori, începe procesul de integrare.
- Teoretic, aceasta înseamnă:
  - (1) Unitățile sunt combinate și testate împreună pentru a forma componente.
  - (2) Aceste componente sunt grupate și testate pentru a forma pachete.
  - (3) și aşa mai departe în sus în ierarhie (vezi Fig. 6.2.1.2.6.a) până când sistemul complet a fost asamblat și testat progresiv, în manieră exhaustivă.



**Fig. 6.2.1.2.6.a.** Ierarhia sistemului de programe

- În practică însă, se va descoperi că de obicei, indiferent cât de bine au fost planificate lucrurile pe hârtie, procesul nu este chiar atât de curat și de ordonat.
  - (a) Un motiv este acela că unele „componente” vor fi gata în timp ce alte „unități” sunt încă codificate.
  - (b) Un altul este că atunci când apar bug-uri la oricare nivel de test, unitățile cu probleme trebuie trimise înapoi pentru remediere.
- Din aceste considerente, **testarea de integrare** trebuie să fie planificată ca un proces ordonat de construcție care însă permite anumite **ocoluri**.
- Există cel puțin două moduri de a proceda cu **integrarea de jos în sus**.
  - (1) Unul este ca **programatorii** să producă modulele de cel mai jos nivel, adică unitățile, și să le direcționeze spre un **grup separat pentru integrare**.

- (2) Un altul este acela ca grupurile de programare să-și integreze porțiunile din sistem în bucăți mult mai mari pe care să le predea către un grup separat.
- Prima modalitate poate fi teoretic mai atractivă.
- A doua modalitate este mai practică și mult mai satisfăcătoare pentru programatori, deoarece îi conferă programatorului individual mai multă responsabilitate decât să continue să producă piese mici (unități) pentru a le asambla altcineva.
- Ca și în cazul testării de sus în jos, grupul responsabil de integrare ar putea fi responsabil și pentru scrierea programului de control de bază pentru sistem.
- Încă timpul planificării primare a testelor, este necesar să se decidă la ce nivel va fi predatea grupului de integrare munca efectuată de grupurile de programare.
  - De exemplu, se poate acorda grupurilor de programare responsabilitatea pentru proiectarea detaliată, codificare, testarea modulelor, documentare și integrare până la nivelul de **pachete de program** (vezi fig. 6.2.1.2.6.a).
  - Când integrarea unui pachet individual a fost finalizată, **pachetul** este predat grupului de integrare pentru fuziunea finală a pachetelor în subsisteme și a subsistemelor în sistem.
  - Desigur, poate fi ales un alt nivel la care modulele să fie trimise către Grupul de integrare.

### **2.1.2.7 Integrarea: Specificația de testare**

- Deoarece **specificația de testare de integrare** este cheia procesului formal de testare, va fi analizată puțin mai detaliat

---

#### **SPECIFICAȚIA DE TESTARE**

**(DE INTEGRARE, SISTEM, DE ACCEPTANȚĂ, DE SITE)**

**(MODEL)**

---

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA REALIZĂRII:**

**REALIZAT DE:**

---

**SECTIUNEA 1: SCOP**

*Există patru seturi separate de specificații de testare: de integrare, sistem, de acceptanță și de site-ului. Modelele pentru toate cele patru seturi sunt identice, cu excepția faptului că trebuie inserat calificativul corespunzător („integrare”, „sistem”, „acceptare” sau „site”). Conținutul specificațiilor poate varia, desigur, considerabil, deși două dintre ele (acceptarea și site-ul) vor fi adesea identice. Secțiunea de față, Scop, ar trebui să servească în fiecare caz ca o introducere a documentului, descriind intenția acestuia și modul în care urmează să fie utilizat.*

## **SECȚIUNEA 2: DOCUMENTE APLICABILE**

## **SECȚIUNEA 3: GENERALITĂȚI PRIVIND TESTAREA DE (INTEGRARE, SISTEM, ACCEPTANȚĂ, SITE)**

**3.1. Filozofia de testare**

**3.2. Obiective generale**

**3.3. Proceduri generale**

**3.4. Criterii de succes**

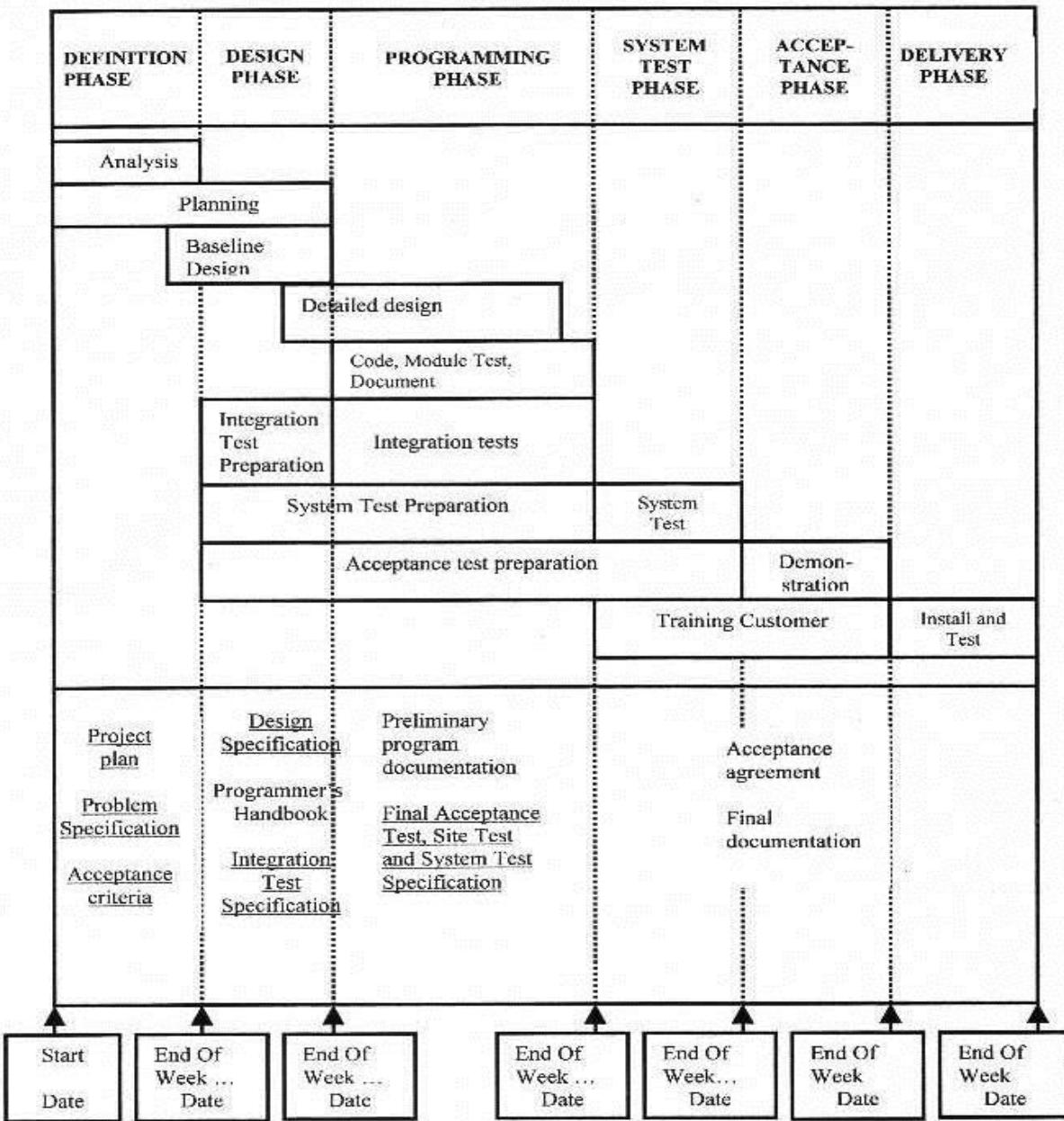
## **SECȚIUNEA 4: MATRICI DE ACOPERIRE**

*Inclusează un diagramă care listează de-a lungul axei verticale zonele care trebuie testate și de-a lungul axei orizontale numărul (numerele) cazului de testare care acoperă fiecare zonă. Când este complet, acest grafic reprezintă o referință încrucișată între toate zonele care trebuie testate și toate cazurile de testare care acoperă acele zone.*

**Fig.6.2.1.2.7.a. Model generic de Specificație de Testare**

- Specificația testării de integrare trebuie să fie gata de utilizare la începutul fazei de programare, când integrarea începe efectiv.
- Documentul trebuie, prin urmare, să fie finalizat în faza de proiectare (vezi fig. 6.2.1.2.7.b.)

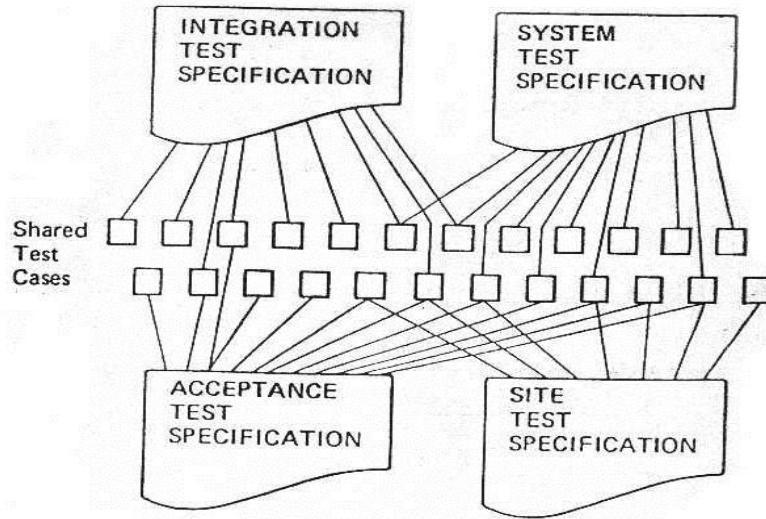
## The Model of Project Life Cycle



**Fig. 6.2.1.2.7.b.** Model de ciclu de viață al unui proiect software

- Specificația testării de integrare descrie:
  - (1) Filosofia testării.
  - (2) Obiective.
  - (3) Proceduri și instrumente generale.
  - (4) Criterii de succes.

- (5) **Matricea de acoperire** - care arată ce teste specifice (sau „cazuri de testare”) acoperă care zone funcționale ale sistemului programului.
- **Specificația testării de integrare** este necesară indiferent dacă se utilizează testarea de sus în jos sau de jos în sus.
- Specificația de testare de integrare necesită un număr de **cazuri de testare**.
  - Un **caz de testare** conține obiectivele detaliate, datele și procedurile necesare pentru un test dat.  
O privire asupra matricei de acoperire menționată mai devreme ar trebui să arate care caz sau cazuri de testare se aplică pentru o anumită zonă funcțională.
- **Elementele cheie** dintr-un **caz de testare** sunt:
  - (1) **Datele** necesare testului.
  - (2) Un **script** (scenariu).
    - Un **script** (numit adesea scenariu) este un set de proceduri care descriu pas cu pas pentru acest test:
      - (a) **Ce** trebuie făcut.
      - (b) **Cine** trebuie să o facă.
      - (c) **Când** trebuie făcut.
      - (d) **Ce** să se caute.
      - (e) **Ce** să se înregistreze.
    - Scripturi similare sunt descrise în capitolul următor în care este discutată testarea sistemului.
- Acoperirea de către o **specificație de testare de bază**, a **cazurilor de testare**, mai degrabă decât realizarea unui document de testare uriaș al întregului sistem, este un alt exemplu de **modularitate**.
  - Este mult mai ușor să se determine locul unde te află atunci când lucrurile sunt făcute în bucăți clar delimitate, curate și finite.
  - Dacă apar probleme, revenirea la repetarea unui test este simplă atunci când se poate indica un singur caz de testare și dispune reluarea lui.
- Fig. 6.2.1.2.7.c. ilustrează patru tipuri de specificații de testare formale care sunt discutate aici și în următoarele două capitole.



**Fig. 6.2.1.2.7.c.** Tipuri de specificații de testare

- Fiecare tip de test are aceeași organizare conceptuală.
  - De fapt, precum se poate vedea din fig.6.2.1.2.7.c., anumite cazuri de testare pot servi la fel de bine în timpul testării de integrare, testării sistem, testării de acceptanță și testării site-ului.
- O bună planificare la începutul proiectului vă va permite să se maximizeze utilizarea multiplă a cazurilor de testare la diferite niveluri de testare.
- Se recomandă ca testele să fie pregătite cu atenție în prealabil, astfel încât, atunci când a sosit timpul de integrare, concentrarea să fie focusată pe rularea testelor, pe evaluarea rezultatelor și pe efectuarea de remedieri.
  - Este prea târziu să se demareze planificarea testelor atunci când testarea începe efectiv.
    - Tot ce poate face atunci, este să se speră și să se roage, că proiectarea de bază și testarea modulelor să fi fost făcute atât de bine încât lucrurile să decurgă conform planificării.

### 2.1.3 Grupul de testare

- În timpul fazei de programare, sarcina Grupului de testare este să se pregătească pentru testarea sistem, pentru testarea de acceptanță și pentru testarea de site (la fața locului).
- Acest grup de testare nu este același grup responsabil pentru testul de integrare.
  - Orientarea sa este destul de diferită.
- Testerii de integrare s-au preocupat de:
  - (1) Asamblarea modulelor de program împreună.

- (2) **Testarea interfețelor.**
- (3) **Testarea sistemului** atât din punctul de vedere al **logicii**, cât și din cel al **funcționalității**.
- **Testerii de sistem și de acceptanță** sunt aproape exclusiv preoccupați de funcțiile de testare.
  - Ei nu sunt direct preoccupați de structura sistemului de programe.
  - Sunt preoccupați mai degrabă de:
    - (1) **Cum funcționează sistemul de programe.**
    - (2) **Cât de bine îndeplinește cerințele** menționate în **Specificația problemei**.
- Grupul de testare intră în evidență în următoarele două capitole, dar trebuie să se pregătească acum, în timpul fazei de programare.
- În timpul **fazei de programare**, munca **grupului de testare** include:
  - (1) **Scrierea specificațiilor de testare.**
  - (2) **Construirea cazurilor de testare** specifice.
  - (3) **Anticiparea** (prezicerea) **rezultatelor**.
  - (4) **Pregătirea** datelor de testare.
  - (5) **Efectuarea** de aranjamente pentru **timpul de calcul**.
  - (6) **Stabilirea** planificării testelor.
  - (7) **Organizarea** bibliotecilor de teste.
  - (8) **Alegerea și asigurarea** instrumentelor de testare.

#### **2.1.4 Grupul de personal de sprijin (staff)**

- Unii privesc grupurile de personal ca pe niște anexe, care de regulă crează dureri de cap.
  - Ocazional, acest punct de vedere este justificat, deoarece unii manageri se întâlnește de atât de mulți asistenți de diferite tipuri, încât este dificil să se determine **cine** este managerul.
- Acest lucru se întâmplă de regulă în **organizațiile mari**, cu foarte mulți angajați, deoarece regulile, reglementările și documentele asociate pot scapa de sub control și sunt angajați “alți angajați” pentru a le controla.
- Astfel, unii angajați creează mai multe **reguli, reglementări și documente**, determinând răspândirea rapidă a birocratiei.
  - Acest lucru se poate întâmpla chiar și în organizațiile mai mici, în special atunci când clientul se întâmplă să fie mare.

- Ce ieșe în evidență referitor la considerentele formulate mai sus referitor la grupurile de personal:
  - După un timp, nimeni nu știe de ce sunt acolo, darămite de ce au fost formați inițial.
  - Un manager ia adesea un membru al personalului pentru a lucra într-o zonă, dar nu definește în mod concret slujba acelei persoane.
    - Rezultatul este că munca lui se poate suprapune cu cea a altor persoane.
    - Această situație este cunoscută ca conceptul de sindrom stuff.
  - Dacă membrul personalului este harnic, el își va defini propriul domeniu de activitate și foarte curând va genera cerințe și necesități pentru mai mult personal.
- Există o singură modalitate de a evita creșterea funcțiilor personalului: Managerul de proiect trebuie să definească atribuțiile (fișa postului) membrilor personalului la fel de clar cum definește slujba unui programator.
  - Cu siguranță un manager de proiect nu ar angaja un programator și i-ar spune să-și găsească o lucrare de programare de făcut. În schimb managerul i-ar spune programatorului:
    - Aceasta este treaba generală,
    - Aceasta este piesa de cod care trebuie realizată,
    - Aceasta este planificarea,
    - Așa trebuie raportat progresul!
  - Același lucru trebuie făcut și cu un membru al personalului.
    - Nu trebuie angajat nimeni decât dacă i se pot atribui responsabilități concrete.
- Cele două tipuri de **funcții ale personalului** de care este posibil să fie nevoie în cadrul unui proiect sunt cele **tehnice și administrative**.

#### **2.1.4.1 Funcțiile personalului tehnic**

- Persoanele care furnizează **suport tehnic** trebuie să fie ei ele însăși competente din punct de vedere tehnic.
- **Funcția lor** este de **a se concentra pe sarcini** care îi ajută pe toți ceilalți oameni tehnici din proiect.
- **Joburile lor specifice** sunt:
  - (1) **Controlul timpului de calcul**. (Dacă situația impune această activitate)

- Toate nevoile de timp de calcul trebuie direcționate către o singură persoană care ar trebui să asigure timpul în fiecare săptămână, să îl programeze cât mai echitabil între cei care îl solicită, să rezolve conflictele, să respecte prioritățile, să țină evidențe exacte ale timpului solicitat și utilizat, să planifice timpul necesar cu săptămâni și luni înainte și să distribue aspirina atunci când timpul este anulat.
- O parte a acestei sarcini este aceea de a **stabili** și de a **aplica regulile de utilizare a timpului de calcul**. Membrul personalului trebuie să scrie **procedurile** (limpede și clar) pentru transmiterea rulărilor de la distanță sau pentru utilizarea „practică” a mașinii, să organizeze preluarea și livrarea testelor și a ieșirilor computerului și să asigure și alte facilități fizice, cum ar fi pubele și dulapuri precum și facilități pentru serviciul de curierat dacă este necesar.
- Pe scurt, această persoană ar trebui să fie **interfață** dintre sistemele de calcul și utilizatorii acestora.
- (2) **Furnizarea de asistență tehnică**.
  - Același membru al personalului trebuie să ofere și asistență tehnică:
    - Se estimează cantitatea de serviciu tehnic necesară și se asigură necondiționat.
    - Trebuie avut grijă de problemele incidente, cum ar fi nevoia de cerințe tehnice speciale.
    - Trebuie stabilite priorități clare ori de câte ori este necesar.
- (3) **Menținerea manualului programatorului**.
  - Organizarea manualului programatorului, distribuirea lui și menținerea lui actualizată ar trebui să fie făcute de personalul tehnic.
- (4) **Instruire**.
  - Cu excepția cazului în care instruirea este o funcție foarte mare pentru proiectul în desfășurare, Grupul de personal trebuie să fie responsabil atât pentru instruirea internă, cât și pentru cea externă și ar trebui să asigure instructori, facilități de instruire, materiale de instruire scrise, programe și estimări ale costurilor de formare.
- (5) **Gestionarea cerințelor și sarcinilor tehnice speciale**.
  - Ocazional, există sarcini tehnice specifice, de scurtă durată de făcut, pentru care **nu** există un o entitate anume căreia să îi fie atribuite.
  - *De exemplu, poate există o problemă supărătoare care afectează mai multe dintre grupurile proiectului și care trebuie urmărită. Este recomandabil să fie inclus cineva neutru în estimările personalului pentru a rezolva problema și pentru a stinge a incendiul*

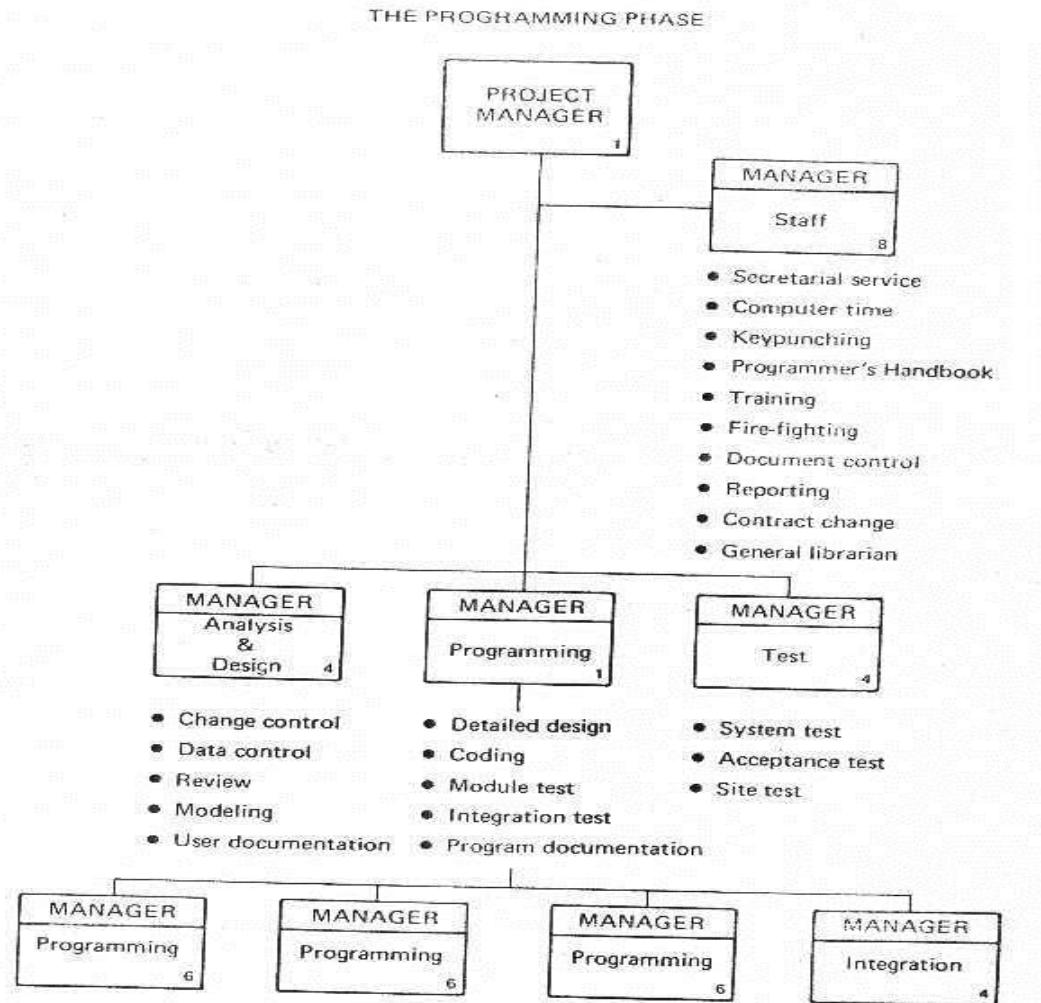
#### **2.1.4.2 Funcțiile personalului administrativ**

- Înainte de a prezenta funcțiile specifice ale personalului administrativ trebuie precizat ce nu este personalul administrativ:
  - (1) **Nu este** managementul de proiect.
    - Este un ajutor pentru managementul de proiect.
  - (2) **Nu este** departamentul de control al calității.
    - Controlul calității este o funcție de management, iar calitatea nu va fi asigurată prin faptul că o mulțime de administratori se uită peste umărul programatorului și completează formulare și rapoarte.
  - (3) **Nu este** nici un grup de management al personalului, nici un grup de administrare a salariilor.
    - Toate acestea sunt joburi de management.
- **Funcțiile personalului administrativ** sunt următoarele:
  - (1) **Controlul documentelor.**
    - Aceasta este o funcție la fel de vitală ca oricare din proiect. Dacă documentația scăpa de sub control, proiectul poate foarte ușor eșua.
    - Personalul administrativ are sarcina de a **gestiona documentația**, așa cum este prevăzut în Planul de documentare.
    - Acest job presupune:
      - (a) Înființarea și operarea Bibliotecii Generale a Proiectului.
      - (b) Gestionarea tuturor interfețelor dintre proiect și orice organizație externă de publicații tehnice.
      - (c) Urmărirea numerelor de documente și emiterea unora noi la cerere.
      - (d) Publicarea sau actualizarea unui index periodic al documentației care să enumere numele și numerele tuturor documentelor proiectului.
      - (e) Asigurarea tuturor serviciilor și echipamentelor de reproducere.
  - (2) **Controlul raportărilor.**
    - (a) Personalul administrativ îl asistă pe managerul de proiect în colectarea datelor de stare și elaborarea rapoartelor de stare ale Managerului de proiect către conducerea companiei și ale managerului de proiect către client.
    - (b) Obține și distribuie către managerul de proiect și către toți ceilalți manageri rapoarte periodice privind situația financiară a proiectului.
    - (c) Personalul pregătește raportul final, și menține istoricul proiectului descris mai devreme.

- (d) Dacă se utilizează în cadrul proiectului metodologia PERT sau alte sisteme automate de raportare și control, personalul administrativ furnizează intrările pentru acestea (din datele obținute de la managerii de linie) și distribuie rezultatele.
- (3) **Controlul modificării contractului.**
  - Atunci când este convenită la nivel tehnic o modificare a contractului, personalul administrativ are sarcina de a completa documentele asigurându-se că clientul este de acord în mod oficial cu modificarea.
  - O parte a jobului se referă la evaluarea costului modificării. În acest sens, personalul va corobora cele patru părți implicate:
    - (1) Oamenii tehnici care fac prima estimare de cost.
    - (2) Managerul de proiect.
    - (3) Serviciile financiare și juridice ale companiei.
    - (4) Clientul.
- (4) **Suport de secretariat și dactilografie pentru proiect.**

## 2.1.5 Numărul de persoane implicate într-un proiect

- Referitor la numărul de persoane implicate într-un proiect de dimensiune medie, se va discuta despre varianta lui Metzger:
- Figura 6.2.1.5.a. este identică cu figura 6.2.1.a (b), dar cu două informații adăugate:
  - (1) Un rezumat al sarcinilor fiecărui grup
  - (2) Numărul de persoane din fiecare grup. Cifrele sunt, desigur, supuse argumentării și vor varia de la un proiect la altul
- Motivația pentru alegerea numerelor de persoane pentru un proiect software de 20-30 de persoane este următoarea:
  - (1) **Managerul de proiect.**
    - Desigur că există „unul”.
    - Uneori sunt două persoane.
      - Al doilea se numește asistent manager de proiect, manager adjunct sau ori care alt termen lipsit de sens.
      - De evitat să existe doi manageri, deoarece este dificil să se știe cine este responsabil pentru ce.



**Fig.6.2.1.5.a. The numbers game**

- (2) **Personalul administrativ**
  - **Un manager și 1-2 lucrători.**
  - Cei 1-2 sunt personal tehnic și administrativ.
  - Membrul administrativ are uneori rolul de secretar.
  - Membrul administrativ, împreună cu managerul grupului, se ocupă de chestiunile contractuale, controlul documentației și pregătirea raportului.
- (3) **Analiză și proiectare.**
  - **2-3 lucrători plus managerul** ar trebui să fie suficienți pentru a se ocupa de funcțiile alocate.
  - Spre sfârșitul fazei de programare, numărul poate fi redus sau, mai probabil, grupul poate fuziona cu Grupul de testare pentru a ajuta la efectuarea testării sistemului și a testului de acceptanță.
- (4) **Testare.**

- În această fază, responsabilitatea principală a Grupului de testare este pregătirea pentru testarea sistemului, testarea de acceptanță și testarea la fața locului.
- **3-4** persoane ar trebui să fie suficiente.
- (5) **Programare**.
  - Trei grupuri relativ mici compuse din **3-6** persoane.
  - Toate grupurile din această organizare sunt în mod intenționat mici.
  - Munca unui manager de nivel întâi este grea.
  - Dacă se definește corect slujba unei astfel persoane, nu ar trebui să îi se atribuie zece, doisprezece sau douăzeci de oameni pe care să-i gestioneze și apoi să ne așteptăm la un o activitate de prim rang.
  - Grupurile de programare trebuie păstrate suficient de mici pentru ca managerul să se poată implica strâns în detaliile lucrării.
    - Lucrând cu o hoardă, este det aștepta ca această persoană să devină un plimbător de hârtii, copleșită de activități birocratice.
- (6) **Integrare**
  - Este grupul responsabil cu integrarea modulelor.
  - Grupul de integrare poate cuprinde **1-2** persoane.
  - Dacă se face integrarea de sus în jos, grupul numit „**Integrare**” ar putea fi etichetat „**Programare**”.
  - În această situație, acest grup ar fi responsabil pentru modulele de sistem de nivel înalt și pentru integrarea muncii celorlalte grupuri cu propria activitate.
- (7) În același timp, ar trebui adăugat un **Grup pentru managementul configurației software** (Configuration management) în paralel cu Grupul de Integrare, grup care va conține **1-2** persoane.

