Proiectarea sistemelor software

7.1. Procesul de proiectare

Proiectarea software-ului implică următoarele stadii:

(1*) Studierea şi înţelegerea problemei*.

Problema trebuie examinată din unghiuri diferite, astfel încât să permită o privire din interior a cerinţelor de proiectare.

(2) *Identificarea caracteristicilor principale şi, în cele din urmă, o posibilă soluţie.*

- Adesea este util să se identifice mai multe soluţii şi să se evalueze fiecare dintre ele.
- Alegerea soluţiei depinde de experienţa proiectantului, de componentele reutilizabile pe care le are la dispoziţie, de simplitatea soluţiilor derivate.

(3) *Descrierea fiecărei abstractizări utilizate în soluţie.*

- Înainte de a crea documentaţia formală, totuşi proiectantul trebuie să stabilescă dacă este necesar să construiască o descriere informală a proiectului.
- Erorile şi omisiunile din nivelurile înalte ale proiectării, care sunt descoperite de-a lungul proiectării la nivelurile scăzute, pot fi corectate înainte ca proiectul să fie documentat.

Un model general de proiectare a software-ului este un graf orientat.

- Nodurile grafului reprezintă entităţile de proiectare, cum ar fi procesele, funcţiile sau tipurile, iar legăturile reprezintă relaţiile existente între entităţi.
- Scopul procesului de proiectare este de a crea un asemenea graf fără inconsistenţe şi în care toate relaţiile dintre entităţi sunt legale (posibile).

Proiectantul începe cu o imagine foarte neformală a proiectului şi o rafinează, adăugându-i informaţie pentru a realiza un proiect mai bine formalizat (Fig.7.1).

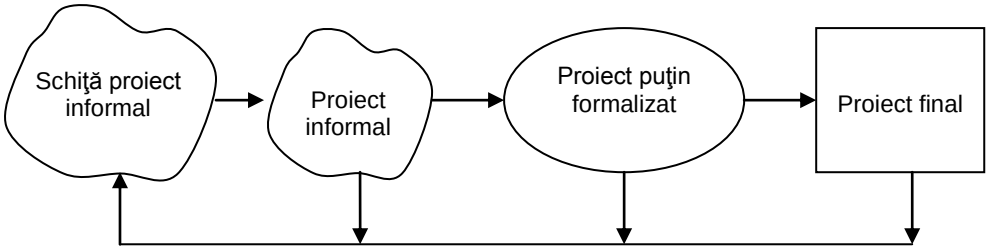

Fig.7.1. Procesul de proiectare

Procesul de proiectare implică descrierea sistemului într-un număr diferit de niveluri de abstractizare, permiţând astfel descoperirea mai devreme a omisiunilor sau a erorilor.

Această reacţie creează posibilitatea ca diferitele stadii de proiectare să fie rafinate.

Fig.7.2. prezintă stadiile procesului de proiectare şi al descrierilor de proiectare produse ca rezultat al acestor activităţi.

Ieşirea fiecărei activităţi de proiectare este o specificaţie.

Astfel, procesului de proiectare continuă, adăugându-se din ce în ce mai multe detalii la specificare.

Ultimile ieşiri sunt specificaţiile algoritmilor şi ale structurilor de date care vor constitui baza pentru implementarea sistemului.

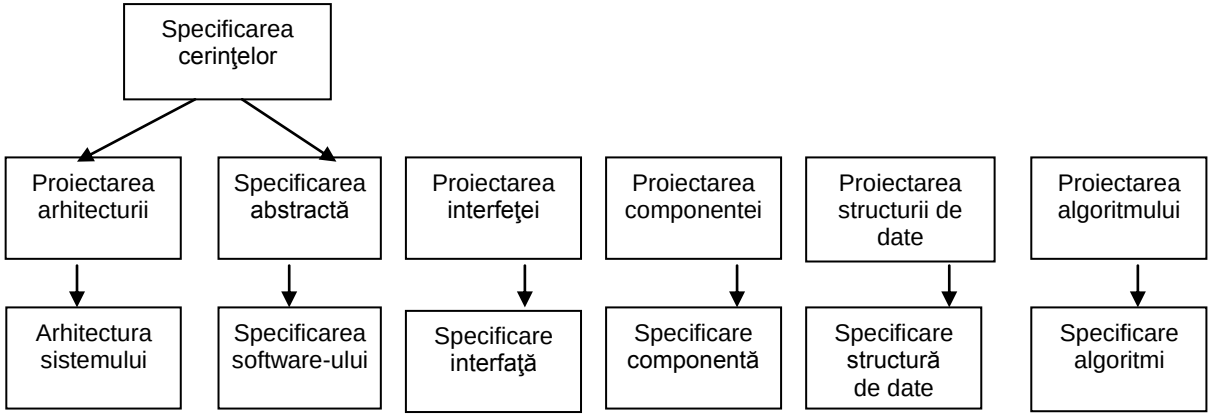

Fig.7.2. Activităţi de proiectare şi produse proiectate

Activităţile prezentate în fig. 7.2 sunt esenţiale în proiectarea sistemelor software mari şi cuprind următoarele:

(1). Proiectarea arhitecturii. Se realizează subsistemele întregului sistem identificându-se şi documentându-se relaţiile dintre ele.