## 2.2 Organizarea de tip echipă. Echipa programatorului șef

- **Abordarea în echipă** este o modalitate de organizare în jurul unui **grup de specialiști**.
  - Întruchiparea abordării acestei modalități de organizare în programare se numește Echipa programatorului șef.
- **Harlan Mills** de la IBM, inițiatorul conceptului, compară **echipa programatorului șef** cu o **echipă chirurgicală**, în care un chirurg șef planifică și efectuează o operație cu ajutor vital și sprijin de la asistenți cu înaltă calificare, atât chirurgi cât și nechirurgi.
- Ceea ce urmează este o prezentare generală a modului în care această idee este pusă în practică.

## 2.2.1 Participanți. Mod de lucru

- Nucleul unei echipe a programatorului **șef** este format din trei persoane:
  - (1) **Programatorul șef**.
  - (2) **Programatorul adjunct (backup)**.
  - (3) **Bibliotecarul tehnic**.
- (1) **Programatorul șef**
  - **Programatorul șef** este **managerul tehnic responsabil** de **dezvoltarea sistemului de programe**.
  - Această persoană va scrie în mod normal cel puțin modulele critice de „sistem” – adică porțiunea din sistemul programului care exercită controlul asupra tuturor modulelor „de lucru” de nivel inferior și care interacționează cu acestea.
  - În funcție de dimensiunea totală și complexitatea sarcinii, el și adjunctul său pot scrie întregul sistem de programe.
    - Acolo unde sunt implicați și alții, programatorul șef le atribuie lucrul și le integrează toate modulele cu ale sale.
  - **Programatorul șef** este **interfața principală** cu **clientul**, cel puțin în chestiuni tehnice
    - Poate exista un **omolog managerial** care se ocupă de sarcini non-tehnice.
- (2) **Programatorul adjunct**
  - Asistă în orice mod stabilit de programatorul șef,
    - **Funcția sa principală** este să **înțeleagă** toate **fațetele sistemului**, precum o face programatorul șef.
    - A doua sa **funcție** este să fie gata oricând **să preia funcția de programator șef**.
  - În mod normal, programatorului adjunct i se atribuie anumite porțiuni ale sistemului pentru proiectare, codificare și testare, precum și alte sarcini, de exemplu, pregătirea unui plan de testare.
- (3) **Bibliotecarul tehnic (configuratorul)**
  - **Bibliotecarul tehnic** este o persoană diferită de „bibliotecarul general” care conduce biblioteca generală a proiectului și are responsabilitatea normală asociată de obicei cu o bibliotecă.
  - Bibliotecarul **tehnic** este responsabil de rularea bibliotecii de suport pentru dezvoltare sau de gestionarea instrumentului de configurare software utilizat de organizație (CVS - Control Version System, Continuous, ClearCase, CM Synergy, etc).

- Această persoană este un membru deplin și vital al echipei programatorului șef, nu este împrumutat cu jumătate de normă din altă parte.
- **Atribuțiile bibliotecarului** includ:
  - (1) Pregătirea datelor și a programelor în vederea execuției, conform instrucțiunilor programatorilor,
  - (2) Transmiterea și preluarea rulajelor computerului,
  - (3) Evidența și arhivarea tuturor rezultatelor.
- Această echipă de trei poate fi mărită de alte persoane, cum ar fi:
  - (1) Programatori care sunt specialiști într-un domeniu dat.
  - (2) Programatori mai puțin seniori care codifică o anumită porțiune a sistemului proiectată de șeful sau programatorul adjunct.
    - Nu există o limită precisă a mărimei unei astfel de echipe, dar **șase până la opt** participanți, prin consens și experiența de până acum, să fie vârful gamei.
- **Ideea echipei de programatori șef** a luat naștere ca urmare a căutării unor modalități mai bune, mai eficiente de a produce programe complexe fără erori.
  - Întreprinderi uriașe, cum ar fi OS/360 de la IBM, au precizat clar că trebuie găsite modalități de îmbunătățire dramatică a calității și de reducere a costurilor în eforturile viitoare de dezvoltare a software-ului.
  - **Scopul** conceptului de echipă este că acele obiective de calitate și eficiență să fie atinse printr-o organizare foarte strânsă și disciplinată a unui număr mic de oameni foarte motivați, cu experiență și calificare în toate aspectele dezvoltării programului, de la analiză până la proiectare, cod, testare și documentație.
  - Menținând **mic** numărul de oameni, problemele de comunicare umană (și, prin urmare, problemele de comunicare a programului) pot fi reduse drastic.
    - Comparați interfețele posibile între, să zicem, șase persoane, spre deosebire de treizeci!
- Dar pur și simplu **alegerea oamenilor de top și organizarea lor într-un grup mic nu este suficientă**. Au nevoie de cele mai bune instrumente cu care să lucreze:
  - (1) Bibliotecă de suport pentru dezvoltare sau instrumente de configurare software.
  - (2) Dezvoltare de sus în jos, inclusiv instrumente de proiectare, cod, testare și documentare.
  - (3) Programare structurată, tehnologii orientate pe obiecte sau instrumente UML.
  - (4) Acest tip de instrumente sunt mai nou recomandate, desigur, indiferent de structura organizatorică adoptată.

- Pe măsură ce conceptul echipei de programatori șef este încercat de tot mai multe organizații, acesta va fi inevitabil rafinat și va duce la alte idei și instrumente.
- O consecință naturală este utilizarea unei „echipe de echipe”, în care o problemă mare este defalcată într-o manieră ierarhică în subsisteme majore care sunt atribuite echipelor individuale care sunt responsabile în fața unei echipe de „nivel superior”, care este la rândul ei este responsabilă pentru sistemul în ansamblu.

## 2.2.2 Organizarea proiectului folosind echipa programatorului șef

- Presupunând că s-a convenit să fie utilizată în cadrul proiectului curent, abordarea bazată pe echipa programatorului șef.
- Cum ar putea arăta întreaga organizare și cum ar fi gestionate diferitele funcții?
- Organizarea bazată pe echipa programatorului șef, ar putea arăta ca în Figura 6.2.2.2.a. organizare care implică ceva mai mult de jumătate mai multi oamenii implicați în organizarea conventională. Se prezintă două variante a și b.
- Numerele vehiculate aici nu sunt foarte semnificative, deoarece nu se discută despre un job de dimensiune cunoscută.
- Pentru un sistem dat, este posibil să se eliminate jumătate din Grupul de personal, tot Grupul de analiză și proiectare și o parte din Grupul de testare, ca în Figura 6.2.2.2.b.

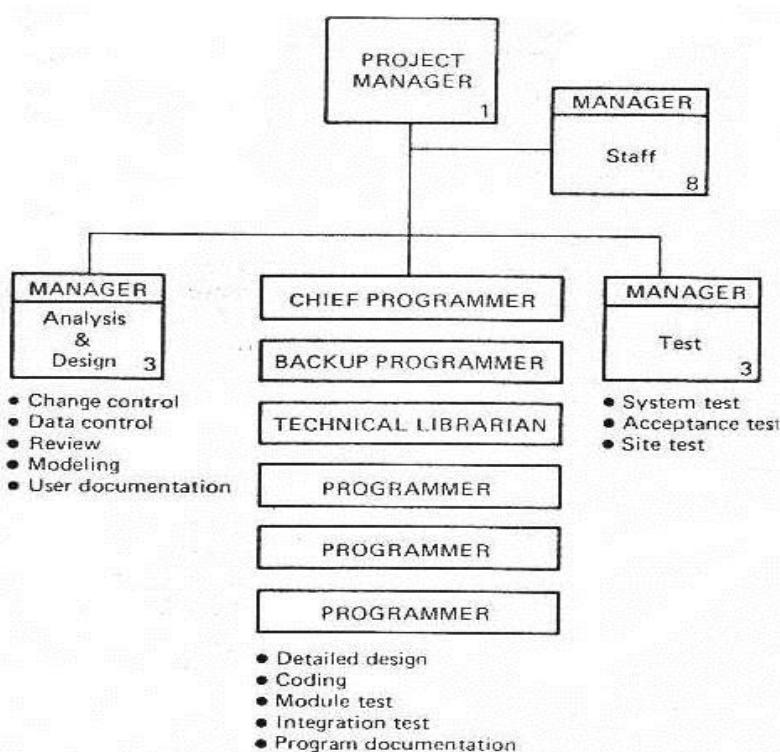
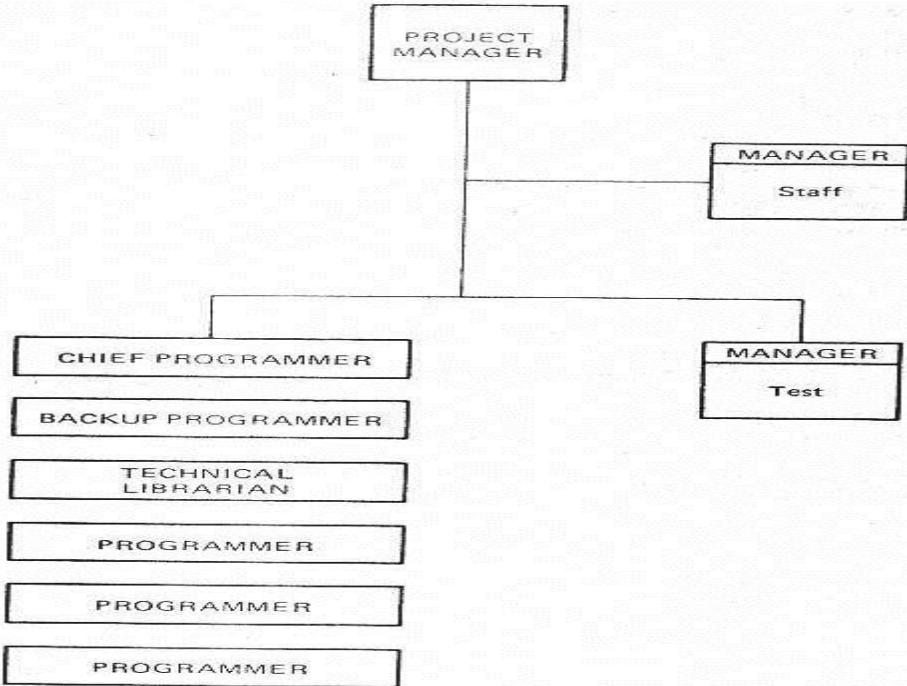


Fig.6.6.2.a. Organizare bazată pe echipă (1)



**Fig. 6.6.2.b.** Organizare bazată pe echipă (2)

- Toate aceste funcții ar putea fi gestionate de echipa de programatori șef.
- Răspunsurile referitoare la numărul de oameni depind nu numai de natura și complexitatea jobului, ci și de talentele oamenilor din echipa programatorului șef.
- Abordarea în echipă presupune membri ai echipei înalt calificați și dedicați, dar de fapt, nu există nimic fixat cantitativ cu privire la acești termeni.
  - Cât de **înalt calificat**?
  - Cât de **dedicat**?
  - Nu există nimic în abordarea în echipă care să scutească managementul de a face **judecăți** și de a lua **decizii critice**.

### 3 Controlul modificărilor

- Procedura de control al modificărilor trebuie stabilită cu grijă.
  - Prea mult control poate sufoca; prea puțin poate destabiliza.
- Nu trebuie construit un imperiu de control al modificărilor în care există volume de proceduri care descriu cum să fie gestionată orice schimbare posibilă.
  - Trebuie avute în vedere doar elementele esențiale asupra cărora este neapărat nevoie de control iar restul trebuie lăsat pentru acțiunile de management de zi cu zi.

### **3.1 Documente de referință (bază)**

- În primul rând, trebuie să se decidă **ce se dorește** a fi controlat, adică care sunt lucrurile care se dorește a fi utilizate ca fundații sau linii de bază în cadrul proiectului.
- Metzger recomandă două **documente de bază (de referință)**:
  - (1) **Specificația problemei**.
  - (2) **Specificația de proiectare**.
- Dacă se depune efort pentru a face aceste două documente cât mai bune posibil și dacă se configuraază convenabil procedurile pentru a controla modificările aduse acestora, atunci este greu să se greșească.
  - În schimb, dacă documentele de referință sunt superficiale și prost realizate sau dacă nu se reușește să se controleze modificările aduse acestora, este greu ca lucrurile să meargă corect.
- Există un **al treilea tip de referință** pe care ar putea fi necesar să fie luat în considerare.
  - Cele două documente menționate mai sus sunt stabilite la începutul ciclului de dezvoltare și sunt utilizate pentru a ghida producția sistemului de programe.
  - Dacă există responsabilitatea dezvoltatorului pentru întreținere sau pentru mai multe versiuni ale sistemului dincolo de livrarea inițială, atunci (3) **Sistemul de programe livrat** devine o nouă linie de bază (o nouă referință).
  - Cu alte cuvinte, atunci când se lucrează la o a doua, a treia sau a n-a versiune a sistemului de program, ultima livrare poate fi considerată ca linie de bază.
- În continuare, totuși, se va discuta doar despre un ciclu de dezvoltare cu **o singură livrare**.

### **3.2 Proceduri de control**

- Dacă se cade de acord să se controleze modificările în raport cu specificația problemei și cu specificația de proiectare, în continuare trebuie luat în considerare un mecanism de control simplu care poate fi personalizat pentru a se potrivi proiectului.
- Ori de câte ori o persoană sesizează că este nevoie de o modificare despre care crede că poate afecta una dintre liniile de bază, trebuie parcursi următorii pași:
  - (1) O persoană propune o **modificare formală**.
  - (2) Grupul de analiză și proiectare analizează modificarea propusă și recomandă adoptarea sau respingerea ei.
  - (3) Recomandarea este apoi înaintată **Bordului de control al modificărilor**, care își pronunță decizia, sub rezerva contestării ei fie de către dezvoltator, fie de către client.

- (4) **Grupul de Analiză și Proiectare** documentează decizia, iar modificarea, dacă este adoptată, este **implementată**.
  - Acum să aruncăm o privire mai atentă asupra modului în care ar putea funcționa această procedură.
  - (1) **Propunerea unei modificări**
    - Oricine, fie din organizația dezvoltatorului, fie din partea clientului, poate propune o schimbare.
      - Pentru a face acest lucru, se complează un formular simplu de propunere de modificare care descrie necesitatea modificării și, dacă este posibil, modalitatea de a face modificarea.
    - De regulă, un programator propune o modificare **numai dacă** apreciază că unul dintre documentele de bază ar putea fi afectate.
      - Nu trebuie trimisă nici propunere de modificare de fiecare dată când o bucată de design detaliat pentru un modul este ușor modificată.
    - Există o categorie de modificări care nu se încadrează în această procedură formală de control, dar care trebuie menționată în trecere:
      - Să presupunem că un programator dorește să facă o modificare la unul dintre modulele deja trimise pentru testul de integrare.
        - Modificarea nu afectează nici specificația problemei, nici specificația de proiectare, dar afectează designul detaliat – specificația de codificare pentru modul.
        - Modificarea ar putea fi pentru a corecta o eroare găsită cu întârziere sau pentru a îmbunătăți o bucată de cod.
      - Dacă se acceptă sau nu schimbarea în acest caz ar trebui să fie la latitudinea **celui care se ocupă de testarea de integrare** care implică modulul respectiv.
 

Dacă modificarea are sens, ar trebui acceptată numai sub forma unei noi copii a modulului de program, a unei specificații de codare corectate și a unei identificări actualizate a modulului.
- (2) **Investigarea propunerii**
  - **Toate modificările propuse** sunt investigate de Grupul de analiză și **proiectare**.
  - Oricărei modificări propuse îi este atribuit un **investigator**.
    - Investigatorul analizează propunerea pentru a-și face o idee despre importanța și impactul acesteia, apoi o planifică pentru o decizie la o întâlnire viitoare (de obicei următoarea întâlnire programată) a Bordului de Control al Modificărilor.

- Dacă modoficarearea este urgentă, poate fi convocată o întâlnire specială de îndată ce investigatorul are suficiente informații pentru a face o recomandare.
- **Investigatorul:**
  - (a) Analizează toate aspectele pertinente ale modificării.
  - (b) Scrie un raport.
  - (c) Trimită o copie a raportului fiecărui membru al Bordului într-un timp rezonabil (să zicem, două zile lucrătoare) înainte ca Bordul să se întrunească.
- Raportul fiecărui investigator trebuie să includă:
  - (a) Un rezumat al modificării propuse.
  - (b) Numele și organizația inițiatorului.
  - (c) Clasificarea modificării (Tipul 1 sau 2) după cum este determinată de investigator (a se vedea mai jos).
  - (d) Impactul modificării asupra costurilor, planificării sau altor programe.
  - (e) O recomandare pentru sau împotriva **adoptării**.

- (3) **Tipuri de modificări.**

- Investigatorul poate clasifica modificarea în una dintre două categorii:
  - **Tipul 1** - dacă modificarea afectează oricare dintre documentele de referință sau cauzează un cost, o modificare de planificare sau are un alt impact.
  - **Tipul 2** dacă modificarea nu afectează nicio linie de referință, are un cost neglijabil, nu afectează planificare sau nu are vreun alt impact.
- Trebuie avut grijă că modificările să fie prea ușor clasificate ca de tip 2, atunci când chiar costă ceva și ar trebui să fie de tip 1.
  - Nu trebuie să fiți „băieți drăguți” și să permiteți ca proiectul să fie ciugulit de prea multe modificări de tip 2.
- Desigur, se pot face lucrurile să fie mult mai complicate, dar nu este deloc recomandabil.
  - Nu are niciun sens să se inventeze o duzină de categorii de schimbări diferite pentru a acoperi tot felul de combinații de situații.
  - Fie schimbarea va cauza unele probleme (Tipul 1), fie nu necesită atenție (Tipul 2).
  - Chiar și mașinăriile greoaie ale managementului configurării la nivelul guvernului federal al SUA se descurcă cu doar două categorii de schimbări. Cu siguranță nu vrei să fii rușinat de cea mai mare birocrație din lume!

- (4) **Bordul de control al modificărilor**

- Bordul trebuie să fie format din reprezentanți ai diferitelor grupuri de proiect.
- La întâlnirile periodice (să zicem, o dată pe săptămână), bordul ar trebui să ia în considerare toate propunerile de modificare programate.
- Bordul discută fiecare modificare și dispune referitor la ea.
- Managerul de proiect trebuie să decidă dacă: (a) bordul va funcționa democratic prin vot pentru fiecare problemă sau (b) dacă i se permite președintelui să ia decizia după ascultarea argumentelor.
  - Democrația este grozavă, dar cu siguranță se va descoperi că lucrurile se mișcă mult mai repede dacă i se dă președintelui puterea de a decide care este recomandarea consiliului.
  - Managerul de proiect poate oricând să anuleze decizia dacă unul dintre ceilalți membri ai consiliului îl convinge că o anumită decizie a fost una proastă.
    - Totuși, nu se recomandă a se proceda prea des în acest mod, pentru a nu diminua autoritatea președintelui bordului.
- **Bordul de control al modificărilor** ar trebui să fie compus din următoarele persoane:
  - **Președinte**: managerul Grupului de analiză și proiectare. El trebuie să fie dur, corect, perspicace din punct de vedere tehnic și priceput politic.
  - **Membri permanenti**: managerul Grupului de programare, managerul Grupului de testare și managerul Grupului de personal.
  - **Alți membri**: investigatorul propunerea luată în considerare; **personal tehnic** invitat de oricare dintre membrii permanenti.
- La orice reuniune a bordului, atunci, trebuie să fie prezenți cel puțin cinci participanți, patru membri regulați și un investigator.
  - Este important să nu se permită ca aceste întâlniri să devină prea mari invitând prea mulți figuranți, dar, evident, dacă există cineva care poate arunca mai multă lumină asupra propunerii decât oricine altcineva, acea persoană ar trebui să fie prezentă.
  - Adesea, aceasta va însemna că persoana care a propus schimbarea va fi acolo.
- Ar trebui să fie invitat clientul la ședințele bordului?
  - În general, da, deși pot fi momente când ar fi preferabil ca acesta să nu fie de față.
    - De exemplu, clientul ar trebui exclus ori de câte urmează să fie expuse sau discutate date sensibile ale companiei dezvoltatoare sau probleme delicate referitoare la costuri.

- Cea mai bună modalitate de a rezolva această situație este aceea de a nu invita clientul la discuția respectivă.
  - Apoi, odată ce costul este clar, clientul poate fi invitat ori de câte ori se dorește.
- (5) **Tipuri de recomandări**
  - Dacă bordul este de acord cu investigatorul că o modificare este **de tip 2**, modificarea ar trebui acceptată automat și nu este necesară nicio acțiune suplimentară a bordului.
    - Modificarea este tratată urmând Procedura de remediere a erorilor.
  - Dacă este de tip 1, bordul trebuie să recomande ce dispune referitor la modificare. Există două posibilități:
    - **Acceptarea modificării** și indicarea când trebuie făcută modificarea (immediat sau într-o versiune viitoare a programului).
    - **Respingerea modificării**.
- (6) **Modificări propuse de beneficiar (client)**
  - Unele modificări vor fi propuse nemijlocit de către client.
  - Ele trebuie investigate, luate în considerare de către bord, estimate costurile și impactul lor și aprobată oficial de către client.
  - Este întotdeauna dreptul clientului să anuleze orice decizie a bordului, dar ca urmare, trebuie negociate modificările corespunzătoare ale contractului semnat.
- (7) **Implementarea unei modificări**
  - În funcție de recomandarea bordului pentru o modificare de **tip 1**, sunt posibile două acțiuni finale:
    - Dacă consiliul recomandă **respingerea**, propunerea este înregistrată ca închisă.
    - Dacă consiliul recomandă **adoptarea**:
      - (1) Investigatorul întocmește un rezumat al modificării, în care precizează costul acesteia și planificarea pentru efectuarea modificării.
      - (2) Pachetul este apoi dat Managerului de proiect pentru a fi semnat.
      - (3) Managerul de proiect îl semnează.
      - (4) Dacă există un impact asupra costului sau planificării, managerul de proiect trimite pachetul clientului pentru aprobare.
      - (5) Atunci când clientul îl aprobă în scris, investigatorul distribuie tuturor celor implicați un document scris care descrie modificarea.
      - (6) Acum modificarea poate fi implementată de către programatori.

- Cele mai sus prezentate reprezintă **procedura formală**.
- Vor exista însă multe situații în care o modificare nu poate fi reținută zile sau săptămâni.
  - În astfel de cazuri, Managerul de proiect poate accelera procesul investigând imediat, convocând o ședință rapidă, specială a bordului, redactând recomandarea pe scurt, aprobat-o verbal, obținând acordul verbal al clientului și cerându-i programatorului să continue.
  - Dar, managerul de proiect trebuie să se asigure că documentele aferente își urmează cursul oficial - aprobaarea clientului, notificarea de modificare și aşa mai departe. În caz contrar, managerul de proiect va pierde în curând evidența lucrurilor.
- Periodic, președintele bordului trebuie să înainteze managerului de proiect o listă rezumativă a tuturor modificărilor luate în considerare, recomandarea bordului în fiecare caz și o expunere foarte scurtă a principalelor argumente pro și contra.
  - Dacă managerul de proiect observă ceva cu care crede că nu este de acord, el poate cere bordului să reanalyzeze problema.
- Grupul de analiză și proiectare este, de asemenea, responsabil cu ținerea evidenței tuturor modificărilor apărute în proiect.
  - În multe proiecte, modificările apar în număr mare și foarte rapid.
  - Unele sunt desemnate pentru implementare imediată, altele sunt amânate pentru o versiune ulterioară a sistemului programului.
  - Este important ca toată lumea – atât dezvoltatorul cât și clientul – să știe exact care este starea lor și când vor fi făcute modificările acceptate.

## 4 Unelte de programare

- Instrumentele pentru **activitatea de programare** trebuie selectate înainte ca Faza de Programare să înceapă efectiv.
- Unele instrumente, cum ar fi sisteme de operare, modele de simulare, HIPO, pseudocod, diagrame structurate, diagrame de flux, tabele de decizie și matrice de acoperire, au fost menționate în capitolul anterior.
  - Aceste instrumente prezentate ca **instrumente de analiză și proiectare**, în mare parte continuă să servească și ca **instrumente de programare**.
- Instrumentele care vor fi prezentate în continuare sunt scurte descrieri ale altor instrumente pe care managerul de proiect, le poate lua în considerare în timpul fazei de programare.
  - Unele dintre ele sunt **programe**, altele sunt unelte hardware, iar altele sunt pur și simplu documente.

- În proiectele de mari dimensiuni (vezi Capitolul 10) este adesea necesar să existe un grup separat de personal care oferă restului membrilor proiectului ajutoare adecvate de programare (simulatoare, modele, testere, etc.).

#### **4.1 Specificații scrise**

- Abordarea adoptată în acest curs, a evidențiat trei documente cheie: **Specificația problemei**, **Specificația de proiectare** și **Specificația de codificare**.
- Mai devreme în acest capitol a fost menționat un alt document numit **Specificația de Testări de Integrare**.
- Toate aceste instrumente sunt atât de importante încât merită să fie reluate.
- (1) **Specificația problemei.**
  - Este scrisă de analiști în timpul fazei de definire.
  - Este un document de bază.
  - Descrie problema la care se referă proiectul aflat în dezvoltare.
- (2) **Specificația de proiectare.**
  - Este scrisă de designerii de programe în timpul fazei de proiectare pentru a descrie întregul sistem de program, de fapt, soluția problemei.
  - Este, de asemenea, un document fundamental care pune bazele tuturor proiectelor detaliate care urmează.
- (3) **Specificațiile de codificare.**
  - Aceste specificații sunt scrise de programatori în timpul fazei de programare.
  - Fiecare specificație descrie în detaliu proiectarea pentru o parte a sistemului general prevăzut în Specificația de proiectare.
  - Codarea se face în baza acestei specificații.
  - Managerul de proiect poate decide ca specificația de codificare să fie inclusă ca un comentariu extins în cod. În acest caz, trebuie stabilite reguli speciale specifice.
- (4) **Specificația testării de integrare.**
  - Redactată în timpul fazei de proiectare, această specificație descrie obiectivele testării de integrare, procedurile de testare specifice și datele de testare, pentru atingerea acestor obiective.

#### **4.2 Executive de testare**

- Majoritatea sistemelor de programe includ o formă de **program de control** sau **executiv de testare**.

- (1) În **testarea de jos în sus**
  - Un **executiv de testare** este de regulă, o versiune modificată a eventualului **program executiv complet**.
  - Începe ca o formă simplificată, poate doar ca un schelet al programului final.
  - Este scris din timp pentru a oferi un cadru real pentru testarea integrării.
  - Unele **executive de testare** conțin module „fictive” sau „stub-uri”, care sunt înlocuite treptat pe măsură ce omologii lor „adevărați” ies din testul modul.
  - Stub-urile pot face mai mult decât să înregistreze și să imprime o indicație că au fost invocate.  
De exemplu, ele pot efectua o operațiune simplă, cum ar fi imitarea a ceea ce va face modulul real atunci când acesta este în cele din urmă introdus în sistem.
  - Pe măsură ce stub-urile sunt înlocuite cu module reale, sistemul de programe începe să prindă contur.
- (2) În **testarea de sus în jos**,
  - Executivul de testare este în esență codul pentru modulele de nivel superior („modulele sistem”).
  - Unele executive de testare conțin ajutoare speciale de testare care sunt eliminate în timpul etapelor finale ale testului de integrare.
    - (1) Un astfel de ajutor de testare poate fi un **program de urmărire** pentru a ține evidență **secvențelor de evenimente din sistem** pentru analiză ulterioară.
    - (2) Un alt ajutor este un program care oferă afișaje sau „instantanee” ale registrelor cheie ale computerelor sau zonelor de stocare în momente strategice.
      - Acest tip de ajutoare de testare se numesc Depanatoare (Debugger) și de obicei fac parte din Instrumentele de dezvoltare a programelor.
      - *Un exemplu de executiv de testare utilizat în etapele incipiente ale dezvoltării sistemelor „în timp real” este o versiune non timp real a programului de control al sistemului. În mod similar, adesea este scris un program de control cu un singur procesor înainte de dezvoltarea unei capacitați complete de multiprocesare.*
  - Un executiv de testare poate fi complex, dar, de regulă, trebuie menținut cât mai simplu din două motive:
    - (1) În primul rând, valoarea sa constă parțial în a-l pregăti devreme, înainte ca programul executiv „adevărat” să fie terminat. Dacă se depune prea mult efort pentru executivul de testare, acesta nu va fi gata mult mai devreme decât cel real.

- (2) În al doilea rând, cu cât executivul este mai complex, cu atât va fi mai greu să fie separate problemele sau erorile, de cele ale modulelor reale care trebuie testate.

### 4.3 Simulatoare de mediu (ambient)

- Un **simulator de mediu** este un program care înlocuiește temporar, în scopuri de testare, o parte a lumii (ambientului) cu care sistemul de programe trebuie în cele din urmă să interacționeze.
  - *De exemplu, să presupunem că se dezvoltă un sistem de programe pentru dirijarea traficului aerian. Unul dintre echipamentele cu care programele trebuie să comunice în cele din urmă este un set radar. Dar pentru că radarul este încă în curs de dezvoltare sau pentru că conectarea la el nu este încă fezabilă, poate fi necesar să se dezvoltate programe care „să arate” ca un radar. Aceste programe speciale încercă de fapt să simuleze atât intrările radarului în sistemul de programe operațional, cât și răspunsurile radarului la ieșirile sistemului.*
- Alte programe de simulare ar putea înlocui spre exemplu consolele speciale de afișare încă în curs de dezvoltare.
- Alte altele pot fi folosite pentru a alimenta sistemul cu seturi de date reprezentând condițiile operaționale din lumea reală; în exemplul nostru, condițiile de funcționare pot fi încărcături de trafic sau informații despre vreme.
- Pentru a fi cel mai eficient, un simulator de mediu trebuie să fie transparent pentru programele utilizate, adică programele ar trebui să necesite cât puține sau dacă este posibil deloc, modificări speciale pentru a comunica cu simulatoarele.
  - Programele ar trebui să funcționeze ca și cum ar avea de-a face cu lumea reală.
  - Orice modificare care se face pentru a permite sistemului de programe să ruleze cu simulatorul vă diminua încrederea că sistemul va funcționa corect atunci când simulatorul este înlocuit cu echipamentul real.
- În funcție de dimensiunea și natura jobului, **programele de simulare a mediului** pot:
  - Să opereze pe același computer ca și programele operaționale.
  - Sa fie executate pe un computer separat care interfețează cu cel care rulează sistemul de programe aflat în dezvoltare.
- Costul dezvoltării simulatoarelor de mediu poate fi enorm.
  - Spre exemplu, în munca de control al traficului aerian sugerată mai sus, costul simulatoarelor ar putea fi cu ușurință mai mare decât cel al programelor operaționale în sine.
- Necesitatea unor astfel de instrumente trebuie abordată în fazele de definire și proiectare.

- Simulatoarele necesită aceeași grijă în proiectare și programare ca și programele operaționale.

#### **4.4 Medii de programare specialize**

- În zilele noastre, organizațiile dezvoltare folosesc ca instrumente de dezvoltare medii de programare specialize.
- Astfel de instrumente de dezvoltare sunt specializate pe diferite limbaje de programare și oferă caracteristicile necesare dezvoltării codului sub formă de funcționalități integrate cum ar fi editare, compilare, depanare, facilități de urmărire (trasare), ajutor pentru optimizarea codului, etc.,.
- Exemple clasice: *Developer Studio (Microsoft)*, *Visual Studio*, *JavaBuilder*, *Borland C și Pascal*, *Rational Rose*, *Kituri de dezvoltare pentru diferite microcontrolere, diferite SDK-uri (Kituri de dezvoltare software)*, etc.,.

#### **4.5 Ajutoare automate de documentare (Automated Documentation Aids)**

- Va trebui să se afle ce ajutoare automate de documentare, dacă există, sunt disponibile în compania dezvoltatoare sau de la alte companii.
- Există o mulțime de sisteme care au scopul de a automatiza desenarea și actualizarea diagramelor de flux, de exemplu sau documentarea codului.
- Dacă se are în vedere utilizarea un sistem automatizat de documentare, decizia trebuie luată din timp.
  - Este costisitor, irositor și agravant să se decidă la jumătatea unui proiect că se trece la un sistem de documentare automatizat.
- Există o mulțime de astfel de instrumente de documentare automatizate oferite de diferite companii software (Doxygen, JavaDoc, SoDA, etc.).

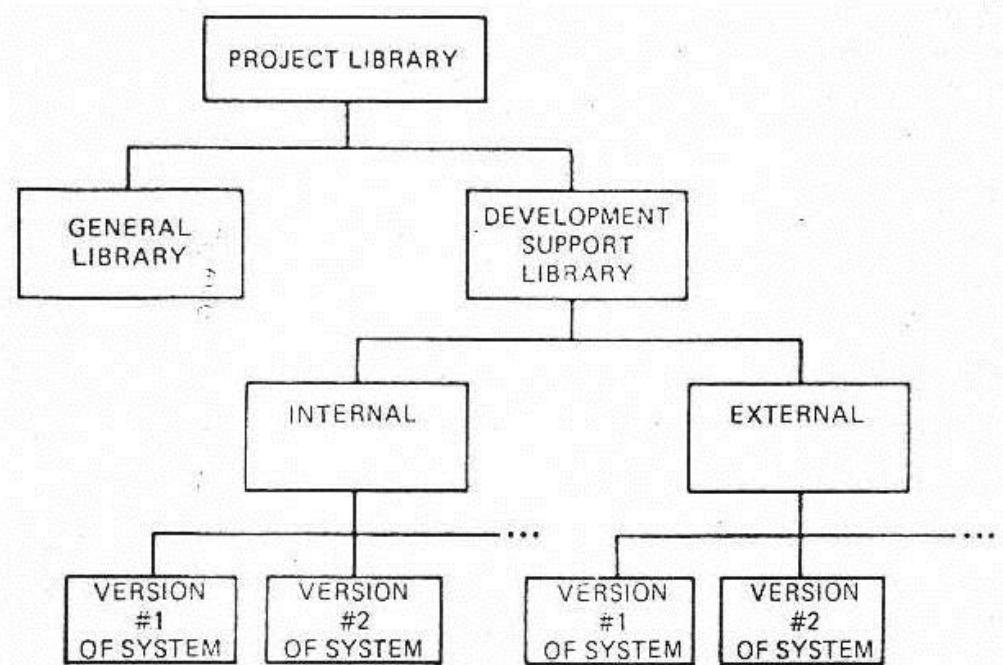
#### **4.6 Monitoare software și hardware**

- Un monitor software este un cod adăugat unui sistem în scopul inspectării și colectării datelor din punctele cheie în timpul execuției.
  - Un exemplu de mare succes este Sistemul de colectare a statisticilor (SGS) utilizat cu ani în urmă în programul Apollo de la Houston. Datele culese de SGS în timpul funcționării sistemului Apollo au arătat dezvoltatorilor multe despre timpii de execuție și frecvența de funcționare a modulelor cheie. Aceste date au fost folosite și pentru a calibra și îmbunătăți acuratețea modelelor de simulare Apollo.
- Un monitor software poate reprezenta un efort mare de programare, ca în cazul SGS, sau ar putea echivala cu inserarea câtorva rutine mici în punctele critice din sistemul de programe pentru a analiza utilizarea modulelor.

- Unii producători oferă spre închiriere sau vânzare dispozitive hardware care pot fi atașate la un computer pentru a efectua anumite măsurători.
- Majoritatea acestor monitoare sunt menite să ofere date care să ajute la a determina cât de eficient folosește sistemul diferite canale de intrare-iesire sau modul în care puterea de calcul a sistemului este partajată între activitățile de calcul și cele de intrare-iesire.

#### 4.7 Biblioteca proiectului

- O **bibliotecă** este prin definiție o colecție organizată de informații.
- Biblioteca proiectului ar trebui să conțină două secțiuni:
  - (1) Biblioteca **Generală**.
  - (2) **Biblioteca suport (de sprijin) pentru dezvoltare**.



**Fig.4.7.a.** Biblioteca proiectului

##### 4.7.1 Biblioteca generală

- Biblioteca generală păstrează copii master sau fișiere master ale tuturor documentelor de proiect din această secțiune, altele decât cele din Biblioteca de suport pentru dezvoltare.
- O astfel **listă de documente de bază** ar trebui să includă:

- Planul proiectului.
  - Specificarea problemei.
  - Specificația de design.
  - Specificațiile de testare.
  - Note tehnice.
  - Note administrative.
  - Documente de modificare.
  - Rapoarte de testare.
  - Rapoarte de stare.
  - Istoricul proiectului.
  - Formulare.
  - Index de documentare.
- În plus, întotdeauna trebuie păstrată o copie a modulelor și a documentației pentru orice versiune anterioară a sistemului care a fost finalizată și livrată.
  - Fiecare document din bibliotecă trebuie să primească un **număr unic de identificare** atribuit de către bibliotecar.
    - Bibliotecarul ar trebui să aibă acces rapid la echipamentul de reproducere și să poată pune la dispoziție o copie a unui document principal atunci când este solicitat.
  - Nu trebuie să lase niciodată copia master a unui document din mâinile lui.
    - El poate păstra la îndemână un număr de copii ale documentelor solicitate de către client, în loc să le reproducă doar la cerere.
  - Bibliotecarul trebuie să țină un jurnal al numerelor de document, astfel încât, atunci când orice membru al proiectului este gata să emită un document nou într-o anumită categorie, de exemplu, o notificare de modificare a designului, trebuie doar să sună la bibliotecă pentru a obține un număr unic de document.
  - Periodic, bibliotecarul ar trebui să actualizeze Indexul de documentație și să trimită sau să publice pe Intranet noua versiune.
    - Indexul de documentație este o listă a tuturor documentelor aflate la acel moment în bibliotecă.
    - Lista trebuie să arate titlurile documentelor, autorii, datele emiterii și numerele de identificare.
  - Bibliotecarul ar trebui, de asemenea, să adune periodic **înregistrările vitale** și să le stocheze ca **copii de rezervă** într-o unitate separată fizic de unitatea cotidiană.

- Înregistrările vitale sunt orice materiale desemenate de managerul de proiect ca fiind necesare pentru a reconstrui sistemul dacă în cazul unui incendiu sau o altă catastrofă îl-ar distrugă iremendabil.
- Înregistrările vitale pot include o copie pe CD (DVD) a sistemului de programe în starea în care se află el la acel moment, împreună cu o copie a specificațiilor care descriu sistemul la acel moment.
- Costă relativ puțin să se facă această treabă și dar beneficiile pot fi de neprețuit.
  - *Metzger dă un exemplu. Un proiect din anii 1960 nu a aplicat astfel de prevederi. Programele au fost dezvoltate într-o clădire „ignifugă” la o bază a forțelor aeriene din Florida. Sistemul era aproape finalizat. Într-o noapte clădirea a fost distrusă de un incendiu și practic totul s-a pierdut – carduri, casete, liste, lucrări. Și, desigur, nu a existat nicio altă copie a sistemului. Povestea are însă un final fericit, deoarece programatorii aveau destule înregistrări neautorizate în casele lor pentru a pune din nou sistemul pe picioare. Contract salvat, plata efectuată.*

#### **4.7.2 Biblioteca suport de dezvoltare**

- Printre inovațiile și îmbunătățirile importante ale procesului de programare se numără și **Biblioteca suport de dezvoltare** (Development Support Library (DSL), numită și cu alte denumiri, inclusiv Program Support Library sau Programming Production Library).
- **Biblioteca suport de dezvoltare** este locul central de stocare al proiectului pentru versiunea oficială a sistemului de programe în curs de dezvoltare. Este alcătuită din două secțiuni, internă și externă precum și din procedurile care reglementează utilizarea acestora.
- (1) **Biblioteca internă** conține, pe disc, bandă, CD-uri, stream-ere, memorii calculator sau pe alte medii de stocare următoarele componente:
  - (a) Programele în curs de dezvoltare și datele referitoare la dezvoltarea lor.
  - (b) Natura exactă ceea ce este stocat depinde de natura proiectului și în special de sistemul informatic utilizat;
  - (c) De obicei, biblioteca internă include cel mai recent cod sursă și cod obiect pentru toate modulele din sistemul dezvoltat, date de testare, instrucțiuni ale limbajului de control (scripturi) și aşa mai departe.
- (2) **Biblioteca externă** este formată din:
  - (a) Listingurile corespunzătoare stării curente a fiecărui tip de date stocat în biblioteca internă și caiete de execuție (run notebook-uri) care arată rezultatele testelor rulate.
  - (b) Biblioteca externă conține, de asemenea, documente de arhivă — versiuni mai vechi ale listingurilor aferente proiectului, pentru a fi utilizate în scopuri istorice și ca și copii de rezervă în cazul pierderii documentelor actuale.

- (3) **Proceduri.**
  - (a) Biblioteca de suport (de sprijin) pentru dezvoltare servește ca locație unică pentru versiunea oficială a sistemului de program în curs de dezvoltare.
  - (b) Ar trebui să fie susținută de instrumente specializate pentru managementul configurației (CVS, Continuous, ClearCase etc.)
  - (c) Elimină practic reținerea versiunilor private ale modulelor de către programatori individuali și face sistemul actual complet vizibil și deschis spre inspecție de către toți membrii proiectului.
  - (d) Toate sumiterile de cod nou, modificări ale codului existent, solicitări de rulări de testare și aşa mai departe, sunt transmise prin bibliotecarul tehnic, cuiva special instruit pentru acest job.
  - (e) Bibliotecarul este interfață unică dintre programator și proiect; el introduce toate intrările noi în sistem și notifică și distribuie toate ieșirile noi în caietele sau în fișetele corespunzătoare.
- O anumită bibliotecă suport pentru dezvoltare poate conține la un moment dat mai mult de un sistem de programe, deoarece:
  - Pot exista versiuni ale unui sistem la diferite niveluri de finalizare la un moment dat, în special în cazul proiectelor mai mari;
  - Biblioteca poate servi o comunitate mai mare decât doar proiectul curent. Nu există niciun motiv pentru care un număr de proiecte să nu poată utiliza aceeași facilitate de bibliotecă. În acest caz, poate fi necesar mai mult de un bibliotecar tehnic.
- Fiecare set de date, fie că este cod de program, cod de control sau date de testare, este identificat în mod unic în bibliotecă.
- Identificatori unici sunt utilizati de asemenea pentru a separa diferite versiuni ale unui program unele de altele și proiecte complet diferite unele de altele.
- Desigur, este important ca bibliotecarul tehnic să fie bine pregătit și capabil deoarece activitatea lui este una extrem de importantă.
- Este important, de asemenea, ca bibliotecarul tehnic să fie ferm și să nu fie ușor de influențat.
  - Programatorilor nu ar trebui să li se permită să ocolească procedurile bibliotecii; în caz contrar, biblioteca își pierde imediat valoarea de punct vital de control al proiectului.
  - Programatorii sunt renumiti pentru abilitățile lor de a introduce acele mici schimbări de ultim moment. Nu sunt mici schimbări. Fiecare schimbare este o dinamită potențială, mai ales dacă proiectul se apropiați de testul sistem sau de cel de acceptanță.
  - Trebuie asigurată o protecție totală împotriva acelor patch-uri de la miezul nopții, făcând practic inaccesibilă stocarea internă a bibliotecii.

- Programatorii ar putea fi supărați, dar asta este mai bine decât să explodeze în față un test de acceptare în timp ce clientul este de față.

## 5 Atribuțiile managerului de proiect în faza de programare

- În timpul fazei de programare, atributul esențial al managerului de proiect îl constituie promovarea **excelenței**.
  - Pur și simplu îndeplinirea sarcinilor, respectarea termenelor limită, respectarea bugetelor, recompensarea echitabilă a lucrătorilor, mulțumirea clientului, menținerea integrității personale și a companiei etc. — toate acestea sunt esențiale, dar nu suficiente.
- Managerul trebuie întotdeauna:
  - Să caute modalități de a oferi un produs excelent.
  - Să asigure în același timp satisfacția personală și creșterea în carieră a oamenilor săi.
  - Aceste lucruri merg mână în mână;
    - Un produs excelent va pune în evidență cariera celor care îl produc.
    - Oamenii mulțumiți cu carierele în creștere vor produce un produs mai bun.
- Există o mulțime de puncte de vedere diferite și uneori contradictorii cu privire la modul de a gestiona un proiect.
  - **Townsend** îl descrie pe managerul de proiect ca fiind cel care „cară apă pentru oamenii săi, astfel încât aceștia să-și poată continua cu treaba”.
  - Există manageri excelenți care sunt de acord cu acest punct de vedere.
  - Există, de asemenea, unii manageri care susțin opusul, având un fel de viziune ptolemeică, „Universul există pentru a-i servi pe ei!”
- „A căra apă” este important, dar sunt mult mai multe cerințe de îndeplinit:
  - Pentru a fi cu adevărat eficient, un manager trebuie să fie respectat, iar pentru a fi respectat trebuie să conducă cu adevărat.
  - Aceasta înseamnă să ai suficientă cunoștere sau să știi cum să găsești și să folosești consilierea tehnică competentă, pentru a putea stabili direcția tehnică pentru organizație.
  - Este o activitate extrem de grea, dar absolut necesară în urmărirea excelenței.
- Următoarele secțiuni ale capitolului descriu atribuțiile unui manager în timpul fazei de programare.
  - În multe privințe, aceste atribuții sunt aceleași pentru un manager de nivel întâi ca și pentru un manager de nivel superior.

- Diferențele semnificative între jobul de management de nivel întâi și cel de nivel superior sunt rezumate în ultima secțiune.
- Iată câteva dintre subiectele legate de management care vor fi discutate:
  - (1) Conducerea tehnică
  - (2) Planificarea și controlul
  - (3) Comunicarea
  - (4) Asigurarea condițiilor de muncă și a instrumentelor adecvate
  - (5) Atribuirea de sarcini
  - (6) Orele de activitate (working hours)
  - (7) Adăugarea mai multor persoane
  - (8) Raportarea stării tehnice
  - (9) Raportarea situației financiare
  - (10) Instruire
  - (11) Evaluare și consiliere
  - (12) Menținerea unei atmosfere sănătoase (sanity maintenance)

## 5.1 Conducerea tehnică

- Managerul nu trebuie să fie cea mai bună persoană tehnică din organizație pentru a stabili direcția tehnică.
- Pe lângă atributile evidente, managerul mai are nevoie de:
  - (1) O dorință intensă a oamenilor săi de a fi acordați nemijlocit pe tehnologia business-ului (afacerii).
  - (2) Ca managerii să cunoască, să înțeleagă și să aplique cele mai recente metode de management.
- În domeniul programării a existat o tendință crescândă de a lega mai strâns de **managementul și tehniciile de programare**.
- **Managementul programării** a devenit poate ceva mai **specializat**.
  - Teoria generală a managementului spune că un manager bun poate gestiona la fel de bine fie o piață de carne, fie un proiect computer și, fără îndoială, acest lucru este valabil pentru o mână de oameni talentați.
  - Pentru mulți alții, totuși, în zilele noastre acest lucru nu funcționează. Asta e și parerea noastră.
- De fapt, **gestionarea proiectelor software** este o problemă dificilă, deoarece **managerul de proiect** are nevoie atât de **instrumente tehnice specifice**, cât și de **instrumente de management**. În fiecare caz:

- Acestea ajută oamenii tehnici în executarea sarcinilor lor, de la analiză până la proiectare, codificare și testarea produselor lor.
- Dar, în același timp, aceste tehnici ajută liderii tehnici și managerii să coordoneze și să controleze procesul de dezvoltare.
- Ce este însă cu adevărat important:
  - (1) Manualul tradițional al managerului nu este suficient; este extrem de importantă cunoașterea în detaliu a acestor instrumente tehnice/de management dar ele trebuie folosite de manager pentru a comunica cu oamenii lui tehnici.
  - (2) Având în vedere focalizarea foarte importantă și punctul de control oferit de o bibliotecă de sprijin pentru dezvoltare (sau ceva similar), există mult mai multe oportunități pentru oricine din proiect de a înțelege starea sistemului.
  - (3) Managerul trebuie să se asigure că în buget sunt alocați întotdeauna bani și timp, pentru ca toți membrii proiectului să-și continue educația în carieră.
  - (4) Echipa trebuie întotdeană încurajată să-și facă timp pentru a afla despre businessul firmei, să se perfecționeze, să studieze, să se informeze cu ultimele noutăți, fără să se simtă vinovată pentru că nu „produce”.
- În ceea ce privește conducerea tehnică, de obicei, în timp apare sentimentul de a nu te mai simți competent pentru a te ocupa de probleme tehnice odată ce ai devenit manager.
  - Dar există modalități disponibile la-ndemâna managerului de proiect pentru a rezolva astfel de situații:
    - (1) Studiul și însușirea noilor instrumentele.
    - (2) Cultivarea sentimentului că oamenii tehnici înțeleg strădania managerului de a percepe aspectele tehnice și înțeleg de asemenea așteptările din partea acestuia de a se utiliza în comunicare un limbaj pe care să îl poată înțelege.

## 5.2 Planificare și control

- Activitățile de **planificare și control** sunt esența oricărei activități de management.
  - A **planifica** înseamnă a stabili ceea ce vrei să se întâmple.
  - A **controla** înseamnă a te asigura că se întâmplă.
- **Planificarea și controlul** sunt subiectele centrale ale întregul curs.

## 5.3 Comunicarea

- Problema pe care majoritatea oamenilor o au în comunicare nu este cât de bine vorbesc, ci cât de bine ascultă.

- Unii oameni sunt slabii la comunicarea orală, dar compensează notând lucrurile.
  - După o întâlnire, de exemplu, acest tip de persoană notează ceea ce crede că s-a spus sau ce decizii s-au luat la întâlnire. Aceasta este o tehnică foarte bună.
  - Există întâlniri în fiecare zi în orice organizație, din care oamenii ies cu impresii complet diferite uneori chiar contradictorii despre ceea ce s-a spus sau s-a decis.
- Un **manager** ar trebui să comunice **propria sa filozofie de management** oamenilor care lucrează cu el. Este o sarcină dificilă, dar absolut necesară.
- Nu este necesar ca managerul de proiect să convoace o întâlnire pentru a ține un discurs intitulat „Filosofia mea de management”. În schimb, se pot convoca întâlniri frecvente de proiect în care managerul poate profita de orice deschidere rezonabilă pentru a vorbi despre cum vede el lucrurile.
- Din punct de vedere practic, aceasta este o abordare corectă și presupune discuții despre următoarele aspecte din partea managerului de proiect:
  - (1) Care sunt așteptările sale din punctul de vedere al organizării.
  - (2) Ce fel de responsabilități se așteptă ca să-și asume oamenii din proiect.
  - (3) Câtă libertate are un membru al echipei.
  - (4) Ce crede despre citirea ziarelor și periodicelor la locul de muncă.
  - (5) Opinia sa despre întârzieri.
  - (6) Punctul de vedere referitor la ascultarea muzicii sau despre navigarea pe internet.
  - (7) Care sunt probleme politice care pot afecta proiectul și aşa mai departe.
- Dacă managerul poate discuta cu diferite ocazii despre aceste subiecte, el va aduce la cunoștință personalului direcția generală a gândirii sale și va avea un grup de oameni care înțeleg mai bine locul lor și ceea ce li se cere.
- Dar **managerul de proiect nu** trebuie să uite, ca la rândul său, trebuie **să-i asculte pe ceilalți**.
  - Nu trebuie să se lase atât de vrăjit de propriile voastre cuvinte încât să nu solicite și să asculte punctele de vedere opuse din partea subordonaților.
  - Dacă oamenii de la întâlnirile de proiect vorbesc în cele din urmă și pun la îndoială observațiile și modul de a gestiona lucrurile de către managerului de proiect, acesta a reușit să-i câștige.
    - A înlocuit monologul cu dialog.
    - Ascultă observațiile și acționează rapid în consecință.
- Un ultim aspect privind comunicarea:

- Managerul de proiect trebuie să stabilească **definițiile de bază** pentru proiectul pe care îl coordonază și să insiste ca acestea să fie **aplicate în mod consecvent**.

#### **5.4 Asigurarea condițiilor de muncă**

- Indiferent de produsul care se construiește, oamenii din echipă sunt cei care îl realizează.
- (1) Sarcina managerului de proiect este să le ofere **mediul și instrumentele** de care au nevoie pentru a le îndeplini sarcina.
  - Managerul de proiect, trebuie să maximizeze şansele de succes.
- (2) Un lucru pe care îl poate face este să ofere cele mai bune facilități fizice pe care firma și le poate permite.
  - Poate că programatorii nu au nevoie de birouri cu mochetă și mobilier sofisticat, dar sigur au nevoie de liniște și intimitate.
  - Procesul de programare nu poate tolera nici un mediu zgomotos de fabrică, nici întreruperi constante.
    - Dacă s-ar putea solicita o taxă bănească pe fiecare distragere sau întrerupere cu care un programator se confruntă în cursul unei zile, s-ar putea strângă rapid suficienți bani pentru unele dintre remedii.
    - Luați în considerare efectul întreruperii unui programator în mijlocul unei bucăți complexe de cod.
      - Mai târziu, nu numai că trebuie să se întoarcă pentru a relua firul logic a ceea ce făcea, dar poate uita cu ușurință o parte din ceea ce a avut inițial în minte.
      - Rezultatul: o eroare (un bug).
      - Bug-ul duce la pierderea timpului propriu în timpul testării modulului; consumă timp suplimentar de calculator; poate apărea în timpul testelor de nivel superior când va cauza o pierdere disproportională a timpului oamenilor și a timpului computer.
    - Cealaltă problemă este dorința binecunoscută a programatorilor de a asculta muzică în timp ce lucrează.
      - De obicei, problema poate fi rezolvată prin dotarea posturilor de lucru cu echipamente personale de ascultare.
      - Această atitudine poate rezolva o mulțime de probleme interne.
  - (3) Un alt domeniu în care se poate ajuta întregul proces de programare este furnizarea **celui mai bun mediu de dezvoltare și testare posibil**.