(2). Specificarea abstractă. Se generează specificări abstracte ale serviciilor pentru fiecare subsistem în parte şi se stabilesc restricţiile sub care va opera sistemul.

(4). Proiectarea componentei. Serviciile produse de subsistem sunt partiţionate în funcţie de componentele din acel subsistem.

(5). Proiectarea structurilor de date. Structurile de date folosite în implementarea sistemului vor fi proiectate în detaliu şi specificate.

(6). Proiectarea algoritmului. Algoritmii utilizaţi pentru a furniza servicii vor fi proiectaţi în detaliu şi specificaţi.

Procesul este repetat pentru fiecare subsistem, până când componentele identificate pot fi transpuse direct în componente ale unui limbaj de programare cum ar fi package, proceduri sau funcţii.

O abordare recomandată pentru proiectarea pe scară largă este proiectarea top-down în care problema este recursiv împărţită în subprobleme, până când sunt identificate toate subproblemele elementare.

Forma generală a proiectului care de obicei iese la iveală dintr-o astfel de abordare este aproximativ ierarhică (fig.4.3), legăturile de încrucişare observându-se în graf pe nivelurile scăzute ale arborelui de proiectare, astfel încât proiectanţii pot identifica posibilităţile de reutilizare.

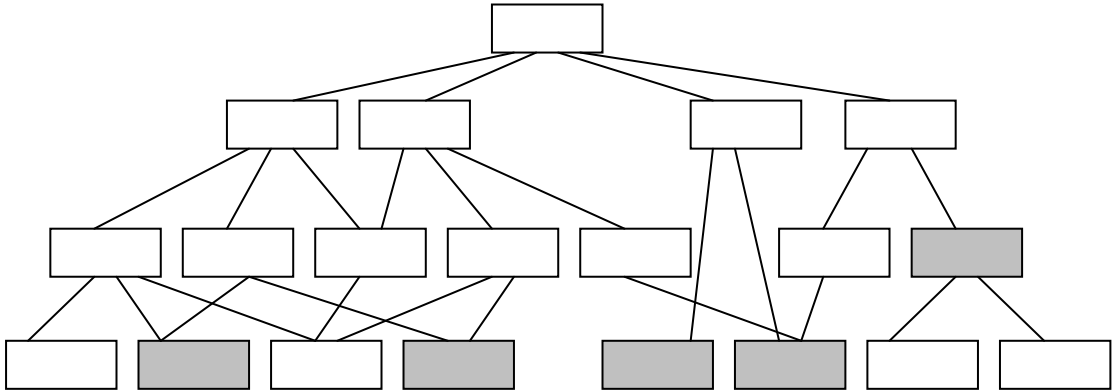

Fig.7. 3. Structura proiectării software

De fapt nu este recomandabil să se proiecteze sistemele într-o manieră care este strict top-down.

Proiectanţii îşi utilizează adesea experienţa anterioară de proiectare şi nu au nevoie întotdeauna să descompună toate abstractizările, ştiind exact care parte a sistemului poate fi construită.

Proiectarea top-down a fost propusă în conjuncţie cu descompunerea funcţională şi este o abordare validă în care componentele proiectate sunt strâns legate.

Totuşi când este adoptată o proiectare orientată pe obiect şi multe obiecte existente urmează să fie reutilizate, proiectarea top-down nu este cea mai indicată.

Proiectantul utilizează obiectele existente într-o reţea de proiectare şi construieşte proiectul cu ele.

Nu există nici un concept de ierarhizare a tuturor obiectelor existente într-un singur obiect ierarhic.