- (4) Vorbind despre comunicare, trebuie încercat tot posibilul pentru a instaura un mediu în care oamenii vorbesc și spun când lucrurile nu sunt în regulă.
  - Managerul de proiect trebuie să depună tot efortul pentru ca să creeze un cadru specific pentru această activitate.
    - O modalitate de a face acest lucru este să intre în alertă pentru primul comentariu care sună ca o plângere sau o critică validă.
    - Să se concentreze asupra ei.
    - Să remedieze cât mai urgent orice a fost criticat.
    - Să se asigure că toată lumea știe că a reacționat pozitiv la critica cuiva.
  - Managerul de proiect trebuie însă să reacționeze la plângeri și critici într-un mod pozitiv.
    - În schimb, dacă reacționează negativ la prima critică, probabil că nu va auzi niciodată alta, constructivă sau de altă natură.
  - *Experiența lui Metzger: „Cu ceva timp în urmă am participat la o întâlnire de proiect în care șeful a discutat despre statut, a impus câteva reguli de bază noi, a pus întrebări și a terminat întâlnirea când nu mai existau întrebări. Imediat după aceea, s-a adunat un mic grup de programatori. Înălță biroul meu și am criticat conducerea în lipsă. L-am ascultat și apoi l-am întrebat pe cel mai vocal membru al grupului de ce nu a vorbit în ședință. El mi-a răspuns că oricum conducerea nu asculta niciodată, deci de ce să deranjeze. De câte ori se întâmplă asta în fiecare zi nu este dificil de ghicit, dar există o singură persoană care poate preveni acest lucru: managerul. El dă tonul.”*
- (5) Managerul ar trebui, de asemenea, să acționeze ca un tampon, luând măsurile necesare pentru a furniza oamenilor săi informațiile strict necesare și pentru a elimina pe cele ne necesare.
- (6) Managerul de proiect trebuie să acționeze ca un filtru la distribuția informațiilor pentru oamenii săi.
  - Orice organizație mare este afectată de birocrație, de prea multe hârtii destinate multor persoane greșite.
  - Acest lucru se întâmplă adesea pentru că managerul direcționează toate informațiile celorlalți oameni cu o notă „citește și transmite mai departe”.
  - Fie managerului îi este frică să ascundă informații de care ar putea avea nevoie membrii personalului, fie îi este frică să rateze ceva pe care unul dintre ei ar putea să-l înțeleagă.
  - Indiferent de situație, pur și simplu, nu este util ca toată lumea să citească totul.
- (7) Managerul trebuie să se asigure că în planificarea proiectului se acordă o atenție specială pentru utilizarea celor mai potrivite (și avansate) instrumente atât pentru management, cât și pentru programare.

- (8) Este nevoie de un management luminat pentru a-i încuraja pe oameni să citească literatura de specialitate, să caute și să facă săpăturile necesare pentru a găsi noi instrumente și noi moduri de a face treaba.
- Câteva **sfaturi practice** de urmat pentru **manager**:
  - (1) Să aibă contact direct cu oamenii din echipă în loc să le trimită documente sau e-mailuri scrise.
  - (2) Să manifeste tact și diplomatie în evidențierea greșelilor.
  - (3) Să acorde atenție simțului măsurii.
  - (4) Să nu supraîncarce persoanele capabile (programatori, manageri, personal), pentru a rezolva probleme.
  - (5) Să stabilească o perioadă generală de liniștee în organizația (de exemplu, de la 9:00 la 13:00), în care întâruperile sunt strict interzise.
  - (6) Să stabilească un cadru săptămânal / lunar cu oră generală fixă pentru diferite tipuri de întâlniri și discuții.
    - Întâlnirile ordinare trebuie planificate pe baza acestui cadru.
    - Desigur că există și excepții.

## 5.5 Atribuirea muncii

- Atribuirea (repartizarea) muncii este una dintre cele mai importante activități ale managerului de proiect, probabil în fruntea listei acestuia.
  - *Experiența lui Metzger: „Am asistat odată la un efort mare de propunere al cărui obiectiv era să câștige un job uriaș de programare sub contract cu guvernul federal. Propunerea în sine a implicat mai mult de cincizeci de persoane. Nimeni nu a fost numit manager de propunerii. Unele domenii de activitate au fost acoperite de trei sau patru persoane, fiecare dintre acestea crezând că el sau ea ar trebui să facă acea treabă. Alte domenii nu au fost acoperite deloc. Managerul care avea puterea de a corecta toate acestea avea reputația că nu atribuia sarcini specifice. A așteptat, de multe ori în zadar, ca un erou să iasă în față și să se ofere voluntar. Lucrurile pur și simplu nu funcționează așa. Persoana care este în măsură și a fost desemnată să dirijeze, trebuie să dirijeze.”*
- Există două moduri de atribuire a muncii:
  - (1) **Atribuirea muncii pe persoană**.
  - (2) **Atribuirea muncii pe domenii**.

### 5.5.1 Atribuirea muncii pe persoană

- Soluția este relativ **simplă**.

- Mai întâi, trebuie **atribuită o anumită persoană** pentru orice loc de muncă.
  - Ideea că un executiv ocupat poate fi **manager interimar** al unui proiect major în timpul său liber este **ridicolă**.
  - Este mai bine să fie numită chiar **o persoană puțin mai puțin calificată** la un **job cu normă întreagă** decât să atribui postul unui **manager interimar** care **nu poate dedica suficient timp** pentru a face față jobului.
- Presupunând că a o anumită persoană a fost desemnată ca **manager** al unui proiect sau ca **responsabil al efortului de propunere**. Această persoană:
  - (1) Mai întâi trebuie **să înțeleagă jobul** (lucrarea).
  - (2) Apoi să încerce **să partajeze lucrarea și să atribuie bucăți** din ea unor indivizi. Dar acest lucru **nu este suficient**.
  - (3) Trebuie realizată **o descriere a postului fiecărei persoane**. Nicio excepție! **Descrierile posturilor** trebuie **redactate în scris** nu verbal!
  - (4) Apoi **se distribuie tuturor o copie a tuturor sarcinilor**.
    - Primul lucru care se va întâmpla este că jumătate din echipă va veni să se plângă de faptul că activitatea altcuiva se intersectează cu activitatea sa.
    - Dacă managerul are noroc, cineva va semnala faptul că zona X nu este acoperită de nimeni.
  - (5) După câteva zile de astfel de plângeri în care oamenii au analizat suficient de profund sarcinile, managerul **stabilește o întâlnire** pentru a discuta despre problemele care au apărut.
  - (6) Apoi managerul **rescrie sarcinile**, le **distribuie** din nou și așteptă a doua rundă de explozii.
  - (7) Probabil fi nevoie de **câteva repetări**, unele dintre întâlniri pot fi inconfortabile, dar în curând **vor exista fișe de post** care **nu se suprapun** și care **acoperă integral ceea ce trebuie făcut**.

### **5.5.2 Atribuirea muncii pe domenii**

- O abordare alternativă a atribuirii muncii aflată la-ndemâna managerului este următoarea:
  - (1) Să aloce lucrările după subiect (tematică) și să lase pe fiecare să-și realizeze propria descriere a muncii.
  - (2) Apoi ajusteze aceste descrieri după cum consideră potrivit și să le distribuie tuturor.

### **5.5.3 Obiectivele atribuirii muncii**

- Nu contează ce abordare referitoare la atribuirea muncii adoptă managerul de proiect, atâtă timp cât sunt atinse următoarele obiective:
  - (1) Toate sarcinile proiectului sunt acoperite integral.
  - (2) Fiecare știe exact care este sarcina lui.
  - (3) Toată lumea știe exact care sunt sarcinile celorlalți.

## 5.6 Ore de lucru

- În ultimă instanță, atribuirea muncii se concretizează în alocarea de ore de lucru
- Există diferite tehnici pentru alocarea și controlul orelor de lucru.
- Tehnicile prezentate mai sus pot fi combinate în funcție de:
  - Specificul firmei.
  - Specificul angajatilor.
  - Specificul proiectului.

### 5.6.1 Alocarea orelor de lucru normale

- Orelle de lucru normale sunt alocate sarcinilor (lucrărilor), pe baza:
  - (1) Unei estimări a priori a dimensiunii sarcinilor.
  - (2) Unei planificări prestabilite.
- Fiecare programator, manager sau alte persoane implicate trebuie să țină evidență timpului de lucru personal.
- Managerii de nivel I și II trebuie să aibă evidență timpului de lucru pentru echipele lor ca bază pentru control.
- Managerul de proiect trebuie să aibă o imagine a timpului total de lucru petrecut pentru dezvoltarea proiectului ca bază pentru control.

### 5.6.2 Ore de muncă suplimentare

- Dacă managerul de proiect este relativ nou în domeniul programării computerelor, poate că încă nu a luat parte la proiecte de panică în care totul este întârziat și la care conducerea a trebuit să recurgă la **remediu final: planificarea de ore de muncă suplimentare**.
- Este greu de demonstrat că există o **risipă de resurse** mai gravă decât aceasta.
  - Există o tentație firească de a crede că lucrând un anumit grup de oameni cu 25% mai multe ore pe săptămână, se va lucra mai mult cu 25%;

- Unii manageri chiar cred că eforturile suplimentare accidentale ajută la întărirea spiritului de echipă și că acest lucru dezvoltă un puternic sentiment de camaraderie.
- Dar **experiența** demonstrează exact contrariul:
  - În primul rând, cu 25% ore în plus se pot produce cu ușurință cu 10% mai puțină muncă, sau cel puțin mai puțină muncă utilizabilă.
  - Având mai multe ore de lucru în fiecare zi pentru o perioadă extinsă de săptămâni sau luni, la majoritatea oamenilor se va instala pur și simplu o nouă rutină, de a lucra mai încet.
    - Munca lor va deveni în timp mai neglijentă.
  - În prima săptămână, bineînțeles, poate fi realizată o muncă suplimentară, și poate chiar a doua și a treia, dar după aceea poate deveni un efort pierdut.
    - Oamenii își încetinesc inconștient eforturile de a umple timpul programat.
  - Chiar și munca suplimentară realizată în prima sau a doua săptămână, de altfel benefică, nu va fi nici pe departe proporțională cu costul ei – fie costul ei în dolari, fie costul ei moral, deoarece viațile private și de familie ale membrilor echipei sunt inevitabil afectate.
  - Și în ceea ce privește acel sentiment de camaraderie, acesta evoluează curând în negativism: al naibii de client pentru că a insistat asupra acelui termen de livrare; al naibii de conducere pentru că a fost de acord cu asta; la naiba cu timpul petrecut la calculator; la naiba cu totul!
- De fapt, se discută despre orele suplimentare planificate. Această soluție pur și simplu nu funcționează bine.
- Singura șansă reală ca orele suplimentare să funcționeze bine este atunci când sunt făcute voluntar și pe perioade scurte de timp.

### 5.6.3 Tehnica Flexy-Time

- Mai este ceva de luat în considerare cu privire la orele de lucru.
- Diverse companii au **experimentat ideea** de a **lăsa angajații să-și stabilească propriul program de lucru**.
  - (1) În general, **timpul total** care trebuie lucrat în fiecare zi **este stabilit**, dar **orele de începere și de sfârșit** sunt lăsate la latitudinea lucrătorului.
  - (2) Acest lucru poate fi **liberalizat în continuare** prin stabilirea unui număr de ore pentru, de exemplu, o săptămână întreagă și lăsând planificarea pe zile sau ore la latitudinea individului.
- Aceasta este **Tehnica Flexy-Time**.

- Poate că aceste noțiuni sunt prea liberale pentru anumiți manageri, dar ideea în sine a zilei de lucru flotante sigur nu este.
  - Conceptul a fost încercat și s-a demonstrat că poate funcționa.
  - Are un **avantaj** evident:
    - Satisfacerea nevoilor și înclinațiilor individuale precum și acelora legate de ceasurile biologice ale fiecărui angajat, ce ar trebui să conducă la o satisfacție crescută la locul de muncă.
  - Există însă și **dezavantaje** evidente:
    - De exemplu, cum se poate convoca o întâlnire de departament când nu se știe niciodată cine va fi prin preajmă la o oră dată?
    - Și ce se poate spune despre de frustrările programatorului A care trebuie să vorbească cu programatorul B, dar orele alese de A sunt devreme în ziua și cele ale lui B târziu în noapte?
- O modalitate evidentă de a experimenta ideea ca fiecare să-și stabilească propriul program de lucru, este să se înceapă cu puțin, să se analizeze cât de bine funcționează și să se adapteze pe măsură ce avansează.
  - (1) Se poate începe prin a limita intervalul orar în care le este permis lucrătorilor să-și flexibilizeze programul.
  - (2) Se poate stabili în fiecare zi, să zicem între orele 1:00 P.M și 16:00, ca toată lumea să fie la serviciu. Această perioadă poate fi foarte bine corroborată cu şablonul general de timp fix al departamentului.
  - Aceasta ajustare le va permite unora să lucreze de la 7:00 A.M. până la 16:00, altora de la 13:00. până la 21:00, iar altele de la 10:00. până la 7:00 P.M.
- Sună ca o durere de cap pentru manager? Trebuie să încerce!

#### **5.6.4 Tehnica termenului prestabilit (dead-line)**

- În pasul următor, orele de lucru ar putea fi eliminate ca măsură a muncii sau a valorii.
- Mijlocul de măsurare sau de control ar putea fi ca angajatul să finalizeze o anumită sarcină până la o dată predeterminată, adică un termen prestabilit.
  - Aceasta este **tehnica termenului prestabilit** (limită).

#### **5.7 Adăugarea mai multor persoane**

- Așa cum recurgerea la ore suplimentare este, în general, **risipitoare, nu** este util ca regulă, ca **proiectul să fie încărcat cu mai mulți oameni** pentru a încerca să se rezolve problemele care sunt rezultatul în primul rând, a unei proaste planificări.

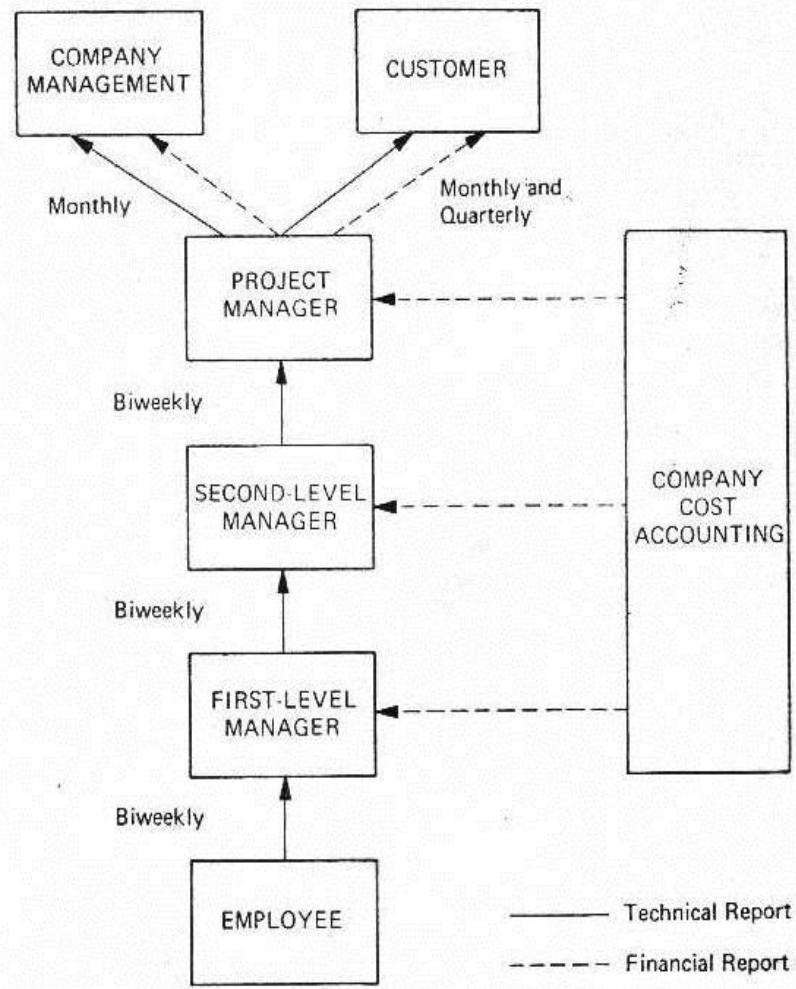
- Managerul de proiect trebuie să se ferească de credință greșită că ceea ce lipsește din timpul calendaristic pentru respectarea unor termene imposibile, poate fi compensat cu alocarea de lucrători suplimentari. Pur și simplu nu funcționează.
- Brooks simplifică excesiv și oarecum în mod revoltător, atunci când postulează **Legea lui Brooks**: „*Adăugarea de forță de muncă la un proiect software întârziat îl întârzie și mai mult. Nașterea unui copil,*” spune Brooks, „*durează nouă luni, indiferent câte femei sunt însărcinate*”.
- Există **două alternative rezonabile** pentru a adăuga mai mulți oameni atunci când proiectul este întârziat în raport cu planificarea inițială:
  - (1) **Reprogramare**. Acest lucru, desigur, va face clientul să riposteze, iar aceasta la rândul ei va face conducerea firmei să se sesizeze.
  - (2) Se poate aranja furnizarea eventual a unei versiuni intermediare, **incomplete** a sistemului în dezvoltare.
    - Dacă prima livrare poate fi făcută în momentul în care trebuia să se livreze produsul final, acest fapt ar putea fi de ajutor; doar că atunci versiunea finală trebuie reprogramată.
- Dacă managerul de proiect se implică în **reprogramarea** și **livrarea de versiuni intermediare** ale sistemului, trebuie să fie cât mai sigur că de data aceasta, poate să efectueze livrările **așa cum a promis**.
  - Va exista un impuls teribil de convingător de a livra cât mai curând posibil și este foarte probabil ca noile date oferite să fie din nou prea optimiste.
  - Cel mai bine pentru managerul de proiect aflat în această situație ingrată este:
    - Să admită și să recunoască orice planificare sau execuție proastă pentru care a fost responsabil până în acum.
    - Dar să se asigure că nu va mai repetata aceste greșeli.
- Clienții care tipă sunt un coșmar, la fel și șefii care tipă.
  - Nu trebuie lăsate tipetele lor să afecteze managerul, mai ales atunci, când acesta simte că are dreptate; altfel, îi va auzi tipând din nou peste câteva luni.

## 5.8 Raportarea stării tehnice a proiectului

- Oricine face o treabă pentru oricine altcineva trebuie să **comunice** cumva **cum merg lucrurile**.
- De obicei, acest lucru se realizează prin două modalități:
  - (1) **Rapoarte scrise**.
  - (2) **Revizii orale**.
- Să le luăm în considerare fiecare pe rând.

### 5.8.1 Rapoarte scrise

- În orice context, **răportarea oricărui stării** presupune **existența un plan de referință pentru raportare**.
- Câteva **considerente generale**:
  - (1) Prima cerință pentru raportarea efectivă a stării este aceea că trebuie să existe un **plan și repere (termene)** în funcție de care se măsoară progresul.
  - (2) Un al doilea aspect este că toate rapoartele trebuie să fie adaptate pentru a se potrivi cu nivelul de management pentru care sunt destinate.
    - Rapoartele trebuie adaptate atât în ceea ce privește **conținutul**, cât și **frecvența**.
- O **schemă rezonabilă de raportări** pentru un proiect poate funcționa astfel:
  - (1) **Persoanele fizice**, de exemplu, programatorii, raportează săptămânal managerilor lor de prim nivel cu privire la starea tuturor sarcinilor sau pachetelor de lucru care le-au fost atribuite.
  - (2) **Managerul de nivel I**, după ce a primit intrări de la oamenii săi, raportează **managerului de nivel II** starea sarcinilor selectate;
    - Adică raportează acele sarcini afișate ca repere pe diagrama cu bare a misiunii sale de muncă.
    - Un manager de nivel întâi responsabil pentru, să zicem, cincisprezece sarcini, ar putea raporta unui manager de nivelul doi doar patru sau cinci dintre acestea.
  - (3) **Managerul de nivelul doi** raportează **managerului de proiect** starea sarcinilor de referință din diagramele sale.
    - În mod efectiv, ceea ce primește **managerul de proiect** nu este o **însumare netă a tuturor lucrărilor individuale ale proiectului**.
    - În loc să transmită toate datele posibile, fiecare manager acționează ca un filtru și trimite doar ceea ce este important.
  - (4) La rândul său, **managerul de proiect** transmite un raport condensat **conducerii și clientului** (vezi Fig. 6.5.8.1.a.).



**Fig. 6.5.8.1.a.** Schemă de raportare a stării proiectului

- Toate rapoartele din acest lanț trebuie să urmeze formate similare, compatibile, stabilite de **Planul de documentare** bazat pe **sistemul de documentare**.
- În acest fel, de proiect solicită ocazional mai multe date de la un manager, acesta îi poate oferi rapoartele mai detaliate primite de la subordonații săi, iar rapoartele detaliate vor avea sens pentru el.
- Se recomandă utilizarea următorului **ghid pentru redactarea unui raport**:
  - (1) În **secțiunea unui raport** în care se discută anumite probleme, **se precizează aceste probleme**, într-o manieră adecvată **nivelului cititorului**.
  - (2) **Problemele** se expun în funcție **de prioritățile lor**.
    - Raportorul trebuie să se asigure că cititorul raportului știe care sunt problemele pe care le consideră cele mai importante.
  - (3) **Raportorul** trebuie să încerce întotdeauna **să indice soluții potențiale de rezolvare** și să precizeze **stadiul încercărilor de a găsi aceste soluții**.

- El trebuie să ofere cititorului sentimentul că este implicat și că lucrează la problemă (dacă este).
- (4) Un **principiu de bază** al **tehnologiei de raportare** se referă la structurarea tuturor activităților din proiect în bucăți discrete.
  - Raportările din rapoartele de stare trebuie să se refere la acele bucăți discrete.
- (5) **Nu trebuie lăsat nimeni** să producă **rapoarte calitative** pline de aprecieri sau de „proccente de realizare” fără sens.
- (6) Trebuie insistat ca **raptorii** să raporteze în **termeni cantitativi**.
  - *Experiența lui Metzger. Îmi amintesc de un proiect în urmă cu mulți ani în care am raportat parțial completând o diagramă cu bare. Când o bară a devenit complet neagră, acea treabă era teoretic făcută. Din păcate, am rămas fără bare înainte ca treaba să fie terminată. Am constatat că completarea barelor respective nu ne spunea nimic. Ele nu erau nimic mai mult decât decorațiuni deprimante de perete într-o cameră de control al stării proiectului.*

## 5.8.2 Revizii orale

- **Reviziile orale** au fost discutate în capitolul anterior.
  - Aici, totuși, trebuie subliniată **utilitatea recenziilor orale** ca parte a **activității managerului de proiect**.
- **Reviziile orale** realizate în cadrul proiectelor au multe **avantaje**:
  - (1) Reviziile în sine, oferă lucrătorilor un sentiment de apartenență la ceva important, situat în afara cabinelor lor cubice de unu pe doi metri patrăi.
  - (2) Îi ajută pe oameni să înțeleagă rolul pe care îl joacă munca lor în jobul integral.
  - (3) Provoacă o ruptură de la rutina curentă.
  - (4) Oferă oportunitatea pentru descoperirea unor probleme și abordarea unor soluții mai bune.
- Proiectul va beneficia enorm dacă recenziile sunt astfel configurate încât să ofere oamenilor ocazia de a se face auziți.
  - S-ar putea chiar să se ajungă ca dezvoltatorul să fie privit ca o ființă umană, mai degrabă decât ca un ananim reclus care stă în biroul din colț cu ușa închisă.

## 5.9 Raportarea stării financiare

- Rapoartele financiare sunt adesea generate de către departamentul de contabilitate care este separat de managementul proiectului.
  - Aceste rapoarte pot fi trimise managerului de proiect, sau fiecare dintr-un manager din subordinea acestuia.
- Indiferent de caz, managerul de proiect trebuie să se asigure în timpul derulării proiectului, că sarcinile sau pachetele de lucru care urmează să fie utilizate ca bază pentru raportarea financiară se referă la aceleași sarcini utilizate pentru raportarea tehnică.
- Managerul de proiect trebuie să fie întotdeauna capabil să echivaleze rapoartele tehnice și financiare fără a fi nevoie de un exercițiu masiv de conversie pentru a afla cum se leagă cele două rapoarte.

## 5.10 Instruire

- Fiecare manager are responsabilitatea de a-și instrui oamenii.
  - Această instruire este total diferită de orice altă activitate de formare care trebuie făcută conform contractului.
- Managerul trebuie să caute să ridice nivelul de competență și înțelegere al fiecărui individ din organizație.
  - În caz contrar, organizația stagnează.
- Când vremurile sunt grele și costurile generale ale unei organizații trebuie reduse, **educația** este una dintre primele candidate la reduceri de cheltuieli. Este considerată un lux.
  - Cursurile sunt anulate pentru că se adaugă la cheltuielile generale. Si totuși aceiași oameni care ar fi participat la acele cursuri stau acum, decontându-și timpul pe unul din tipurile de cheltuieli generale acceptate, făcând puțin sau nimic.
  - Acei oameni inactivi trebuie aduși la programe de educație.
    - Desigur, dacă politica firmei este de a scapa de oameni de îndată ce nu mai au de lucru, argumentul este academic.
    - Există mai multe **tipuri de formare** pe care un **manager** ar trebui să le ofere și există **o serie de moduri de a le oferi**.

### 5.10.1 Instruirea personalului tehnic

- Să începem cu oamenii tehnici, de exemplu, cu **programatorii**.
- **Programatorii** ar trebui să fie instruiți în, sau cel puțin familiarizați, cu limbi de programare și sisteme de calcul, altele decât cele de interes pentru proiectele lor curente.

- Cum altfel vor putea ei crește?
- Cum ar putea ei oferi soluții tehnice alternative dacă tot ce știu este calculatorul x și limbajul y?
- Acești oameni ar trebui să primească timpul necesar și încurajarea să participe la cursuri formale, să se aboneze și să citească literatură tehnică de specialitate și să se întâlnească cu omologii lor din alte proiecte.
  - Acesta ultim aspect este adesea greu de înghițit de către un manager. Este ca și cum ar încuraja pauzele de cafea. Dar ce răsplătă! Ceea ce rezultă de cele mai multe ori din astfel de discuții, este fie o idee tehnică nouă, fie identificarea unei capcane de evitat.
- **Programatorii noii** au nevoie de o **atenție specială** până când se vor evidenția competențele fiecărui.
- Managerii, în special managerii ecologici, tind adesea să accepte noii recruți ca și profesioniști, cu experiență, fără a cere vreo dovdă.
  - Nu este întotdeauna adevărat! S-ar putea să apară surprize!
- Cum se rezolvă astfel de situații și cum se aduc lucrurile pe făgașul lor normal?
  - Răspunsul este: prin efectuarea de proceduri de rutină de parcurgere structurată a codului și prin utilizarea tehnicilor de programare peer (între egali) pentru fiecare programator din proiect.
  - Cercetările lui **Mills** și a altora au arătat că această abordare este extrem de valoroasă, atât ca tehnică de control al calității, cât și ca mai ales mecanism de instruire.
- Un tip important de instruire pentru toți oamenii din proiect (inclusiv pentru manager) este rotația prin alte locuri de muncă decât cele normale.
  - Adesea, acest lucru se poate face fără o investiție enormă de timp, iar câștigul poate fi substanțial.
    - Acest lucru se poate întâmpla în cadrul aceluiași proiect sau în timp în timpul unor proiecte diferite.
  - Rotația sau antrenamentul încrucișat este utilă și pentru oamenii netehnici.
    - Spre exemplu ce ar fi dacă s-ar organiza un curs în fundamentele analizei sistemului, proiectării și programării pentru secretare și dactilografe.
    - Cu siguranță le-ar face plăcere să aibă habar despre toate acele concepte sofisticate care apar în materialele cu care lucrează și ar face desigur o treabă mai bună transcriindu-le.
  - Nu există nimeni în proiect care să nu beneficieze de cel puțin o expunere oarecare, la ceea ce se întâmplă în afara propriului loc de muncă și acest lucru va aduce, la rândul său, beneficii proiectului.

- Mare atenție însă: legile federale și legile muncii din unele state (este vorba despre SUA) restricționează sever tipurile și durata rotațiilor care pot fi utilizate în mod legal.
  - Nu pot fi, de exemplu, interschimbați „profesioniști” și „clericii”.
  - În astfel de situații trebuie consultați avocații companiei la ce este permis în zona respectivă.
    - Nu trebuie întrebați dacă se poate face; este cel mai ușor să spui nu.
    - Trebuie informați că managerul intenționează să facă acest lucru și întrebă cum să o facă în mod legal.

## **5.10.2 Instruirea managerilor**

- O categorie specială o reprezintă managerii din proiect.
  - (1) Ei au nevoie de asistență zilnică din partea managerului de proiect.
  - (2) De asemenea, ei trebuie să participe la cursuri și seminarii de management.
  - (3) Au nevoie de cursuri de actualizare tehnică pentru a ajuta la evitarea erodării (obsolescence).
  - (4) Managerii care lucrează în cadrul proiectului trebuie să vadă în managerul de proiect un bun exemplu de gestionare.
    - Dacă managerul de proiect își planifică prost activitatea (și se bazează pe ore suplimentare excesive pentru a îndeplini propria muncă), el oferă, prin exemplul personal, un model de formare specific unui management neglijent.
    - Poate cel mai important lucru dintre toate, este faptul că managerul de proiect trebuie să reunească personalul întregului proiect suficient de des, pentru ca toată lumea să fie la curent cu starea proiectului și cu încadrarea în planificare.
    - Sunt probabil greu de estimat beneficiile deosebite pe care le va dobândi proiectul prin faptul că toată lumea înțelege ce se întâmplă și unde se încadreză fiecare cu activitatea personală.

## **5.11 Evaluare și consiliere**

- (1) Managerul de proiect, are o influență enormă asupra vieții oamenilor săi.
- (2) Cât de bine este plătită o persoană din echipă este desigur, foarte important, dar la fel de importantă este și capacitatea managerului de a ajuta fiecare persoană din proiect să-și găsească un loc de muncă în care se simte împlinit.
- (3) Desigur, multe lucruri dintr-un proiect pot fi proaste, dar dacă managerul de proiect reușește să delimitizeze o lucrare care să atragă un individ și care să se încadreze în limitele capacităților sale, toate celelalte probleme dispar.

- Nietzsche spunea: „Cine are un **ce** pentru care să trăiască, poate suporta aproape orice **cum**” (“He who has a **why** to live for, can bear almost any **how**”).
- (4) Managerul de proiect trebuie să convină cu fiecare dintre angajații săi asupra unei sarcini adecvate și a unei planificări pentru realizarea acesteia.
  - Din nou, simplul act de a redacta o descriere a sarcinilor oamenilor și de a obține acordul acestora poate fi extrem de util.
- (5) Adesea, oamenii pot fi lăsați și chiar încurajați să-și stabilească propriile lor programe.
  - Managerul va fi surprins cât de mult vor munci pentru a respecta un termen limită stabilit de ei însăși.
- (6) Când unul dintre oamenii ieșe din cursul normal al lucrurilor sau face o treabă proastă, trebuie atenționat de către managerul de proiect în cel mai bun mod posibil.
  - Acest lucru poate fi uneori atât de dificil încât unii manageri evită să îl facă până când sunt forțați.
  - Când se ajunge într-o astfel de situație, lucrurile sunt de obicei în stadiu de criză.
- (7) O modalitate de a se evita problemele mari este aceea ca managerul să subdivize suficient activitățile și să planifice suficiente puncte de control, astfel încât ratarea unuia să fie un semnal și nu o catastrofă.
- (8) Există manageri care îți spun ce ai făcut greșit, dar nu și ce ai făcut bine.
  - *Experiența lui Metzger.* „Îmi amintesc de un manager care s-a priceput să-mi sublinieze greșelile, dar când în cele din urmă l-am întrebat într-o zi dacă am făcut ceva pentru el de care să fie mulțumit, a fost uimit. Desigur, a fost mulțumit. Nu mi-am dat eu seama că orice nu a fost criticat de el a fost, prin definiție, în regulă? Acest om, care era un manager bun în cele mai multe privințe, pur și simplu nu înțelegea că oamenii au nevoie de o vorbă bună din când în când. Unii oameni, în absența vreunui vești bune, tind să teamă de ce e mai rău.”
- (9) Succesul oricărui manager ar trebui evaluat prin:
  - Cât de bine încurajează creșterea.
  - Cât de corect este răsplătită munca grea.
  - Cum este tratată incompetența.
- (10) Un manager bun nu va ezita să promoveze un subordonat la nivelul propriu al managerului.
- (11) Un manager bun nu va ascunde în mod egoist un angajat cheie în cadrul organizației;
  - În schimb, managerul va promova angajatul deși riscă să-l piardă în favoarea unui alt grup.

- (12) Un manager bun nu va transfera niciodată un „o problemă” (problem child) altcuiva fără un avertisment complet.
- (13) Ceea ce este cu adevărat esențial este modul în care managerul tratează și premiază munca bine făcută și competența.

## 5.12 Menținerea unei atmosfere sănătoase (Sanity Maintenance)

- Multii manageri intră într-o mulțime de necazuri pentru că nu au putut spune nu.
  - Cuvântul **nu** este negativ, până la urmă, și niciun Tânăr manager nu vrea să pară negativ.
    - Orice dorește șeful (sau clientul) trebuie îndeplinit.
  - Toate cărțile despre „gândirea pozitivă” (positive thinking) și „gândirea mare” (thinking big) spun că se poate face orice în această direcție.
    - De exemplu, se poate șterge cuvântul „nu” din vocabular.
      - E o mare greșeală!
    - Există foarte multe lucruri care sunt imposibile și multe altele care sunt nerezonabile.
- Adevăratul gânditor pozitiv este persoana care poate rezolva lucrurile și poate distinge între rezonabil și nerezonabil.
  - Au existat nenumărate dezastre în afacerile de programare, deoarece cineva a permis să fie presat ca să depună efort pentru ceva despre care credea că este imposibil.
- (1) Uneori este o presiune delicată.
  - Managerul se află într-o poziție dificilă și are nevoie de un anumit angajament din partea persoanei pentru a se salva.
- (2) Uneori presiunea este aplicată prin eroziune.
  - I se cere unei persoane să cedeze câte puțin la un moment dat, în rate ușoare, iar la sfârșit când aceasta realizează ce a făcut, este prea târziu.
- (3) Alteori persoanei i se vorbește rapid și este ademenită să joace jocul RAM (Repeat After Me).
  - Managerul îi spune persoanei ceea ce ar vrea el să audă și singura treabă a persoanei este să fie de acord.
- (4) În alte cazuri, se aplică metode mai dure (strong-arm methods).
  - Persoana este făcut să perceapă că viitoarele sale promovări sau chiar slujba actuală sunt în pericol.
- Sunt foarte multe situații în care se aplică astfel de tactici.

- Poate că ceea ce este în discuție este o estimare pe care persoana a trimis-o șefului și este prea mult pentru ca șeful să o înghită.
- Sau poate se cere unei persoane să accepte o muncă suplimentară cu care aceasta simte că nu se va putea descurca.
- **Bill Weimer** de la IBM a fost responsabil pentru multe cursuri de formare inovatoare atât pentru oamenii de management, cât și pentru cei tehnici. El a spus:
  - „*Am constatat că oamenii tehnici, în general, erau de fapt foarte buni la estimarea cerințelor și calendarelor proiectelor. Problema pe care o aveau a fost să-și apere deciziile; trebuiau să învețe cum să-și susțină punctul de vedere.*”
- **Charles P. Lecht** vorbește în același context despre refuzul imposibilului:
  - *În egală măsură responsabil pentru inițierea unui proiect cu eșec predefinit... este managementul care insistă să aibă angajamente fixe din partea personalului de programare înainte ca acesta din urmă să înțeleagă pentru ce sunt aceste angajamente.*
- Prea des, conducerea nu realizează că, cerând personalului „imposibilul”, personalul va simți obligația de a răspunde afirmativ din respect, frică sau loialitate greșită.
  - A spune „nu” șefului necesită frecvent curaj, înțelepciune politică și psihologică și maturitate în afaceri, atribute care din pacate, vin după multă experiență.
- Există câteva modalități care ar putea ajuta managerul de proiect în a menține echilibrului în afacerea complicată a managementului programării:
  - (1) Să se forțeze periodic (poate o dată pe zi) să facă câțiva pași înapoi și să privească la job în ansamblu, nu la detaliu. Să simplifice lucrurile.
  - (2) Să încerce să pună lucrurile în perspectivă.
  - (3) Să se izoleze față de locul său de muncă fizic – fără telefon, fără mapă, fără documente.
  - (4) Să enumere toate lucrurile pe care trebuie să le facă și apoi să decidă care dintre ele sunt cu adevărat importante.
  - (5) Dacă există mai mult decât poate face, o parte trebuie delegată altcui.
  - (6) Trebuie stabilite acele elemente care ar putea fi tăiate de pe listă și pe care nu le va putea face niciodată (există întotdeauna unele dintre acestea).
  - (7) Dacă va aplica aceste precepte destul de des, managerul va începe să privească munca mai rațional.
  - (8) Va avea o conștientizare constantă a cât de mult are de făcut și, prin urmare, cu cât de mult se poate încărca.
- În atribuirea priorităților sarcinilor pe care managerul trebuie să le facă, „problemele oamenilor” trebuie să fie întotdeauna pe primul loc.

- Oamenii din echipă nu trebuie lăsați niciodată să simtă că sunt pe locul doi.
- Dacă managerul pierde loialitatea și respectul oamenilor săi, este mort.

### **5.13 Niveluri de management**

- În cele prezentate până la acest moment nu s-au făcut prea multe distincții între diferitele niveluri de management.
  - Munca este în esență aceeași, indiferent de nivel.
- Ceea ce variază cu adevărat este raportul dintre implicarea tehnică și cea non-tehnică.
  - Acest raport scade pe măsură ce urcăm în lanțul de management.
  - (1) Un manager de nivel întâi este în mod normal foarte direct implicat în munca tehnică pe care o fac oamenii lui.
  - (2) Implicarea tehnică a unui manager de nivelul doi este mai generală.
    - Această poziție necesită mai mult timp decât cel al managerului de prim nivel în chestiuni financiare, propuneri, planificare, probleme de personal și altele asemenea.
  - (3) Și astfel avansând în ierarhie, la un anumit nivel, managerul este preocupat mult mai mult de deciziile generale de afaceri decât de deciziile tehnice detaliante.
- Din toate acestea apare o problemă dificilă:
  - Cum simte managerul de nivel superior ceea ce se întâmplă din punct de vedere tehnic?
  - Cum luptă managerul învecirea (obsolescence) tehnică și poate avea încredere că lucrurile merg bine?
- Există câteva răspunsuri parțiale la aceste întrebări.
  - (1) În primul rând, un manager ar trebui să aloce o parte semnificativă de timp actualizării tehnice, citind specificațiile și audiind briefing-urile subordonăților săi.
    - Dar el trebuie să reziste nevoii de a fi manipulator de biți și trebuie să lase lucrările tehnice detaliante celor mai bine calificați să o facă.
  - (2) În al doilea rând, managerul ar trebui să stabilească verificări tehnice și analize pentru a se asigura că munca tehnică se desfășoară în mod corespunzător.
    - O astfel de verificare o constituie organizarea separată a analizei respectiv designului, discutate mai devreme.
    - O alta este organizarea unui grup de testare separat, deja descrisă.

- Și încă o modalitate este utilizarea pe scară largă a expunerilor sau inspecțiilor structurate pentru a asigura o examinare detaliată a tuturor elementelor dezvoltate în cadrul proiectului.
- (3) O a treia modalitate ca managerul să se mențină pe linia de plutire din punct de vedere tehnic este revenirea periodică la munca tehnică.
  - În unele organizații, este descurajată practica trecerii între joburile de management și cele tehnice, dar dacă se poate realiza, ea poate face minuni pentru competența tehnică a managerului și poate spori considerabil încrederea în a putea gestiona următorul job.
- (4) În sfârșit, managerul trebuie să citească în permanentă literatura din domeniul său și să participe la cursuri ori de câte ori este posibil.
  - De cele mai multe ori, acest lucru va trebui făcut din propriul său timp.

## **Exercițiul #10**

1. Ce este programarea structurată? Care sunt obiectivele și avantajele ei?
2. Descrieți principalele caracteristici ale programării orientate pe obiecte (OOP), proiectării orientate pe obiecte (OOD) și analizei orientate pe obiecte (OOA)? Cum sunt ele legate?
3. Ce fel de modalități de organizare cunoașteți? Descrieți principalele lor caracteristici, avantaje, dezavantaje și domeniul de aplicare.
4. Ce este organizarea convențională? Desenați organograma organizației convenționale. Detaliați fiecare componentă a structurii.
5. Care sunt sarcinile Grupului de analiză și proiectare în timpul fazei de programare? Descrieți-le.
6. Ce sunt inspecțiile și reviziile structurate? Descrieți pașii principali ai acestei activități.
7. Care sunt sarcinile Grupului de Programare în timpul Fazei de Programare? Descrieți-le.
8. Ce fel de modalități de integrare cunoașteți? Descrieți avantajele, dezavantajele și tehnicele de implementare ale acestora. Care este conținutul specificației testului de integrare?
9. Care sunt sarcinile Grupului de testare și ale Grupului de personal în timpul fazei de programare? Descrieți-le.
10. Ce este organizarea de tip echipa programatorului șef? Desenați diagrama echipei programatorului șef. Cum funcționează?
11. Ce este Change Control? Care sunt documentele de referință afectate de Controlul schimbărilor? Care sunt procedurile de control al schimbării? Cum funcționează?
12. Ce instrumente de programare cunoașteți? Descrieți unele dintre ele subliniind rolul lor în faza de programare, de exemplu, executive de testare sau biblioteca de proiecte.
13. Care sunt sarcinile managerului de proiect în faza de programare?

14. Ce fel de modalități de atribuire a muncii cunoașteți? Descrieți principalele lor caracteristici. Care sunt obiectivele atribuirii muncii?
15. Ce tehnici de alocare și control al orelor de lucru cunoașteți? Descrieți-le subliniind avantajele și dezavantajele lor.
16. Care sunt principalele modalități de raportare a stării tehnice a proiectului? Detaliați-le.
17. Ce știți despre instruire? Ce tipuri de instruire cunoașteți? Descrieți-le.
18. Explicați importanța și descrieți conținutul următoarelor atribuții a Managerului de Proiect: evaluare și consiliere, întreținerea unei atmosfere sănătoase.
19. Care sunt principalele niveluri de management și care sunt principalele caracteristici și diferențieri ale acestora?

## **Capitolul 7. FAZA DE TESTARE A SISTEMULUI**

### **1. Testarea sistemului**

- 1.1 Specificații de testare a sistemului
- 1.2 Testerii
- 1.3 Condiții de testare
- 1.4 Efectuarea Testelor

### **2. Instruirea clienților**

- 2.1 Utilizarea sistemului
- 2.2 Întreținerea sistemului

### **Exercițiul #7**

## 7. FAZA DE TESTARE A SISTEMULUI

- Faza de testare a sistemului este faza cel mai greu de vândut.
- Atât managerii, cât și programatorii îi rezistă și totuși, este la fel de critică ca oricare altă perioadă a proiectului.
- Obiectivele fazei de testare a sistemului sunt:
  - (1) Obiectivul principal este:
    - Să supună produsele programatorilor unui set amănunțit de teste care nu au fost concepute și nici executate de programatori.
    - Testele trebuie rulate în condiții cât mai apropiate posibil de mediul real de funcționare al sistemului, cu un minim de simulare.
  - (2) Un al doilea obiectiv este de a începe instruirea clientului pentru a fi gata să preia noul sistem.

### 1. Testarea sistemului

- Unul dintre motivele pentru care managerii rezistă uneori la testarea sistemului este că consideră că este o întârziere în drumul către acceptarea și livrarea produsului.
- Dar ei învață inevitabil că testarea sistemului nu poate fi refuzată.
  - Erorile care nu au fost găsite deoarece testul sistem este omis vor apărea mai târziu, fie în timpul testării de acceptanță, fie după ce programele au fost acceptate și au devenit operaționale.
  - Costul unei probleme depistate cu întârzierea este foarte mare. El se măsoară în termeni de:
    - (1) Nemulțumirea sau dezgustul clientului.
    - (2) Oportunități pierdute pentru un job ulterior.
    - (3) O reputație pătată a dezvoltatorului.
    - (4) Un produs peticit, precum și forță de muncă, timp și bani suplimentari irosiți.
- Dezvoltatorul trebuie să fie pregătit pentru testarea sistemului atunci când programatorii își livrează produsul.
  - (1) Specificația de testare este scrisă
  - (2) Timpul calculator pentru testare programat aproximativ.
  - (3) Biblioteca instalată și pregătită.
  - (4) Oamenii de testare sunt pregătiți.

## 1.1 Specificația de testare a sistemului

- Modelele specificațiilor de testare sunt incluse în Planul de documentație (fig.7.1.1.a).

### SPECIFICAȚIA DE TESTARE A SISTEMULUI (MODEL)

**DEPARTAMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBAT:**

**DATA REALIZĂRII:**

**REALIZAT DE :**

#### SECȚIUNEA 1: OBIECTIVE

Există patru seturi separate de specificații de testare: Integrare, Sistem, Acceptanță și Testare site. Modelele pentru toate cele patru seturi sunt identice, cu excepția faptului că trebuie inserat calificativul corespunzător („integrare”, „sistem”, „acceptanță” sau „site”). Conținutul specificațiilor poate varia, desigur, considerabil, deși două dintre ele (acceptanță și site-ul) vor fi adesea identice. Această secțiune, intitulată Obiective, trebuie să servească în fiecare caz ca o introducere a documentului, descriind intenția acestuia și modul în care urmează să fie utilizat.

#### SECȚIUNEA 2: DOCUMENTE APLICABILE

#### SECȚIUNEA 3: TESTAREA DE (INTEGRARE, SISTEM, ACCEPTANȚĂ, SITE). GENERALITĂȚI

3.1. Filozofia testării

3.2. Obiective generale

3.3. Proceduri generale

3.4. Criterii de success

## SECȚIUNEA 4: MATRICEA DE ACOPERIRE

O diagramă care listează de-a lungul axei verticale zonele care trebuie testate și de-a lungul axei orizontale numărul (numerele) cazului de testare care acoperă fiecare zonă. Când este complet, acest grafic reprezintă o referință încrucișată între toate zonele care trebuie testate și toate cazurile de testare care acoperă acele zone.

**Fig.7.1.1.a.** Model generalizat de specificație de testare

- O **specificație de testare** descrie:
  - (1) **Obiectivele testării**.
  - (2) **Modul de abordare**.
  - (3) O **matrice de acoperire** a cerințelor care arată corespondența dintre fiecare cerință pe care sistemul de programe trebuie să o îndeplinească și cazul de testare corespunzător.
    - Această matrice este foarte importantă; pentru orice cerință de sistem sau funcție dată, ar trebui să fie ușor de localizat în această matrice toate cazurile de testare relevante.
    - Dacă se întâmpină probleme la scrierea acestei matrici, aceasta poate semnala un set neclar de teste.
  - (4) **Cazuri de testare numerotate** concepute pentru a valida îndeplinirea cerințelor sistemului.
- Câte cazuri de testare sunt necesare?
  - Aceasta depinde în întregime de manager, dar ideea este să se procedeze astfel încât un caz de testare să acopere o anumită zonă funcțională.
    - *De exemplu, „pornire”, „procesare de eroare a mesajului de intrare”, „detecție ţintă unică”, „urmărire ţintă”, „calcularea taxelor”, „conversie cu matrice unică”.*
- Numele care se aleg pentru **funcțiile de testare** sunt irelevante.
  - Trebuie alese porțiuni de funcții **suficient de mari** pentru a fi semnificative pentru un utilizator.
  - În același timp, porțiunile de funcții alese trebuie să fie **suficient de mici** încât rezultatele testelor să poată fi ușor asimilate și înțelese.
- Fiecare **caz de testare** constă dintr-un **script, date și liste de verificare** (checklist).
  - (1) Un **script** (scenariu) este un set de instrucțiuni menite să conducă personalul de testare pas cu pas, aproape mecanic, prin teste.
    - Scriptul enumera toate acțiunile necesar a fi executate de către operatorii umani la fiecare echipament implicat în test.

- Le precizează testatorilor:
  - Ce să facă.
  - Când să facă.
- De asemenea, descrie;
  - Ce să verifice.
  - Ce să noteze pentru analiză ulterioară.
- Trebuie făcută ipoteza că un testator înțelege fundamentele și obiectivele fiecărui test, astfel încât în scenariu trebuie incluse foarte puține informații explicative.
- Un exemplu de pagină dintr-un script este prezentată în Figura 7.1.1.b.
- De reținut faptul că porțiunea din dreapta paginii este lăsată goală pentru a înregistra notițe în timpul testului.

---

## SCRIPT TESTARE SISTEM (MODEL)

<b>PROIECT ALFA TEST SCRIPT</b>	
Script Nr.....	
Caz de test Nr: xx.yy	Pag. 7 din 10
Referință la Cazul de test: .....	
PROCEDURI	NOTE
<b>Dialog Editor de frecvență</b>  1. În <i>Data Edit Control</i> , introduceți data în format <i>mddyy</i> . 2. Faceți clic pe butonul OK. 3. În caseta combinată <i>TimeStamp Type</i> , selectați <i>Absolut</i> . 4. În selectorul <i>AbsoluteTimeStamp</i> , setați marca temporală 09:10:23 5. În controlul <i>Listă stații</i> , selectați 2 Prog. Info 6. ....	

**Fig. 7.1.1.b.** Model de script test sistem

- (2) **Partea de date** a unui caz de testare include **date de intrare simulate**, **date de intrare în direct (live)** și **date de ieșire prezise**.
  - (2.1) **Datele de intrare simulate** sunt date pregătite în prealabil în scopul exersării sistemului în timpul unui test dat.
    - De exemplu, dacă sistemul este un sistem de procesare a salariilor, aceste intrări pot include ore lucrate ale angajaților, informații fiscale, deduceri de obligații și altele asemenea.
  - (2.2) **Intrările live** sunt cele care nu pot fi pregătite în mod convenabil din timp, cum ar fi spre exemplu, datele de telemetrie de la un satelit.
    - Acest tip de intrare oferă avantaje care sunt greu de obținut din datele de intrare pregătite, sau din cele simulate. Ele includ un anumit grad de aleatoriu și cu mare probabilitate, o anumită cantitate de „murdărie” (garbage) pe care sistemul va trebui să o gestioneze corect.
  - (2.3) **Rezultatele estimate** sunt previziuni scrise ale datelor exacte care ar trebui să rezulte dintr-un caz de testare dat, acolo unde o astfel de prognoză este posibilă.
    - Dacă **rezultatele sunt determinate în avans**, este necesar să se **compare** doar **rezultatele reale** ale testului cu **cele prognozate** pentru a determina eficacitatea testului.
    - **Predictia în avans a rezultatelor** oferă alte **avantaje** decât simplificarea analizei posttest.
      - Poate **economisi timp calendaristic** deoarece predictia se poate face în paralel cu alte activități, înainte ca testele propriu-zise să fie executate.
      - De asemenea, **poate fi mai precisă** decât analiza posttest la fața locului, deoarece aceasta din urmă dă adesea naștere la decizii pripite.
- (3) Un **set de liste de verificare** pregătite în prealabil pentru a ajuta în timpul analizei posttest.

- **Listele de verificare** sunt ajutoare puternice de analiză atunci când sunt construite atent.
- Un exemplu de listă de verificare parțială este prezentat în Figura 7.1.1.c.

### **LISTĂ DE VERIFICARE TEST SISTEM**

#### **PROJECT ALFA TEST SCRIPT**

Caz de test Nr: xx.yy      Listă de verificare Nr. M14-2      Pag. 1 din 3

Referință caz de testare:.....

Nr.	Articol	Da	Nu
1-0	<b>FEREASTRA FIŞIER LOG</b>		
1-1	Au fost toate informațiile ușor de citit?		
1-2	Toate informațiile afișate au fost clare și lipsite de ambiguitate?		
1-3	Toate informațiile afișate au fost complete?		
1-4	Au fost toate datele afișate bine aranjate?		
1-5	.....		
1-6	.....		
2-0	<b>FEREASTRA DE INFORMAȚII</b>		
2-1	Pozitia ferestrei este semnificativă?		
2-2	Informațiile sunt aranjate într-un mod confortabil?		
2-3	.....		
2-4	....		

**Fig. 7.1.1.c.** Listă de verificare parțială pentru un test sistem

- Fiecare caz de testare trebuie să fie **autonom** și **independent** de toate celealte.
- În cadrul fiecărui test ar trebui să existe **puncte de repornire încorporate**, locuri convenabile pentru reluare în cazul unor întreruperi neașteptate în timpul testului.
- Fiecare test trebuie planificat astfel încât în analiza posttest să poată fi arătat clar:
  - (1) Care au fost **intrările**.
  - (2) Care au fost **rezultatele prezise**.
  - (3) Care au fost **rezultatele reale** ale testului.

## 1.2 Testerii

- **Testerii** sunt cei care **planifică, execută și analizează teste**le aparțin **Grupului de testare** (vezi Organizarea proiectului).
  - (1) Aceste persoane trebuie să fie **competente** din punct de vedere tehnic și inclinate **analitic**.
  - (2) Ele trebuie să **înțeleagă la nivel de detaliu clientul** și specificația problemei.
  - (3) O parte din grup ar fi trebuit să **participe** la **analiza problemei originale**.
  - (4) Cel puțin **un membru al grupului** ar trebui să fie **utilizator**.
- Trebuie creată o atmosferă de competiție prietenoasă (dar nu prea prietenoasă) între **testeri** și **programatori**.
- **Testerii de sistem** sunt pregătiți să **descopere probleme**.
  - Dacă nu găsesc niciuna, în timpul activității lor:
    - Fie testerii nu se pricep.
    - Fie programatorii sunt eroi.
    - Fie sistemul a fost extrem de simplu de construit.
- În mod normal, ar trebui să ne așteptăm la **probleme**.
  - Unele dintre ele se pot datora **erorilor**, dar multe vor fi **probleme de interpretare a Specificației problemei** sau probleme legate de **ușurința utilizării sistemului**.
  - Unele probleme din ultima categorie pot fi rezultatul unor greșeli din documentația utilizator, mai degrabă decât erori de programare.
- Orice **problemă** este un joc cinstit (fair game), în principiu o **problemă de interpretare**.
- **Testerii de sistem** trebuie să „gândească clientul”, adică să abordeze sistemul strict **din punctul de vedere al clientului** și să evalueze constant sistemul din acest punct de vedere.

- Nu trebuie alese ca **testeri** de sistem persoane care sunt blânde, ușor de intimidat sau slabe din punct de vedere tehnic.
- Acești oameni ar trebui să fie **demonici** în încercările lor de a face sistemul să eșueze.
- Eșecul sistemului înseamnă succesul lor.
- Trebuie să fie încurajați să pătrundă în toate colțurile întunecate ale sistemului.
- Nu trebuie niciodată sugerat faptul că ar putea merge mai ușor în anumite zone.
- Trebuie reținut faptul că, ceea ce nu găsește **testerul**, o va face în cele din urmă **clientul**.

### 1.3 Condițiile de testare

- Nicio fază de testare nu ar trebui să înceapă până când **criteriile de intrare în test** nu au fost **îndeplinite**.
- **Criteriile specifice de intrare în test** pot varia, în funcție de produsul testat, dar, în general, acestea includ următoarele:
  - (1) **Produsul** care urmează să fie testat (program sau document) trebuie să fie considerat **terminat** de către dezvoltatorii săi;
    - Orice **excepție** trebuie să aibă **aprobare explicită de la conducere**.
  - (2) **Specificația de testare** scrisă pentru acel produs trebuie să fie **terminată, aprobată și gata de execuție**.
  - (3) Toate **datele de testare suport** trebuie să fie **gata**.
  - (4) Atât **dezvoltatorii**, cât și **oamenii de testare** trebuie să fie **prezenți** sau la **apel**.
  - (5) **Timpul calculator**, dacă este necesar, trebuie să fie **planificat**.
  - (6) Orice **resurse speciale** (de exemplu, echipamente la distanță, facilități temporare de locuit într-un loc îndepărtat, transport) trebuie să fie **gata**.
- Se pot învăța câteva lecții despre testarea sistemului din dezastrele trecute.
  - (1) În primul rând, niciun fel de noroc, perspicacitate politică sau viață sfântă **nu poate justifica să se sară peste faza de testarea sistemului** și să se scape de ea.
    - Într-un sistem de orice complexitate, chiar cei mai buni programatori vor face **greșeli**.
    - Este evident că este de dorit să fie găsite **înainte** ca produsul să ajungă la client.

- (2) În al doilea rând, **nu se poate face testarea sistemului în paralel cu programarea**. Nu există încă nimic care merită testat.
  - Ceea ce poate face, totuși, este să se accepte cu foarte mare grijă să înceapă testarea subsistemelor finalize chiar dacă sistemul nu este finalizat.
  - Acest lucru se aplică numai atunci când subsistemele constitutive sunt aproape independente unele de altele, iar interfețele dintre ele sunt curate și simple.
- (3) În al treilea rând, **nu trebuie răspândită geografic testarea sistemului** până când nu se ajunge la punctul în care **sistemul arată curat** și are nevoie doar de testare în funcție de condițiile unice ale site-ului.
  - Aceste condiții pot include baze de date statice modificate, coordonate ale locației geografice, dispozitive variante de intrare și ieșire și încărcări variante la intrare.
  - De fiecare dată când se introduce o nouă copie a sistemului în mediul de testare, au crescut problemele de control.
  - Fiecare modificare dintr-o versiune trebuie să fie reflectată și testată în toate celelalte versiuni.
  - Comunicarea este de obicei un coșmar și în același timp o scurgere pentru tot felul de resurse.
    - Este însă posibil că acest lucru să nu fie atât de imperativ acum, datorită noilor facilități de comunicare bazate pe internet.

#### **1.4 Efectuarea testelor**

- **Testarea sistemului** începe când programatorii **livrează** grupului de testare **produsul lor finit**:
  - (1) Un **sistem de programe** (sau un subsistem dacă este cazul).
  - (2) Un **draft curat** al documentației.
- **Managerul** grupului de testare **acceptă, semnează** și încide în seif sistemul pe care îl primește.
- **Conducători de testare** sunt alocați pentru diversele grupuri specifice de cazuri de testare.
- **Un conductor de testare** este responsabil pentru **pregătirea, execuția și analiza** unui **set de cazuri de testare**.
  - (1) Conducătorul de testare **programează** oamenii, timpul de calcul, facilitățile **fizice** și orice alte **resurse necesare** pentru aceste **cazuri de testare**.
  - (2) Înainte de începerea unui test, **conducătorul** se **asigură că**:
    - Toți **participantii** și **observatorii** sunt disponibili.

- Au toate materialele necesare, cum ar fi scenarii, liste de verificare și rezultate estimate.
- (3) Conductorul inițiază testul.
- (4) Cei care participă își urmează cu exactitate scenariile, notând orice discrepanțe sau condiții neobișnuite.
- (5) Conductorul ia toate deciziile referitoare la intreruperi sau reporniri, cu excepția cazului în care scenariul prevede deja acestea.
- (6) Observatorii, cărora li se pot oferi copii ale scenariilor, trebuie încurajați să noteze idei și întrebări pe măsură ce testul se derulează.
  - Observatorii pot fi programatori sau alți membri ai proiectului.
  - Ei pot fi, de asemenea, reprezentanți ai clienților.
- (7) Dacă s-a decis să se invite clientul pentru a oferi o previzualizare a sistemului, trebuie precizat clar, că acesta este un test al unui sistem incomplet, nu o demonstrație a unui produs finit.
  - Clientul ar trebui să știe că sunt de așteptat probleme.
  - Dacă își face ideea că testul de sistem nu este mai mult decât o repetiție generală pentru testul de acceptanță, va exista un anumit grad de neliniște, sau chiar de suspiciune, de fiecare dată când apare o problemă.
- (8) Când un test sau o serie de teste a fost finalizată, conductorul cheamă participanții la test pentru analiza rezultatelor.
  - În unele cazuri, această analiză poate fi finalizată și poate fi făcută imediat o evaluare a succesului sau eșecului.
  - În alte cazuri, rezultatele vor necesita un studiu, iar conductorul va trebui să convoace o altă întâlnire de analiză.
  - În orice caz, conductorul este responsabil pentru declararea unui test succes sau eșec (ceea ce poate însemna un succes parțial) și pentru raportarea rezultatului către conducere.
- Trebuie folosit un formular simplu pentru a documenta rezultatele.
  - Raportul de testare trebuie să includă (fig.7.1.4.a.):
    - Numărul de identificare al cazului de testare.
    - Data testului.
    - Rezultatul.

**RAPORT DE TESTARE**  
**(MODEL)**

**DEPARTMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**NUMĂR CAZ DE TESTARE:**

**RREFERINȚA LA CAZUL DE TESTARE ASOCIAȚ:**

**APROBĂRI:**

**DATA TESTULUI:**

**DATA RAPORTULUI:**

**NUME CONDUCTOR DE TEST:**

**PROBLEMA DEPISTATĂ:**

Dacă nu există probleme, se menționează acest lucru.

În caz contrar, fiecare problemă identificabilă este listată pe o foaie separată ca formular de descriere a erorilor atașat acestui raport. Pentru fiecare problemă trebuie furnizate următoarele informații:

- (1) Un număr unic de identificare a problemei folosind numărul cazului de testare.
- (2) Identificarea modulelor de program în care a apărut problema, dacă este posibil.
- (3) O descriere a problemei, cu toate datele disponibile.
- (4) Recomandări, dacă există, pentru posibile soluții pentru problemă.

**REFERINȚĂ DE DESCRIERE A ERORII:**

Nr.crt	Număr eroare	Referință descriere eroare	Obs

**Fig.7.1.4.a. Raport de testare**

- Dacă rezultatul a fost calificat drept **succes**, **nu** este necesară nicio elaborare.

- Dacă au existat **probleme**, acestea trebuie detaliate, iar **conductorul de testare** trebuie să se asigure că au fost desemnați programatorii care să le urmărească și să le corecteze.
  - **Conductorul de testare** trebuie, de asemenea, să decidă ce **testare ulterioară** trebuie făcută după ce a fost făcută o **remediere**.
- După o **remediere**, există mai multe **opțiuni de testare** dintre care se poate alege:
  - (1) **Remedierea** poate implica doar **o modificare minoră a documentației**, astfel încât să nu fie nevoie de re-testare.
  - (2) **Modificarea** poate necesita **o revizuire majoră** a unuia sau mai multor module de program, deoarece specificația problemei a fost interpretată greșit.
    - În acest caz, în urma consultării cu clientul se poate demara negocierea pentru a face modificările agreeate;
  - (3) Poate fi necesară **o analiză, proiectare sau modificare a codificării** dacă remedierea afectează părți ale sistemului deja testate.
    - În principiu trebuie presupus că toate modificările au acest efect.
    - Conductorul de testare, asistat de programatori, trebuie să decidă cât de multă **testare de regresie** este necesară.
      - **Testarea de regresie** este re-testarea porțiunilor terminate anterior ale sistemului pentru a se asigura că modificarea de remediere nu afectează și alte părți din sistem, cauzând unele probleme aparent fără legătură.
    - Deciziile de test de regresie sunt greu de luat. Unele dintre cele mai simple modificări ar putea avea repercurșiuni chiar în intimitatea sistemului.
  - (4) **Documentația descriptivă** (Specificații de codificare) poate fi greșită ca urmare a remedierii.
    - Aceste descrieri trebuie să corespundă exact cu varianta corectată a programelor.
    - Trebuie insistat ca fiecare modul trimis pentru testarea sistemului să fie asamblat sau compilat curat - fără patch-uri.
- Când un sistem sau un subsistem a trecut curat prin **testul final al sistemului**, programele și documentația lor sunt blocate în **bibliotecă** pentru a aștepta demonstrația de acceptare.

## **2. Instruirea beneficiarului**

- Un **obiectiv secundar** al fazei de testare a sistemului îl reprezintă **pregătirea** și, în multe cazuri, **începerea instruirii beneficiarului (clientului)**.
- Există cel puțin două domenii care necesită instruire pentru clienți:

- (1) Utilizarea sistemului.
- (2) Întreținerea sistemului.

## 2.1 Utilizarea sistemului

- De obicei, referitor la utilizarea sistemului, trebuie soluționată una dintre următoarele două situații:
  - Fie **noul sistem automatizat înlocuiește o procedură manuală**.
  - Fie **noul sistem se introducece acolo unde înainte nu exista nimic**.
- În ambele cazuri, **personalul clientului** trebuie **să învețe să opereze** ceea ce le este livrat.
- În acest sens trebuie luate în considerare toate **mijloacele obișnuite de instruire**:
  - (1) Săli de clasă formale.
  - (2) Sesiuni, seminarii.
  - (3) Formare la locul de muncă.
  - (4) Instruire asistată de calculator.
- Unele instruiriri vor fi unice pentru proiectul în cauză.
  - Alte **forme** de instruire pot fi disponibile prin programul obișnuit de instruire oferit de companie.
- În orice caz, vor trebui **scrise și livrate manuale de utilizare** pentru a fi utilizate atât în timpul **instruirii**, cât și **în operarea viitoare a sistemului**.
  - Aici apare o oportunitate de a străluci. Multe sisteme de programe bune au fost livrate însotite de **ghiduri de utilizare slabe**.
  - Scrierea proastă a manualelor se reflectă negativ asupra întregului sistem. Adesea, informațiile destinate utilizatorului sunt singura parte tangibilă de către client a produsului realizat. El va judeca produsul după calitatea documentației oferite.
  - Se recomandă ca dezvoltatorul aibă grijă să o facă cât mai bine. Trebuie găsiți acei oameni printre analiști care manevrează cu ușurință cuvintele și pot exprima un punct de vedere clar și concis.
  - Acest job trebuie demarat cât mai devreme, nu cu trei zile înainte de acceptare.

## 2.2 Menținerea sistemului

- Adesea **clientul** își asumă sarcina **de a face el însuși modificări viitoare ale sistemului** și, prin urmare, trebuie să înțeleagă în detaliu produsul care îi este livrat.

- Este posibil să se solicite dezvoltatorului să instruiască un nucleu de oameni tehnici ai clientului care, la rândul lor, îl vor instrui pe alții.
  - Indiferent de metoda de instruire folosită, este nevoie de o documentație precisă și utilizabilă.
- Două seturi de documente descriptive, ar trebui să descrie complet și corect sistemul de programe:
  - (1) **Specificația de proiectare**.
  - (2) **Specificațiile de codificare**.
  - (3) În plus, pot fi redactate **manuale speciale de depanare** care vor permite altora să înțeleagă particularitățile speciale ale sistemului sau să le arate modalitățile cele mai directe pentru sondarea zonelor de cod deosebit de dificile.
- Uneori, **instruirea clienților** este o sarcină uriașă din cauza:
  - (1) Complexitatății sistemului.
  - (2) Numărului foarte mare de persoane care urmează să fie instruite.
  - (3) Costurilor logistice implicate în exercițiile de antrenament (poate trebuie lansate rachete sau dislocate unități ale forțelor armate).
- În aceste situații, poate fi întelept să se ofere **instruirea în baza unui contract separat**, unul în baza căruia dezvoltatorul este plătit proporțional cu efortul depus.
  - În esență, instruirea continuă până când clientul spune „destul”.
- Dacă clientul este o agenție guvernamentală, există de obicei tipuri adecvate de contracte disponibile, inclusiv contracte de tip „ore de muncă” sau contracte de tip „timp și materiale”.

## **Exercițiul #7**

1. *Ce sunt testele de sistem? Argumentați necesitatea lor.*
2. *Care este conținutul specificației de testare a sistemului?*
3. *Ce este un caz de testare? Descrieți conținutul componentelor Cazului de testare: scriptul de testare, date și lista de verificare.*
4. *Care sunt principalele caracteristici ale testerilor?*
5. *Care sunt criteriile de intrare în test? Descrieți importanța lor.*
6. *Cum se desfășoară un test? Descrieți atribuțiile actorilor implicați. Care este conținutul unui raport de testare?*
7. *Care sunt alte activități care trebuie făcute în timpul fazei de testare a sistemului?*
8. *Ce este instruirea beneficiarului și în ce constă ea? Ce tipuri de instruire a beneficiarului cunoașteți? Descrieți-le pe fiecare în parte.*



## **Capitolul 8. FAZA DE ACCEPTANȚĂ A SISTEMULUI**

### **1 Testarea de acceptanță**

- 1.1 Specificația testării de acceptanță
- 1.2 Criteriile de acceptanță
- 1.3 Executarea testării de acceptanță

### **2 Documentația**

#### **Exercițiul #8**

## Capitolul 8. FAZA DE ACCEPTANȚĂ A SISTEMULUI

- Obiectivul Fazei de Acceptanță este de a demonstra clientului că sistemul ce urmează să fie livrat **satisfacă contractul semnat cu clientul**.
- **Testarea de acceptanță** este cunoscută sub diferite denumiri:
  - Testare demonstrativă.
  - Test de performanță.
  - Verificarea produsului.
  - și așa mai departe.
- Oricare ar fi numele, rezultatul dorit este **acceptarea scrisă a produsului** de către client.

### 1. Testarea de acceptanță

#### 1.1 Specificația testării de acceptanță

- Ca și în testarea sistem, **testarea de acceptanță** este prezentată și formalizată într-un document numit **Specificația testului de acceptanță** (Fig. 8.1.1.a).

#### SPECIFICAȚIA TESTULUI DE ACCEPTANȚĂ (MODEL)

**DEPARTMENT:**

**PROIECT:**

**NUMĂR DOCUMENT:**

**APROBĂRI:**

**DATA REALIZĂRII:**

**REALIZAT DE:**

#### SECȚIUNEA 1: SCOP

Există patru seturi separate de specificații de testare: Integrare, Sistem, Acceptanță și Specificații de testare a site-ului. Modelele pentru toate cele patru sunt identice, cu excepția faptului că trebuie inserat calificativul corespunzător („integrare”, „sistem”, „acceptanță” sau „site”). Conținutul specificațiilor poate varia, desigur, considerabil, deși două dintre ele (acceptanță și site-ul) vor fi adesea identice. Această secțiune, intitulată

*Scop, ar trebui să servească în fiecare caz ca o introducere a documentului, descriind intenția acestuia și modul în care urmează să fie utilizat.*

## **SECȚIUNEA 2: DOCUMENTE APLICABILE**

### **SECȚIUNEA 3: TESTAREA DE (INTEGRARE, SISTEM, ACCEPTANȚĂ, SITE). CONSIDERENȚE GENERALE**

**3.1. Filozofia de testare**

**3.2. Obiective generale**

**3.3. Proceduri generale**

**3.4. Criterii de succes**

### **SECȚIUNEA 4: MATRICEA DE ACOPERIRE**

*O diagramă care listează de-a lungul axei verticale zonele care trebuie testate și de-a lungul axei orizontale numărul (numerele) cazului de testare care acoperă fiecare zonă. Când este complet, acest grafic reprezintă o referință încrucișată între toate zonele care trebuie testate și toate cazurile de testare care acoperă acele zone.*

---

**Fig.8.1.1.a.** Model de specificație de testare

- **Cazurile de testare** sunt construite în același mod ca și cele utilizate în **testarea sistem**.
  - Într-adevăr, **multe sau toate cazurile** pot fi **aceleași** cu cele utilizate în **testarea sistem**.
- **Specificația testului de acceptanță** trebuie scrisă de **comun acord cu clientul**.
  - **Dezvoltatorul redactează, clientul aprobă**.
    - Există multe excepții de la aceasta. Adesea clientul pregătește documentele de acceptare.
  - Nu are nici un rost să-i fie prezentată clientului prima versiune a documentului de acceptanță la sfârșitul proiectului.
    - Clientul s-ar putea pur și simplu să nu fie de acord cu asta și vor apărea probleme.

- Specificația testării de acceptanță trebuie produsă în etape.
  - (1) Mai întâi, în timpul fazei de definire, se redactează secțiunea numită „Criterii de acceptanță” (sau „Criterii de succes”).
    - Acesta este unul dintre cele mai critice documente pentru orice proiect.
    - Se precizează de facto condițiile specifice în care clientul va accepta în mod oficial produsul.
  - (2) Apoi se redactează sectiunile rămase din specificația testării de acceptanță și se construiesc toate cazurile de testare necesare.
  - (3) Specificația finală și cazurile de testare ar trebui să fie finalizate în același timp cu specificația de testare sistem, astfel încât cele două să poată partaja cazuri de testare.

## 1.2 Criteriile de acceptanță

- Criteriile de acceptanță sunt condițiile pe care sistemul realizat trebuie să le îndeplinească înainte ca beneficiarul să accepte oficial sistemul și să fie de acord că acesta îndeplinește contractul.
- Scrierea criteriilor de acceptanță este una dintre cele mai dificile treburi cu care se confruntă dezvoltatorul.
  - Clientul va fi dur în această privință și ar trebui să fie.
  - Cel care semnează un acord de acceptanță știe că jobul lui este în joc.
  - Orice acorduri prietenoase pe care dezvoltatorul și clientul le-au avut în trecut sunt nule și neavenite.
- Criteriile de acceptanță ar trebui stabilite chiar înainte de semnarea contractului, iar uneori acest lucru este posibil.
  - De obicei, totuși, nu mai mult decât niște criterii generale sunt convenite devreme.
  - Contractul ar trebui să preciseze cel puțin:
    - (1) Planificarea redactării criteriilor finale.
    - (2) Restricțiile de timp impuse clientului pentru revizuirea propunerilor dezvoltatorului.
- Clientul trebuie să fie de acord să accepte sau să respingă specificațiile de acceptanță propuse de dezvoltator în perioade de timp definite.
  - Fără această clauză, se poate utiliza întreg timpul calendaristic și epuiza bugetul fără ca încă să existe un acord cu privire la termenii de acceptanță.
- În stabilirea timpilor pentru examinarea și revizuirea documentelor, trebuie avut în vedere faptul că probabil, va fi nevoie de mai mult decât o singură iteratie.

- Este foarte rezonabil să se presupună că clientul va dori să se facă **modificări** și în consecință **trebuie prevăzut timp** pentru **o a doua versiune și revizuirea acesteia**.
- **Criteriile de acceptare** trebuie să se bazeze pe criterii **cantitative, măsurabile**.
  - Scopul dezvoltatorului ar trebui să fie **eliminarea subiectivitatii** din **evaluarea rezultatelor testelor** (acest lucru se aplică oricărui test).
    - Dacă se poate include în criteriile de acceptanță o declarație conform căreia „raportul ABC va fi tipărit în termen de trei minute de la solicitarea butonului”, este mult mai bine decât a spune „raportul ABC va fi tipărit într-un timp rezonabil după solicitarea butonului”. Dezvoltatorul și clientul pot cronometra simplu trei minute cu un cronometru, dar este posibil să nu cadă niciodată de acord cu privire la ceea ce constituie un timp „rezonabil”.
- **Criteriile de acceptanță** acoperă nu numai **performanța** cerută de sistem, ci și **livrarea sistemului**.
  - (1) Când și unde trebuie să fie livrate copiile sistemului?
  - (2) În câte exemplare?
  - (3) În ce formă?
  - (4) Cum ar trebui să fie ambalate materialele?
  - (5) Unul dintre livrabile ar trebui să fie setul de cazuri de testare utilizate în timpul demonstrației de acceptanță.
- Poate fi util să luăm în considerare câteva **exemple de criterii de acceptanță proaste** pentru a înțelege ce ar constitui criterii bune.
- Următoarele **exemple** sunt preluate din **documentele reale de acceptare**:
  - *Antreprenorul va avea prioritate în utilizarea computerelor clientului pentru verificarea programului.*
    - **Ce prioritate? Cea mai înaltă? Următoarea?** Această afirmație nu oferă nimic.
  - *Se vor include un număr adecvat de mesaje pentru a testa sistemul în mod exhaustiv.*
    - **Ce înseamnă adecvat? Zece mesaje sau zece mii? Ce înseamnă „exhaustiv”?**
  - *Antreprenorul va pregăti, cu asistența clientului, mesaje de testare pentru a-și exersa toate opțiunile enumerate în scopul lucrării.*
    - **Cât de multă asistență pentru clienți? Cine face analiza? Cine pregătește de fapt mesajele? Acesta poate fi un job de o săptămână sau de un an om.**
  - *Afișarea vizuală a datelor de urmărire a ţintei va avea loc în timp util, în conformitate cu condițiile de amenințare existente.*

- Ce înseamnă asta? Orice interpretare este la fel de bună oricare alta.
- Antreprenorul va face modificările pe care le solicită clientul în termen de treizeci de zile de la demonstrarea oficială de acceptare.
  - Această afirmație într-un context adecvat ar putea fi utilă, dar în situația de față nu este cazul. Ea reprezintă de fapt un angajament pe timp nedeterminat din partea contractantului.
- Lucru ciudat la toate aceste este acela că formulările de mai sus au fost luate din acorduri de acceptare scrise de **antreprenori**, **nu de clienți!**

### 1.3 Execuția testelor de acceptanță

- **Testarea de acceptanță** este efectuată **în același mod** ca și **testarea sistem**, dar cu următoarele **diferențe**:
  - (1) În primul rând, **clientul** trebuie să joace **un rol activ** **în testarea de acceptanță**.
    - Clientul trebuie **să furnizeze oameni** care să efectueze unele sau toate operațiunile manuale.
    - Clientul va fi cu siguranță foarte important **în analiza posttest** și va fi necesar, desigur, **să aprobe** un test înainte de a putea fi considerat finalizat sau de succes.
  - (2) În al doilea rând, **clientul** ar trebui să insiste **să introducă în sistem și date inedite** care **nu** au fost văzute sau folosite niciodată până acum.
  - (3) În al treilea rând, **acceptarea poate fi condiționată**.
    - De exemplu, pentru a obține acceptarea deplină pot fi necesare teste ulterioare pe situri geografice dispersate (vezi discuția despre proiectele mari în Capitolul 10).

## 2. Documentația

- Pe măsură ce **testarea de acceptanță** continuă, pot fi detectate **defecte minore** atât în **documentația utilizatorului**, cât și în **documentația descriptivă** a sistemului.
  - **Corecturile** trebuie făcute, desigur, înainte de a livra documentele finale către client.
- Din acest motiv, **documentația finală** trebuie **să fie planificată și bugetată** pentru livrare la sfârșitul **Fazei de Acceptanță** sau chiar mai târziu, astfel încât **corecțiile să poată fi incluse** de o manieră fezabilă.
  - Alternativa este să se livreze documentele mai devreme și apoi să se suporte cheltuiala și inconvenientul **emiterii corecturilor și erratelor**, ceea ce nu dă deloc bine pentru proiect.

### **Exercițiul #8**

1. Care sunt obiectivele fazei de testare de acceptanță?
2. Care este conținutul specificației testării de acceptanță? Care sunt particularitățile elaborării sale?
3. Ce sunt criteriile de acceptanță? Care sunt principalele lor caracteristici?
4. Care sunt particularitățile executării testării de acceptanță?
5. Care sunt celelalte activități care trebuie efectuate în timpul fazei de testare de acceptanță?

## **Capitolul 9. FAZA DE INSTALARE ȘI OPERARE**

**1 Testarea de site**

**2 Conversia**

2.1 Operarea paralelă

2.2 Înlocuirea imediată

2.3 Inaugurarea

**3 Întreținerea (menenanța) și reglarea sistemului**

**4 Evaluarea proiectului**

**Exercițiul #9**

## Capitolul 9. FAZA DE INSTALARE ȘI OPERARE

- Multe proiecte, în special cele mai mici, se încheie cu **Faza de Acceptanță**.
- **Demonstrațiile de acceptanță** pot fi efectuate pe **echipamentul clientului**, iar la finalul cu succes al demonstrațiilor **sistemul este deja instalat și operațional**.
- În alte cazuri, totuși, **acceptarea** este **condiționată** și trebuie urmată de:
  - (1) **Testare ulterioară** la o altă locație (site).
  - (2) **Conversia** de la **un sistem vechi** la **cel nou**.
  - (3) **Întreținere și reglare continuă**.
- Deoarece este posibil ca **dezvoltatorul să aibă responsabilități contractuale** pentru unele sau pentru toate aceste sarcini, vor fi acoperite pe scurt în acest capitol.

### 1 Testarea de site

- **Testarea de site** trebuie făcută în **mediul final** în care va funcționa sistemul, adică în **site-ul operațional**.
  - Această **testare** poate echivala cu **simpla repetare a testelor anterioare** (teste de acceptanță) în noul mediu.
  - La cealaltă extremă, testarea la fața locului poate fi o muncă uriașă care necesită multă pregătire și cheltuieli.
    - În această ultimă categorie se află spre exemplu, proiectele monstru de apărare care necesită procesarea de date, arme, subsisteme de ghidare și comunicații, toate testate independent pentru a fi integrate și testate împreună la locul operațional.
- Dacă proiectul necesită **testare de site**, trebuie avut grijă ca această activitate să fie planificată cu mare atenție.
  - În acest sens trebuie scrisă o **specificație de testare de site** similară cu **specificația de testare de integrare și specificația de testare de acceptanță** descrise mai devreme.
- Există mai multe zone care **diferențiază** testarea despre care s-a discutat până acum și **testarea la fața locului**.
  - (1) În primul rând, **echipamentul de calcul** de la locul de exploatare poate să nu fie identic cu cel utilizat în fazele precedente.
  - (2) Pot exista diferențe între **dispozitivele de intrare- ieșire**.
    - De exemplu, pot exista versiuni diferite de sisteme de operare, dimensiuni diferite ale nucleului și chiar modele diferite ale unei familii

de computere care ar putea afecta viteza de execuție și repertoriul de instrucțiuni disponibil.

- (3) Într-un loc fizic de exploatare (în site) toate acestea **reale**; sistemul construit va funcționa probabil **pentru prima dată fără simulare**.
  - Până la momentul testării de site, este posibil ca sistemul să se fi bazat pe **simulare** ca **sursă de intrări în sistem** sau ca **receptor de ieșiri**, sau ambele.
  - De regulă, se întâmplă lucruri ciudate atunci când **simulatoarele sunt înlocuite cu echivalentele lor reale**, indiferent cât de strălucit au fost planificate și executate simulatoarele.
- (4) **Baza de date** pe care sistemul o folosește la locul operațional poate fi diferită de cea utilizată anterior.
  - De exemplu, unele date pot fi identificatori reali de site sau coordonate de locație geografică. Modificările bazei de date pot crea probleme deosebite.
  - Cele mai mici ajustări trebuie testate temeinic.
- Dacă **sistemul** realizat urmează să fie instalat și testat **în mai multe locații simultan**, trebuie să existe certitudinea că **sistemul** este **într-adevăr gata pentru a fi trimis**.
  - **Problemele de comunicare** între mai multe echipe care operează pe teren în diferite locații pot fi destul de dificile fără un **sistem performant de tratare a erorilor**.

## 2 Conversia

- În general, **sistemul dezvoltat va înlocui altceva existent**, poate o **operare manuală**.
- **Clientul** va trebui să convertească operarea de la **sistemul vechi** în **cel nou** și este posibil ca dezvoltatorul să fie profund implicat în conversie.
- Există două tipuri de conversie (și multe variante ale fiecăreia):
  - (1) **Operarea în paralel**.
  - (2) **Înlocuirea imediată**.

### 2.1 Operarea în paralel

- Funcționarea în paralel înseamnă că **vechiul sistem nu este înălăturat** până când nu există siguranță că **cel nou funcționează**.
  - Iată ce are de spus **Robert Townsend** despre acest subiect: „*Indiferent ce spun experții, niciodată, niciodată nu automatizați o funcție manuală fără o perioadă suficient de lungă de funcționare dublă. Când aveți îndoieri, întrerupeți automatizarea. Și nu opriți funcționarea manuală până când nonexpertii din organizație cred că automatizarea funcționează. Nu am*

*cunoscut niciodată o companie grav rănită din cauza automatizării prea încete dar sunt câteva cazuri clasice de companii falimentate prin informatizare prematură."*

- În condiții de **funcționare paralelă**, **clientul** are **flexibilitate maximă**.
- Ieșirile din noul sistem pot fi utilizate imediat, cu posibilitatea de a reveni oricând la vechiul sistem.
- **Noul sistem** poate fi de asemenea **introdus treptat**.
  - În acest caz, clientul folosește părți ale ambelor sisteme cu eventuala înlocuire totală a celui vechi.

## 2.2 Înlocuirea imediată

- În acest caz, **vechiul sistem** este **oprit** și este **instalat noul sistem**.
- Desigur, aceasta este o **optiune riscantă**, dar de cele mai multe ori pot exista **motive convingătoare** pentru a fi **aleasă**.
  - (1) În primul rând, **operarea în paralel** poate fi practic **imposibilă** din cauza **spațiului limitat** sau **a altor resurse**.
  - (2) În al doilea rând, **funcția îndeplinită de sistem** poate să **nu fie** **atât de critică** încât o **defecțiune** să **însemne un dezastru**.
  - (3) În al treilea rând, **cheltuielile operațiunilor paralele** pot fi **prohibitive**.
  - (4) În al patrulea rând, **noul sistem** poate fi radical diferit față de vechile proceduri.

## 2.3 Inaugurarea (Cut-over)

- **Inaugurarea** este momentul **tăierii panglicii**, punctul în care **vechiul sistem este înălăturat** pentru totdeauna și **este adoptat noul sistem**.
- Există câteva elemente importante referitoare la inaugurare pe care dezvoltatorul și clientul trebuie să le ia în considerare din timp.
  - (1) **Criteriile de inaugurare**.
    - Care sunt **condițiile** în care se va realiza inaugurarea?
    - Când este **momentul magic**?
    - Se va face **la risc** sau **numai după o inspecție amănunțită** a **ieșirilor sistemului**?
  - (2) **Responsabilitatea inaugurării**.
    - **Cine ia decizia**? În mod normal, acesta este clientul, dar nu întotdeauna. Beneficiarul și dezvoltatorul trebuie să decidă acest lucru din timp.

- (3) **Responsabilitatea operațională.**
  - Cine operează de fapt sistemul la momentul inaugurării? Dezvoltatorul, sau clientul, sau ambii?
- (4) **Opțiunile de recuperare.**
  - Aproape toate sistemele ar trebui să aibă **proceduri explicite** și un **manual separat de mesaje de eroare** pentru **recuperare** în caz de defecțiune a echipamentului, eroare umană și aşa mai departe.
  - **Recuperarea** ar fi trebuit să fie abordată, desigur, încă din **faza de proiectare**.
  - Ceea ce este **important acum** este ca **cei care operează sistemul** să **înțeleagă clar** cum să folosească **diferitele funcții de recuperare**.

### **3 Întreținerea (mentenanța) și reglarea sistemului**

- Cele mai multe contracte solicită **contractorului** să ofere asistență pentru întreținere și reglare pentru o **perioadă specificată** după livrarea sistemului.
  - (1) **Întreținerea** înseamnă **remedierea problemelor** care apar după inaugurare, inclusiv erori de sistem sau erori de documentare descoperite în timpul fazei de acceptanță.
  - (2) **Reglarea** (acordarea) se referă la **rafinarea parametrilor sistemului** în vederea unei funcționări optime.
    - Se poate face desigur numai după ce sistemul a fost observat în funcțiune o perioadă de timp.
- Toate aceste lucrări se desfășoară cel mai bine în cadrul unui **contract separat de tip „orele de muncă”**, în baza căruia dezvoltatorul furnizează un număr corespunzător de persoane pentru a repara orice dorește clientul să fie reparat iar clientul plătește pentru cheltuielile efectuate.
- Operarea în baza altor tipuri de contracte, cum ar fi **contractele cu „preț fix”, poate fi păgubitoare** pentru **dezvoltator**, deoarece **clientul** poate dori un flux nesfârșit de „reparații” care sunt în realitate cerințe noi sau îmbunătățiri.
- Oricum se decide ca să se resolve problemele apărute, în orice situație **trebuie urmate procedurile de control al modificărilor**.
  - Aceasta este o funcție cel puțin la fel de importantă acum precum a fost în timpul fazei de programare.
  - **Sistemul** este practic în funcțiune, astfel încât orice schimbare insuficient verificată ar putea fi **catastrofală**.

#### 4. Evaluarea proiectului

- În sfârșit, proiectul a ajuns aproape la final. Mai este o treabă de făcut.
- Trebuie **alocat timp** pentru **a analiza cum a decurs proiectul** și pentru **a scrie un raport de evaluare**.
- Acest **raport** destinat conducerii companiei, poate fi utilizat pentru abordarea **viitoarele joburi**.
- Pentru redactarea raportului se utilizează informațiile din **istoricul proiectului** care a fost menținut la zi. În **raportul de evaluare** ar trebui incluse următoarele elemente:
  - (1) **Prezentarea generală a proiectului.**
    - Câteva paragrafe scurte care descriu care a fost problema și cum a fost ea rezolvată.
  - (2) **Succesele majore.**
    - Trebuie evidențiate pe larg.
    - De evidențiat, termenele limită majore îndeplinite, profitabilitatea, satisfacția clientilor, încadrarea în planificări, orice a decurs foarte bine în proiect.
  - (3) **Problemele majore.**
    - Problemele întâlnite pot fi încadrate în mai multe categorii: planificări ratate, depășiri de buget, probleme de performanță tehnică, probleme de personal, probleme în relația cu clientul, calitatea suportului managerial (sau lipsa acestuia).
  - (4) **Forța de muncă estimată vs. forța de muncă efectivă.**
    - Informațiile necesare pot fi obținute din **Istoricul proiectului** și pot fi prezentate sub formă unor **linii directoare** pentru estimarea **următoarelor lucrări**.
  - (5) **Planificări estimate vs. desfășurarea reală.**
    - Să aceste informații pot fi obținute acest lucru din **Istoricul proiectului**. Ele trebuie să sintetizate și tabelate ca și repere pentru **utilizări ulterioare**.
  - (6) **În retrospectivă.**
    - **Ce poate fi repetat și ce s-ar face diferit dacă s-ar repeta același proiect.**

#### Exercițiul #9

1. Care sunt obiectivele fazei de instalare și operare?
2. Ce este testarea de site? Care sunt diferențele dintre testarea de site și alte tipuri de testări?

3. Ce este conversia? Ce tipuri de conversii cunoașteți. Descrieți-le.
4. Ce este inaugurarea? Ce presupune acest concept?
5. Ce este întreținerea? Care este diferența dintre întreținere și reglare?
6. Care este conținutul evaluării finale a proiectului?

## **Capitolul 10. CONSIDERAȚII SPECIALE**

### **1. Proiecte de mari dimensiuni**

- 1.1 Fazele proiectului
- 1.2 Organizare
- 1.3 Controlul clientului
- 1.4 Managementul configurării
- 1.5 Versiuni (release-uri) multiple

### **2. Proiecte de mici dimensiuni**

### **3. Propunerea de proiect**

- 3.1 Ghid pentru redactarea unei propuneri de proiect

### **Exercițiul #10**

## Capitolul 10. CONSIDERAȚII SPECIALE

- Până în prezent, în cadrul cursului s-a discutat în primul rând despre un **proiect de dimensiune medie**, care a fost definit în mod arbitrar ca fiind de aproximativ **douăzeci de persoane**.
- În acest capitol vor fi abordate câteva considerente referitoare la **proiecte mai mari** respectiv **mici**, împreună cu alte câteva idei care ar trebui tratate separat.

### 1. Proiecte de mari dimensiuni

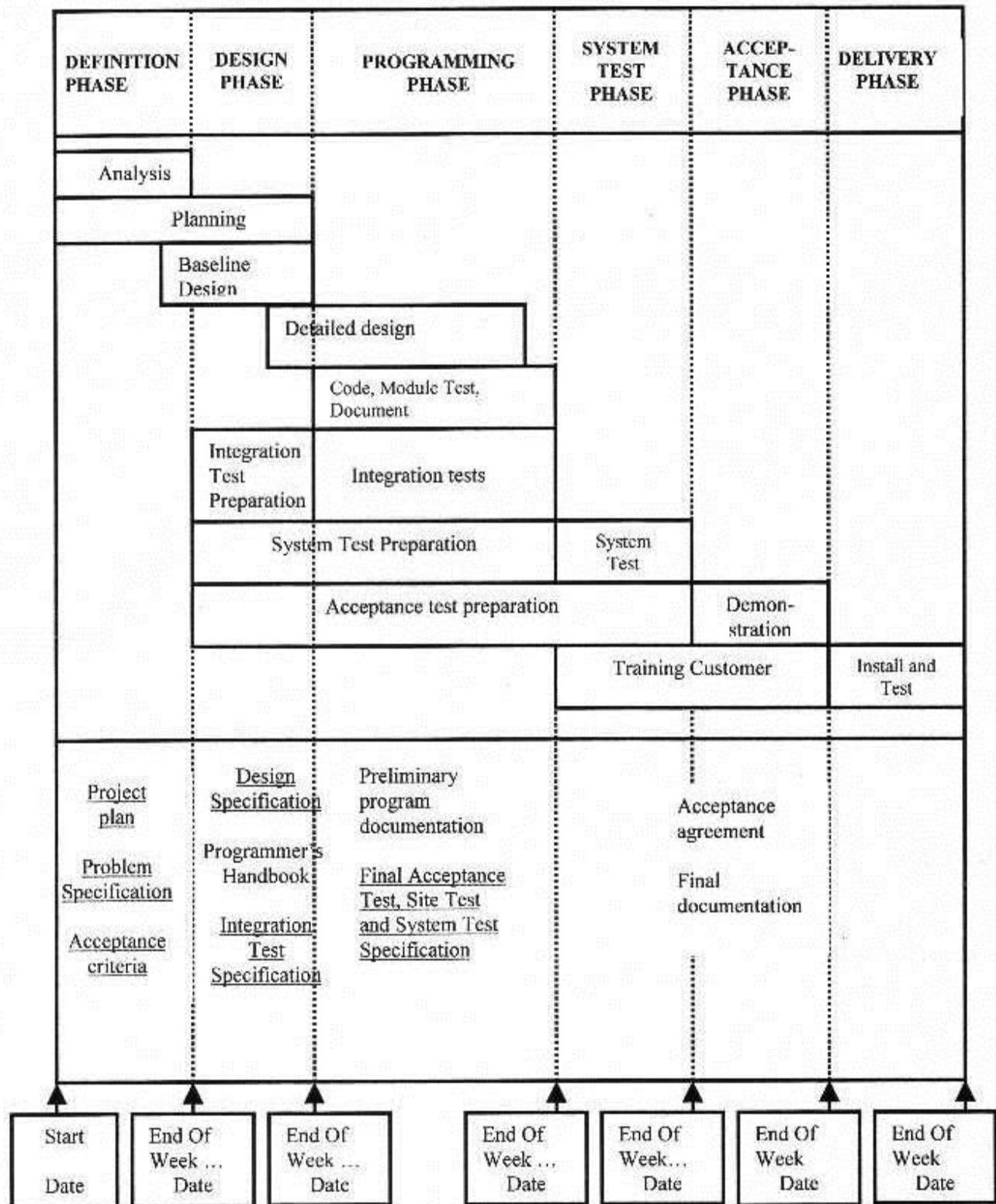
- În primul rând, este nevoie de **un proiect de mari dimensiuni**? Este oare nevoie de mulți oameni într-un loc de muncă de programare?
  - În baza experienței, este cunoscut faptul că se pot realiza multe lucruri valoroase cu o **mână de oameni cu adevărat buni** care sunt **bine gestionati**.
- Nu există absolut nicio îndoială că **un grup de douăzeci de persoane** care lucrează ca **o echipă reală** poate realiza mai mult decât **o masă divizată și greu de gestionat de zeci sau sute de programatori**.
- Există uneori argumente presante și înfricoșătoare care duc la **dublarea sau triplarea** în ultimă oră a estimărilor inițiale ale forței de muncă.
  - (1) Unul dintre ele este aceasta: dacă se propune **un număr semnificativ mai mic de oameni** pentru un job decât propun **concurrentii, oamenii dezvoltatorului**, evident, pur și simplu **nu vor înțelege problema**.
  - (2) Un alt argument: deoarece **clientul** se așteaptă la **cifre mari**, mai bine să le fie propuse.
- De fapt, **nu este o practică bună** ca un dezvoltator să crească artificial forța de muncă la un proiect
  - În timp, acest lucru se va întoarce împotriva sa.
- Desigur, unele **proiecte sunt în mod legitim mari**, iar proiectele mari au unele probleme proprii.
- Să ne uităm la câteva dintre ele.

#### 1.1 Fazele

- În cadrul acestui curs, s-a presupus că ciclul de viață al unui proiect este constituit din șase faze (fig. 10.1.1.a):
  - (1) **Definire**
  - (2) **Proiectare**

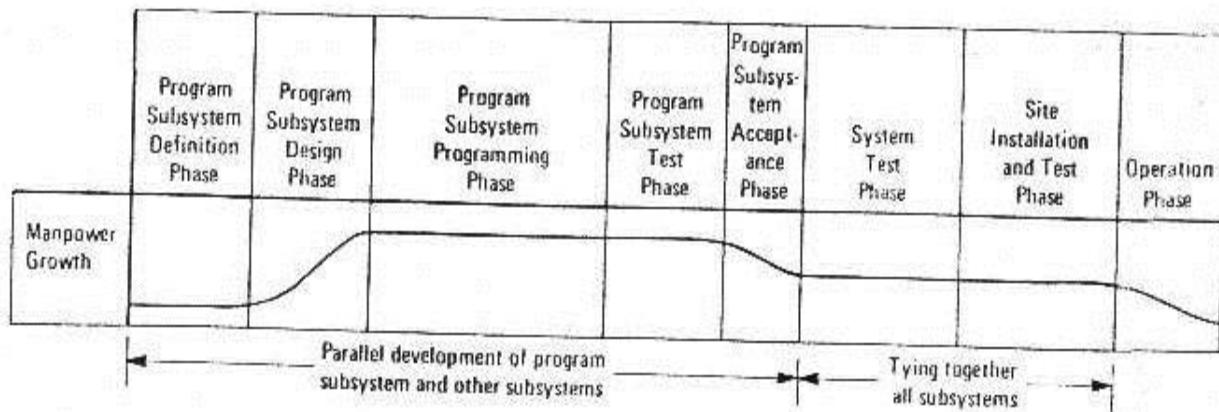
- (3) **Programare**
- (4) **Testare sistem**
- (5) **Acceptanță**
- (6) **Instalare și exploatare**

## The Model of Project Life Cycle



**Fig. 10.1.1.a.** Model de ciclu de viață pentru un proiect software

- Într-un **sistem de mari dimensiuni**, un program realizat de un dezvoltator poate fi doar **unul** dintre **mai multe subsisteme majore** care alcătuiesc **sistemul**. Toate aceste **subsisteme** trebuie să fie **integrate și testate** împreună în proiectul final
- În consecință, una sau mai multe **faze suplimentare** pot fi adăugate la ciclul de dezvoltare din Figura 10.1.1.a.
  - Rezultatul poate fi ceva similar cu Figura 10.1.1.b.



**Fig.10.1.1.b.** Ciclu de dezvoltare extins

- Primele **cinci faze** sunt aceleași cu cele din Figura 10.1.1.a, cu excepția faptului că acum programele sunt în mod clar **subsisteme ale unui sistem mai mare**.
  - În paralel cu munca depusă la **subsistemul de programe**, sunt dezvoltate probabil de alte companii, **alte subsisteme sau echipamente**, cum ar fi sistemele hardware.
  - Aceste activități de **dezvoltare paralelă** nu sunt deloc **independente** una de cealaltă.
- În timpul **fazei de definire**, și chiar mai devreme, sunt necesare cantități masive de **analize și consultări** pentru a scrie **specificațiile** care să definească **subsistemele individuale** necesare.
  - Aceasta este probabil una dintre **cele mai serioase probleme** ale oricărui **proiect mari dimensiuni**.
  - Pe lângă **presiunea** obișnuită exercitată de companiile separate care se luptă pentru o parte cât mai atractivă din totalul proiectului, în realitate există o **confuzie generală** simplă și sinceră.
    - De exemplu, în unele dintre **sistemele militare uriașe**, pe lângă problemele tehnice care uneori sunt uluitoare, nu mai puțin

provocatoare sunt problemele legate de logistică, personal, administrarea contractelor și aşa mai departe. Aceste probleme sunt câteva dintre motivele pentru care **proiectele devin atât de mari**.

- În astfel de situații, sarcina dezvoltatorului va fi ușurată considerabil dacă selectează **cele mai valoroase talente din companie**, pentru a redacta **Specificația problemei**.
  - Probabil că acest **document** va avea un alt nume, dar în orice caz trebuie insistat ca el să descrie:
    - (1) **Problema de ansamblu**.
    - (2) **Problema specifică** care trebuie abordată de subsistemul de programe alocat **dezvoltatorului**.
    - (3) Cum trebuie să interacționeze acest **subsistem** cu **celealte din proiect**.
- Aceast job deosebit de dificil necesită **cei mai buni oameni ai dezvoltatorului**.
  - De obicei, aceștia vor lucra ca parte a unei **echipe** reprezentând **toți contractanții și prezidată** de **client** sau reprezentantul acestuia.
- Odată ce **există** o **Specificație a problemei acceptabilă**, în continuare, **dezvoltatorul** poate opera în aproape același mod ca cel descris în capitolele precedente până când se ajunge la **Faza de acceptanță** a **subsistemului programului**.
- Trebuie insistat asupra **testării de acceptanță a programelor realizate**, chiar dacă acceptarea atât de devreme va fi condiționată.
  - Este de așteptat ca multe **lucruri să nu se desfășoare** aşa cum trebuie pe măsură ce **subsistemul** va fi **integrat** mai târziu **cu celealte subsisteme**.
  - Când apar **probleme inevitabile**, va fi de nevoie pentru **dezvoltator** să poată folosi acele **teste de acceptanță condiționată** ca bază.
  - Dacă se omit aceste aspecte, fiecare **problemă** care apare este probabil să producă **o dispută lungă**, care ar putea merge până la cine a spus ce **în timpul fazei de definire**.
- **Faza de testare a sistemului** din Figura 10.1.1.a poate fi subdivizată într-o **fază „de laborator”** sau „controlată” și o fază „**în direct**” sau „**de teren**”.
  - În timpul **testării sistemului în laborator**, toate subsistemele pot fi legate între ele la **o instalație specială de testare** înainte de a fi trimise la un site real unde condițiile de testare pot fi mult mai aproximative.
    - De exemplu, dacă sistemul în curs de dezvoltare este un sistem de arme de rachete sol-aer, programul, computerul, subsistemele de comunicații, radar și rachete **pot fi integrate în laborator și testate în condiții parțial simulate**. Va fi necesar, totuși, ca testarea să se mute într-un site aflat la distanță, pentru a trage efectiv rachete către ținte.
- Teoretic, **toate testele** ar putea fi efectuate în **site-ul activ**, dar acest lucru **nu este de obicei** fezabil din punct de vedere economic, deoarece ar putea necesita **un număr**

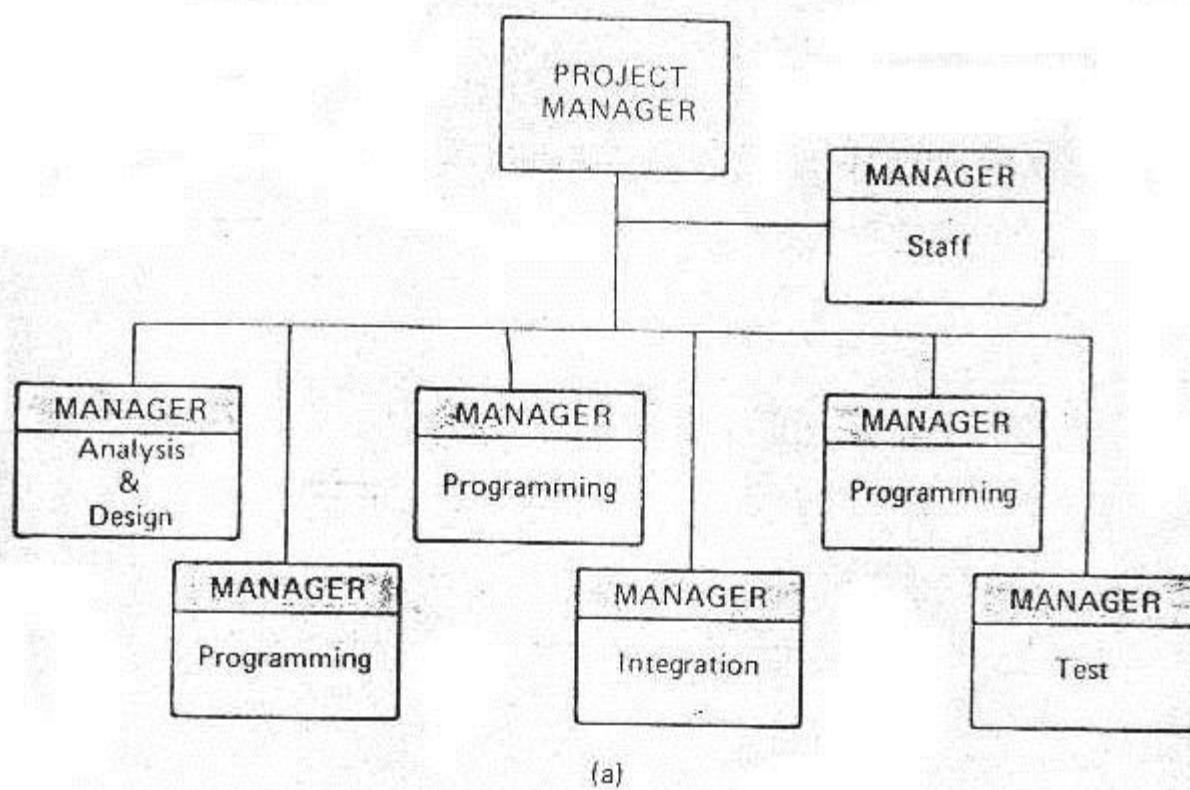
**mare de oameni și multe echipamente suplimentare** pentru a fi mutate departe pentru perioade lungi de timp.

- La **finalizarea testelor de laborator**, **problemele rămase vor fi minime** și vor fi necesare **mai puține persoane la locul de testare** în direct sau pe teren.
- **Faza de instalare de site și faza de testare** efectuate chiar **direct în site-ul real**, nu **reprezintă** nici pe departe **finalizarea proiectului**.
  - Copii ale sistemului vor fi trimise la fiecare dintre **multiplele locații geografice** unde acestea urmează **să devină operațional**.
  - La aceste **site-uri operaționale** vor fi efectuate **mai multe teste**, odată ce subsistemele sunt ajustate pentru **condițiile specifice site-urilor individuale**.
  - Toate aceste **teste pot dura**, desigur, **multe luni sau chiar ani**, mai ales într-o zonă atât de delicată precum **sistemele de apărare**.
  - În acest timp, **controlul modificărilor proiectului este deosebit de important** pentru **a preveni ca modificările urgente pe teren să contamineze copia principală a sistemului**.

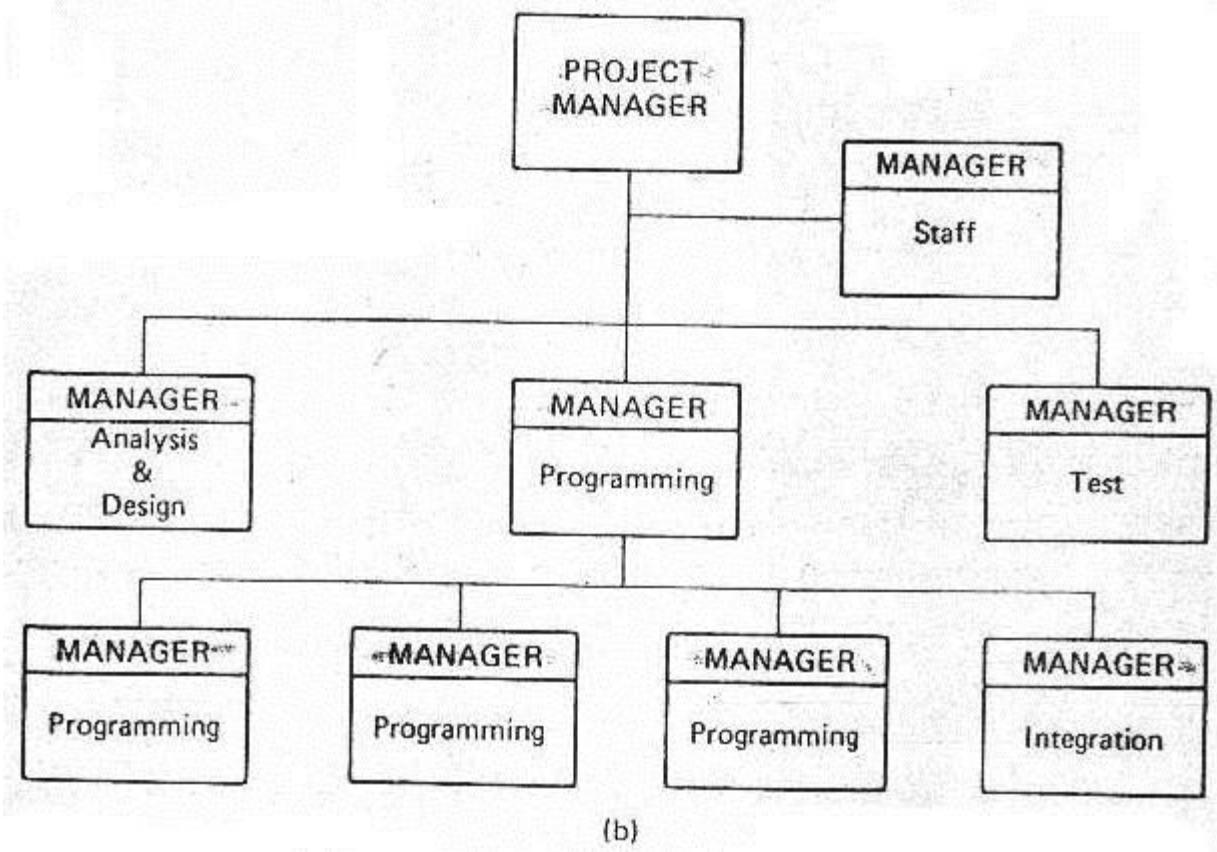
## 1.2 Organizarea

- **Organizarea de bază** din Figura 10.1.2.a. (b) este încă un bun punct de plecare, indiferent cât de mare este jobul.
- Cu toate acestea, pe măsură ce **numărul de oameni crește**, pot fi **adăugate mai multe niveluri de management**.
  - Partea **pozitivă** a acestui lucru este că se menține **un raport adecvat dintre manageri și programatori** (sau alți lucrători).
  - Partea **negativă** este că acum există **mai multe probleme de comunicare și mai multă fragmentare a jobului**.
- Pe lângă **dimensiunea** ca atare a lucrării de realizat, există mai multe **tipuri de funcții** care pot determina **creșterea organizației proiectului**.
  - (1) O cerință comună este scrierea **instrumentelor de suport de programare** care urmează să fie utilizate.
    - Poate fi necesar să fie scrise **compilatoare, pachete de intrare-ieșire, programe de urmărire, ajutoare speciale de testare** sau chiar un „**sistem de operare**” complet.
    - Deoarece toate acestea pot fi cu ușurință la fel de mari ca și jobul principala care trebuie realizat, este posibil să trebuiască ca în **ordinograma de organizare** să fie adăugată **o casetă mare, nouă**, sub **managerul de proiect numită „programe suport”**.
    - De asemenea, trebuie adăugată **o nouă secțiune** la **Planul de proiect** pentru a acoperi acest efort.

- (2) Este posibil să trebuiască să fie luată în considerare și „**diagnosticarea hardware**”.
  - Prin aceasta ne referim la **programe** care **verifică metodic hardware-ul** pentru a localiza **defecțiunile** existente sau pentru a încerca să prezică **defecțiuni iminente**.
  - Unii **producători de echipamente** furnizează aceste **programe** odată cu hardware-ul lor, dar poate fi necesar ca acestea să necesite **să fie complete**.
  - Este posibil să fie nevoie chiar **să se construiască un întreg sistem de astfel de programe de diagnosticare** care să fie legate la sistemul programului operațional astfel încât **să poată fi invocate automat**.
    - Încă o **casetă** sub managerul de proiect!
- Toate celelalte **casete** din Figura 10.1.2.a. (b) sunt susceptibile de a suferi o creștere **semnificativă** a numărului de persoane implicate, pe măsură ce se trece de la un proiect de **dimensiune medie** la un **proiect de mari dimensiuni**.



(a)



(b)

Fig.10.1.2.a. Organizarea convențională a proiectului

- Grupul de personal va fi întărit pentru a gestiona rapoarte de stare mai multe și mai mari și o grămadă mare de alte documente.
- Va fi necesar să se includă mai multe funcții de control, de exemplu, „Managementul configurației”
- Grupul de analiză și proiectare ar putea fi nevoie să crească pentru a efectua nenumărate studii și pentru gestionarea mai multor versiuni diferite ale sistemului de programe în evoluție. Acest grup poate face, de asemenea, o cantitate masivă de modelare și simulare.
- Funcțiile grupului de testare vor deveni probabil mult mai critice într-un sistem mare, deoarece devine necesară testarea abundantă a interfețelor cu alte subsisteme.
  - Alte casete în structura organizatorică a proiectului!

### 1.3 Controlul clientului

- Proiectele mari înseamnă bani mulți, iar banii mulți înseamnă oameni care îți respiră în ceafă.
- Clientul va insista asupra unui control mult mai mare asupra unui proiect de mai multe milioane de dolari decât asupra unui proiect cu un preț modest.
  - Acest lucru este de înțeles, dar ceea ce este supărător este muntele de hârtii care însoțește aceste controale.
- De exemplu, Guvernul Statelor Unite aplică pentru multe dintre contractele sale o schemă de control numită „Configuration Management”.
  - Vom defini acest lucru în secțiunea următoare, dar deocamdată trebuie să subliniem că este un sistem simplu din punct de vedere conceptual: se scrie o definiție de bază a ceva ce trebuie construit și se controlează toate modificările aduse acelei linii de bază. Ce poate fi mai simplu?
  - Dar grămada de manuale, regulamente, specificații și aşa mai departe care a fost construită în jurul acestui concept simplu sperie oamenii de moarte.
    - Ori de câte ori programatorii aud „Configuration Management”, ei se ascund în spatele răcitorului de apă, pentru că, după ce au văzut grămezile de manuale și nesfârșitele formulare, nu doresc sub nicio formă să aibă de a face cu ele.
- În realitate, nu este atât de dificil. Se poate folosi Configuration Management pentru un job foarte frumos dacă se poate găsi cineva care înțelege programarea și managementul programării și care scrie bine.
  - Acestei persoane i se cere să facă o lucrare de traducere.
    - Persoana în cauză își va petrece probabil o lună sau două studiind versiunea specială de management al configurației pe care clientul solicită să fie utilizată.

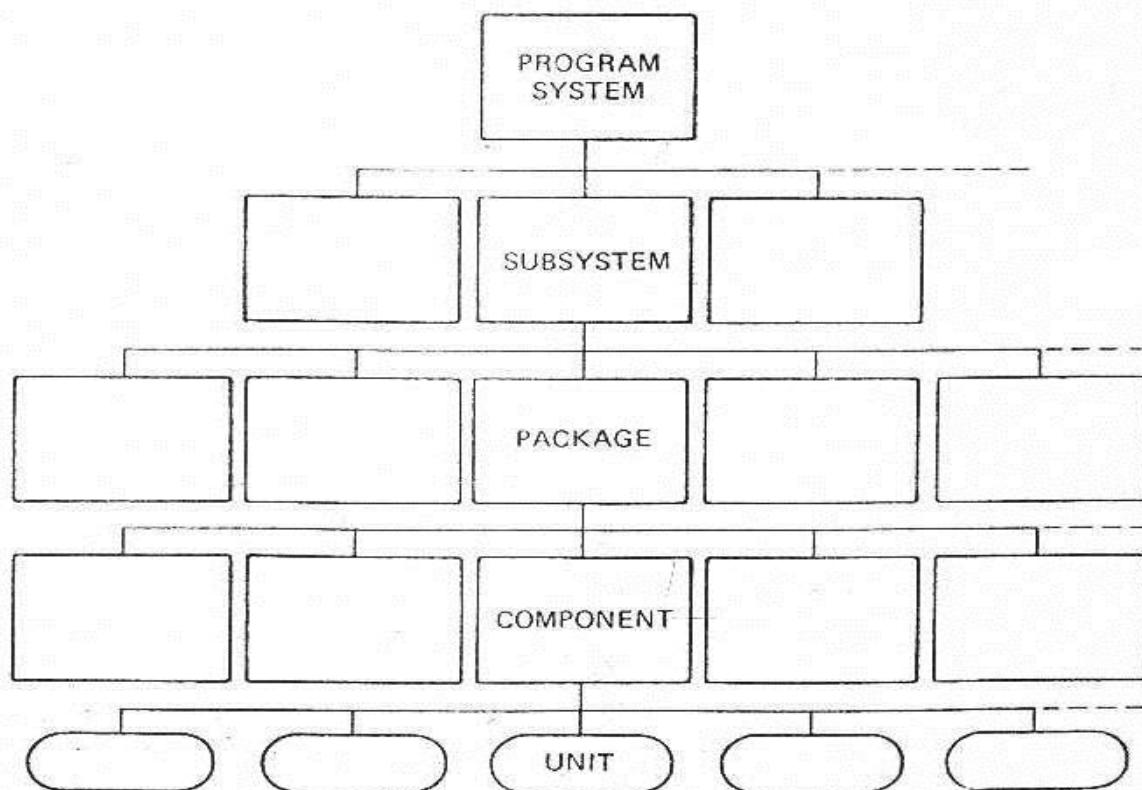
- În SUA, Există cel puțin trei scheme: Army, Air Force și NASA.
  - Toate sunt la fel din punct de vedere conceptual, dar sunt în mod natural diferite în implementare.
  - Dacă toate ar fi fost făcute exact la fel, multe sute de lucrători guvernamentali ar fi rămas fără loc de muncă!
- Acum, odată ce această persoană a înțeles ce este necesar, î se cere să facă o traducere pentru proiectul în discuție.
  - Manualele și alte materiale descriptive sunt scrise pentru a acoperi toate cazurile.
  - Tot ce are nevoie dezvoltatorul este ceva de aplicat proiectului curent.
    - Dacă scriitorul are talentul de a reduce lucrurile la esența lor, el va produce un ghid pentru organizația dezvoltatorului care va avea aproximativ 5% dimensiunea manualelor existente.
- Există și alte domenii în care clientul poate exercita controale dincolo de cele aplicate unui proiect mai mic.
  - De exemplu, clientul poate insista să trimită la unele dintre grupurile dezvoltatorului, proprii săi oameni pentru a influența designul, programarea și testarea, precum și pentru a pregăti un grup de oameni pentru a prelua în cele din urmă întreținerea sistemului.
- S-ar putea crede că toate acestea sunt probleme dificile pentru dezvoltator, dar ele de fapt nu sunt dacă se iau în considerare următoarele:
  - (1) Dezvoltatorul trebuie să încerce să aibă un anumit drept de veto asupra persoanelor pe care clientul î le „împrumută”.
    - Clientul poate încerca să aducă persoane care crează probleme. (Nu poate fi învinovățit pentru că a încercat!).
  - (2) Trebuie negociată o înțelegere clară a controlului dezvoltatorului asupra oamenilor clientului.
    - (a) Pe cât timp sunt împrumutați.
    - (b) Cine îi gestionează.
    - (c) Cine le evaluează munca.
      - Există mai multe exemple în care persoanele împrumutate au fost militari, dintre care unii au părăsit locul de muncă în momente critice când le-au expirat stagiile de serviciu sau când au fost transferați în alte zone.
  - (3) Nu trebuie considerat niciodată un angajat al clientului egal cu unul al dezvoltatorului și, prin urmare, să fie redusă estimarea în consecință a forței de muncă.
    - Dacă angajatul clientului tău se dovedește a nu fi la înălțimea cerințelor, apare o problemă serioasă.

- (4) Nu trebuie atribuită niciunua dintre oamenii clientului vreo sarcină de pe „drumul critic” al proiectului.
- Probabil că cele mai bune locuri pentru a folosi personalul clientilor sunt în zonele de analiză și testare și cele de pregătire a manualelor de utilizare.
  - În analiză ar trebui să adauge propria lor înțelegere unică a cerințelor jobului.
  - În testare, ele pot ajuta să fie evitate problemele ulterioare de acceptare.
  - În mod evident, au un interes personal să scrie documente cât mai bune pentru utilizator.

#### 1.4 Managementul configurării

- Acest concept a fost menționat în acest curs de mai multe ori.
- De regulă, el se schimbă constant fiind subiectul unei evoluții continue.
- Ceea ce urmează se bazează pe versiunea de gestionare a configurației pentru armata SUA.
  - Diferă de alte versiuni în detaliu precum terminologia, dar nu și în concept.
- Ideea de bază a managementului configurării pentru programare este următoarea:
  - (1) Se definește un produs program.
  - (2) Se controlează toate modificările aduse definiției originale.
  - (3) Se arată că produsul final este complet consistent cu definiția originală, luând în considerare toate modificările acceptate.
- Implementarea acestui concept de management al configurării presupune următoarele:
  - (1) Unitatea de bază de lucru care trebuie controlată se numește Element de configurare al unui program de calculator sau CPCl (Computer Program Configuration Item).
    - Un CPCl este o lucrare majoră, probabil de ordinul unui subsistem, în sensul indicat în Figura 10.1.4.a.

THE DESIGN PHASE



**Fig.10.1.4.a.** Program System Hierarchy

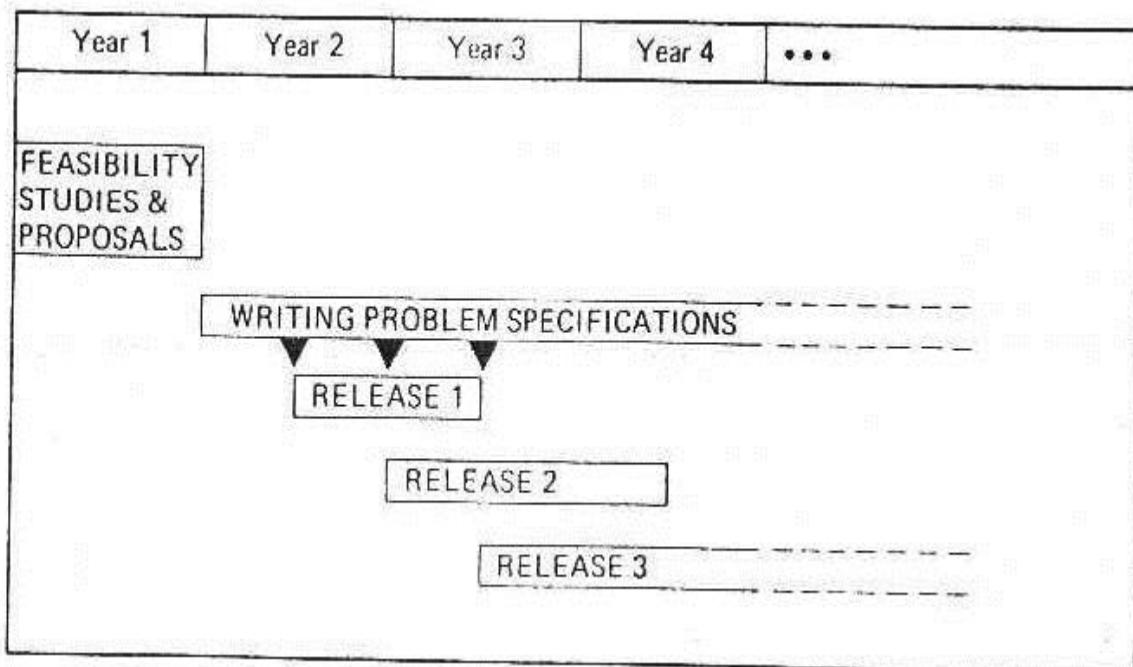
- (2) Sunt stabilite anumite **linii de bază** (baseline) pentru fiecare CPCI.
  - O **linie de bază** este orice lucru care poate fi folosit ca **punct de plecare**; este ceea ce dezvoltatorul și clientul sunt de acord că **va descrie sau constitui produsul**.
- (3) Orice lucru care se **îndepărtează** ulterior de la **linia de bază convenită** este considerat o **modificare**, iar modificarea trebuie **aprobată de client** în prealabil.
- Există trei **linii de bază**:
  - (1) „**Linia de bază funcțională**”, care este o **descriere la nivel brut a CPCI**, stabilită la sau înainte de **începerea proiectului**.
  - (2) „**Linia de bază alocată**”, o descriere mai detaliată a CPCI stabilită în **timpul fazei de definire**.
  - (3) „**Linia de bază a produsului**”, care **descrie CPCI** după ce a fost **construit și testat**.
- Procedura de **management al configurării** presupune:

- (1) Fiecare linie de bază succesivă devine noul standard în funcție de care sunt evaluate modificările pentru acel CPCI.
- (2) O modificare a unei linii de bază poate fi propusă de un membru al organizației clientului, de dezvoltator sau de oricare dintre ceilalți contractori ai proiectului.
- (3) Documentul oficial de propunere de modificare pentru majoritatea modificărilor se numește Propunere de modificare a ingineriei sau ECP (Engineering Change Proposal).
  - Termenul „inginerie” reflectă faptul că timp de mulți ani managementul configurației a fost orientat exclusiv în jurul hardware-ului.
- (4) Fiecare propunere de modificare a ingineriei este prezentată Consiliului de control al modificărilor (CCB – Change Control Board), care este format din reprezentanți din toate domeniile majore ale organizației dezvoltatorului.
  - Responsabilitatea consiliului este de a se asigura că modificarea propusă este solidă și, de asemenea, de a evalua impactul, dacă este cazul, asupra costurilor, planificărilor și liniilor de bază.
- (5) Dacă există un impact, modificarea se numește „Clasa I” și este trimisă Comitetului de control al modificărilor al clientului pentru aprobare.
  - Când clientul trimită înapoi aprobarea scrisă, modificarea poate fi încorporată.
- (6) Pentru modificările despre care Consiliul dezvoltatorului consideră că vor avea un impact neglijabil (numite modificări de „Clasa II”), modificarea nu trebuie să fie aprobată de client.
  - Clientul este însă informat cu privire la modificare și are dreptul de a o reclasifica în Clasa I.
- (7) Personalul de management al configurației ține evidență stării fiecărei modificări pentru fiecare versiune a programelor.
  - Personalul se asigură de asemenea, că se fac toate modificările necesare în documentație și că documentația și programele sunt ținute la zi.
- (8) La momente specifice, personalul organizează audituri în timpul cărora î se demonstrează clientului coerenta programelor și a documentației.

## 1.5 Versiuni (releases) multiple

- La un proiect mare va fi de obicei nevoie de mai mult de o singură versiune a întregului subsistem de programe.
  - Trei sau mai multe versiuni majore, sau „lansări”, nu sunt neobișnuite pentru un proiect care se întinde pe mai mulți ani.

- Motivul pentru mai multe versiuni este acela că unele sisteme sunt atât de dificile din punct de vedere tehnic și sunt atât de mari ca ampoare încât nu pot fi terminate într-un singur ciclu de dezvoltare.
  - Ele necesită **iterații**, eventual unele variante de încercare (trial-and-error), într-o construcție atentă a unui **sistem suprem**.
- **Versiunile multiple** pun noi **probleme de management**.
  - Cum se evită **confuzia nesfârșită** între versiuni?
- O **modalitate bună** de a soluționa această problemă este de **a programa începerea lucrărilor la fiecare nouă versiune**, astfel încât să coincidă cu publicarea unei noi **specificații a problemei**.
  - Dacă **se planifică** trei versiuni, **se recomandă planificarea** a trei specificații ale problemei, fiecare bazându-se pe cea precedentă.
- Figura 10.1.5.a prezintă un program tipic de versiuni multiple.



**Fig.10.1.5.a Scheduling Multiple Releases**

- Fiecare **săgeată neagră** din figură, reprezintă **momentul publicării unei versiuni noi**, mai complete a **specificației problemei**.
  - Fiecare **bară de versiune** care apare în figură reprezintă **un ciclu de dezvoltare a programului** mai mult sau mai puțin **complet**.

- Primul ciclu de versiune poate fi abreviat deoarece programele dezvoltate în timpul aceluia ciclu pot să nu fie suficient de complete pentru a fi supuse fazelor finale (testarea completă a sistemului și instalarea la un loc operațional).
  - Motivul pentru care începutul fiecărui nou ciclu de dezvoltare este legat de emiterea unei noi specificații de problemă este că modificările majore ale specificației pe o bază săptămânală sau lunară ar face aproape imposibilă producerea oricărei versiuni funcționale a sistemului.
    - În schimb, modificările pot fi acumulate în loturi și pregătite ca parte a următoarei specificații a problemei planificate.
- Lucrul la mai mult de o singură versiune la un moment dat face dificilă planificarea forței de muncă.
  - Dacă există trei versiuni în curs de desfășurare, nu pare rezonabil să se înființeze trei organizații complete pentru a le produce.
- Probabil cea mai bună abordare este de a avea un singur grup de analiză și proiectare care scrie specificații de problemă și de proiectare și de a segmenta grupurile de programare.

## 2 Proiecte de mici dimensiuni

- Multe proiecte mari ar putea fi de fapt proiecte mici.
  - Dacă ar înceta construcția de imperii, probabil s-ar putea face mai multe joburi de excelență și nu s-ar mai genera acea mediocritate uriașă care de regulă este asociată unor astfel de construcții. Se pot obține în mod evident performanțe mai bune.
- Materialul studiat în acest curs se aplică în egală măsură și acelor proiecte dezvoltate doar o mână de oameni, dar, desigur, scopul lucrurilor este diferit.
  - (1) Încă este nevoie de un plan de proiect, deși acesta poate avea doar patru pagini.
  - (2) Este nevoie de specificații pentru problemă, proiectare și codare.
  - (3) Încă trebuie planificată testarea sistemului.
  - (4) Trebuie făcute practic toate lucrurile despre care s-a discutat în curs dacă se dorește să se producă ceva de excelență.
- Diferența esențială constă în cantitatea de efort necesar.
  - (1) Într-un proiect de mici dimensiuni, o persoană se ocupă de mai multe joburi (activități).
  - (2) Unele sarcini care consumă cantități mari de energie în proiectele mari, de exemplu, testul de integrare, sunt realizate cu ușurință pe proiectele mici.
  - (3) Și, bineînțeles, tot acel ajutor suplimentar de management și personal necesar pentru un proiect de mari dimensiuni doar pentru a gestiona

„mărimea” și interacțiunile, pur și simplu nu există la un proiect de mici dimensiuni.

- Există însă un pericol pentru proiectele mici, care trebuie remarcat.
  - (1) Managerul devine adesea neglijent în controlul unui job mic care necesită doar câteva persoane.
  - (2) Există sentimentul că managerul este mereu deasupra tuturor, stăpânește lucrurile și că nimic nu ar putea merge prost.
    - Specificația de proiectare nu este scrisă (fiind simplă), cu atât mai puțin specificația problemei.
    - Programele nu sunt documentate temeinic pentru că, până la urmă, Charlie Programatorul are întregul sistem în cap. Deci, de ce să-l insultăm pe Charlie punându-l să scrie documentația?
    - Un plan de proiect este evident inutil – de ce să se scrie un plan pentru un job de câteva persoane?
  - (3) Managerul care lasă lucrurile să meargă astfel de fapt, caută cu lumânarea necazuri, iar când acestea vor veni, lucrurile vor arăta mult mai rău decât pentru un manager a douăzeci de persoane care are probleme.
- Este de așteptat ca un manager competent să poată conduce un proiect mic fără a falimenta compania.
  - Așadar, trebuie acordată cea mai mare atenție sfaturilor oferite în capitolele anterioare ale cursului prin adaptarea lor la jobul curent. Dacă în cele din urmă se dorește a se gestiona un proiect mai mare, trebuie început mai întâi cu unul mic.

### 3. Propunerea de proiect

- O propunere este oferta dezvoltatorului de a realiza un job.
  - Toate propunerile care apar trebuie tratate la fel.
- Atenție: multe propuneri promit lucruri care nu pot fi realizate.
  - De cele mai multe ori (dar nu întotdeauna) aceste propuneri sunt scrise de oameni competenți care au intenții bune.

#### 3.1 Ghid pentru redactarea unei propuneri de proiect

- Sunt bine cunoscute argumentele despre cât de greu este să scrii propuneri:
  - Nu este niciodată timp suficient.
  - Descrierea jobului este de regulă vagă.

- Cererea de propunere (RFP Request for Proposal) sau Caietul de sarcini este prost scris.
- Forțele de muncă adecvate nu sunt disponibile.
- S.a.m.d.
- Este adevărat că scrierea propunerilor este **grea**; cu toate acestea, există câteva linii directoare care pot ușura sarcina considerabil.
- (1) **Fiți selectivi.**
  - Nu scrieți o propunere pentru fiecare job care apare.
  - Concentrați-vă pe cele pe care le vreți cu adevărat, la care vă pricepeți și faceți o treabă de prim rang cu ele.
  - Multe companii lucrează pe baza faptului că numărul de contracte primite va fi într-un anumit raport cu numărul de propunerii generate.
    - Acest lucru este greșit. Numărul de propunerii de succes va fi în relație directă cu **selecția și pregătirea atentă a propunerilor**.
  - O propunere proastă este **mai rea** decât niciuna.
    - (1) În primul rând, **nu va câștiga**.
    - (2) În al doilea rând, **distrugе reputația companiei și șansele pentru viitor**.
    - (3) În al treilea rând, pentru **generarea sa s-a consumat forță de muncă valoroasă** care desigur **a afectat alte activități**.
- (2) **Tăiați lucrurile inutile** (umplutura).
  - Nu se rezolvă nimic prin includerea a trei pagini de propunere bună într-o crustă de umplutură.
    - Credeți că evaluatorilor de propunerii le place să citească toate aceste inutilități?
  - Textul propunerii trebuie să fie **cât mai concis**.
  - Decideți ce vreți să spuneți, spuneți și opriți-vă!
- (3) **Nominalizați un manager de propunere care are autoritate**.
  - Aceasta ar trebui să fie capabil să ia decizii rapide și să le **mențină**.
  - **Nu există loc** pentru a pierde timpul cu un lanț de management lung de un kilometru.
- (4) **Atribuiți sarcini de lucru specifice**.
  - Adunați-vă oamenii într-o **întâlnire de lansare** (kickoff meeting), **precizați clar obiectivele taskului de propunere și distribuiți munca**.
    - Unele dintre sarcini vor fi **investigative**.

- Unele vor implica scrierea unor secțiuni ale propunerii.
- După ce oamenii au avut șansa de a analiza câteva zile sarcinile primite, convocați o altă întâlnire și faceți toate ajustările necesare sarcinilor.
- În ceea ce privește modul de a atribui sarcini de lucru, în primul rând, se recomandă să utilizați Planul de proiect prezentat în capituloane anterioare.
- Propunerea va trebui să conțină în principiu, majoritatea sau toate elementele prezentate în Planul Proiectului.
- (5) **Schițați (structurați) propunerea.**
  - Faceți acest lucru din devreme.
    - Există multe exemple de nenumărate ore de muncă irosite pentru că nimeni nu s-a obosit să contureze documentul propunerii până târziu.
  - De obicei, fiecare dintre cei care scriu, își crează un cadru general și își exercită propriul stil de organizare, paragrafe și aşa mai departe, doar pentru a descoperi mai târziu că ceea ce au realizat nu se potrivește cu scrisul celorlalți.
    - Rezultat: rescrieri masive, risipitoare.
  - Stabiliti mai întâi schița propunerii și liniile directoare de scriere și în acest fel evitați astfel de situații.
    - Încă o dată, schița Planului de proiect vă poate oferi un început bun în conturarea documentului propunerii în sine.
- (6) **Planificați munca.**
  - Dezvoltarea unei propunerii este, până la urmă, un proiect în sine.
  - Acest proiect are obiective, termene limită și resurse limitate.
  - Planificați ce trebuie făcut, când și de către cine.
  - Planificați timp pentru revizuirea schițelor și eventuală rescriere.
  - Lăsați suficient timp pentru aprobare și posibilă revizuire de către conducerea superioară.
  - Nu uitați să acordați suficient timp pentru editarea tehnică, corectură, editarea finală, reproducere, colatare și distribuirea copiilor finale.
- (7) **Nu vă lăsați convinși să vă schimbați estimările pentru că „nu se vor vinde”.**
  - Dacă conducerea superioară decide să-și asume un risc de afaceri reducându-vă estimările, precizați-vă clar poziția.
  - Prin tacerea ta, poți sugerea faptul că ești de acord cu tăierea, dacă chiar ești împotriva ei.
- (8) **Precizați clar ipotezele avute în vedere.**

- Când **înaintați** conducerii dumneavoastră o propunere finalizată și un set de estimări, **includeți în scris** toate ipotezele pe care le-ați făcut pe parcurs, precum și **opinia dumneavoastră** cu privire la **riscurile inerente estimărilor**.
- (9) **Nu vă angajați excesiv.**
  - În majoritatea contractelor, **propunerea scrisă**, dacă **nu este modificată prin contract**, **este considerată un angajament**.
  - Dacă **promiți mai mult decât poți livra**, probabil **că vei rămâne blocat** în acel angajament.
- (10) **Fiți sinceri.**
  - Propunerile sunt de regulă **afaceri mari**.
    - Pot fi aventure pe viață și pe moarte pentru multe companii.
  - Ca atare, **tentăția de a ocoli adevărul**, de **a folosi cuvinte mari**, de **a ocoli unele probleme** este uneori **copleșitoare**.
  - De multe ori puteți auzi un asociat spunând că „*dacă suntem complet sinceri cu privire la această estimare, vom pierde, pentru că după cum știți, compania X nu va fi sinceră cu privire la estimarea ei*”.
    - Prostii! **Rezistați** acestui gen de gândire.
  - **Păstrați-vă integritatea intactă**.
    - Pe **termen lung**, **veți câștiga**.

### **Exercițiul # 10**

1. Care sunt caracteristicile unui proiect de mari dimensiuni? Cum poate managerul de proiect să gestioneze ampolarea proiectului?
2. Care sunt caracteristicile unui proiect de mici dimensiuni? Care sunt diferențele de management în comparație cu un proiect de dimensiune medie?
3. Care sunt recomandările pentru scrierea unei bune propunerii de proiect?