7.2. Proiectarea arhitecturală

Este etapa în care se construieşte **soluţia problemei.** Rezultatul este **un model fizic** al viitorului sistem, care este descris în **Documentul de Proiectare Arhitecturală.**

Cerinţele din documentul Cerinţelor Software sunt alocate unor componente ale viitorului sistem. Rezulta un set de subsisteme/componente interconectate. Arhitectura software rezulta printr-un proces iterativ de descompunere a cerinţelor. Documentul de proiectare arhitecturala trebuie sa specifice rolul fiecarei componente, cerinţele care i-au fost alocate, interfaţa de comunicare cu celelalte componente ale sistemului. De asemenea, in etapa de proiectare arhitecturala se întocmeşte **Planul Testelor de Integrare.**

7.2.1. Procesul: obţinerea modelului de proiectare arhitecturala

1. Abordare descendenta

- Se pleaca de la specificatia cerintelor software: S
- Se descompune setul cerintelor, S, in subseturi relativ independente, S1, S2,

- Fiecare subset de cerinte, Si, este alocat unui subsistem al arhitecturii
- software: SA1, SA2...
- Fiecare subset de cerinte Si este descompus in subseturi mai simple, Si1, Si2, .. care sunt alocate unor subsisteme ale subsistemului SAi: SAi1, SAi2,..
- .....

➢ **Descompunerea modelului de proiectare arhitecturala se opreşte atunci cand nivelul sau de detaliu permite**:

- continuarea dezvoltarii in paralel de catre mai multi membrii ai echipei de dezvoltare,
- planificarea activitatilor urmatoare ale procesului de dezvoltare (pana la livrare),
- estimarea resurselor umane necesare si a costurilor.

Rezultatul este o structura ierarhica alcatuita din subsisteme interconectate, fiecare subsistem fiind descompus pana la nivel de module.

➢ **Un modul de proiectare poate fi**: modul functional (functie, procedura), clasa, componenta, in functie de metoda de proiectare folosita: functionala sau orientat obiect.

- Fiecarui modul de proiectare i s-a alocat un set de cerinte pe care trebuie sa le implementeze.

- Se incearca gasirea unor module existente care ar putea fi folosite in implementarea modulelor definite in procesul de proiectare.

2. Abordare ascendentă

- Centrata pe reutilizare
- Se pleaca de la un set de module existente : module functionale sau clase
- Se cauta o descompunere a cerintelor in subseturi care pot fi implementate folosind componentele existente

3. Abordare hibrida

Se pleaca de la setul de cerinte software, care se descompune iterativ, dar in procesul de descompunere se urmareste definirea de module care pot fi implementate folosind module existente.

7.2.2. Modelul de proiectare athitecturală

- Este o structură ierarhică realizată din subsisteme interconectate, fiecare subsistem fiind alcatuit dintr-un set de module interconectate sau din alte subsisteme, s.a.m.d.
- Modelul poate fi reprezentat prin :

- diagrame de componente si diagrame de distributie combinate cu diagrame de componente
- diagrame de clase, in care relaţiile ierarhice se bazează pe generalizare si specializare
- diagrame de structura (Fig.7.4), in cazul unei descompuneri funcţionale : descompunerea iterativa a funcţiilor pe care trebuie sa le implementeze sistemul

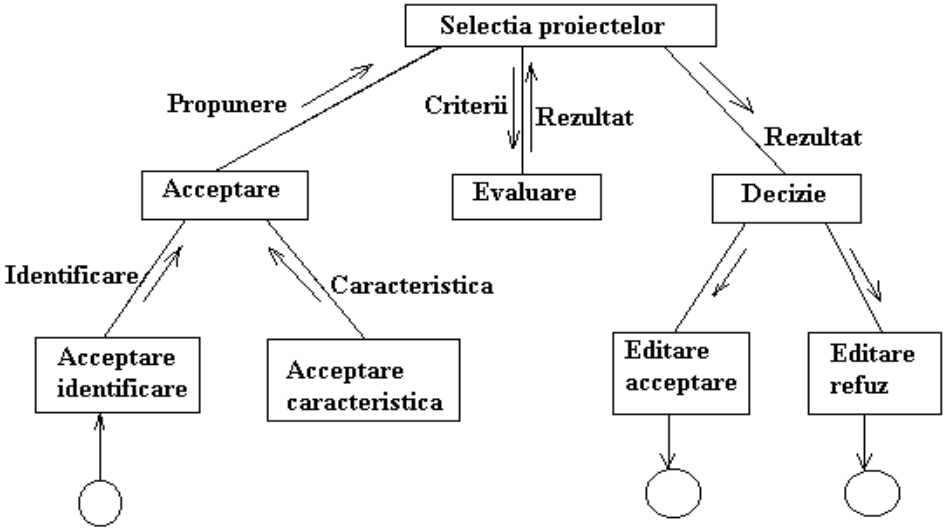

Fig.7.4. Diagrama de structura

- Nodurile arborelui sunt module functionale.
- Arcele reprezinta relatiile de apel intre module.
- Sagetile indica fluxul datelor

Pentru fiecare modul al arhitecturii software se scrie o specificatie:

Fiecare modul este specificat prin:

- Identificatorul(unic) si tipul sau (clasa, functie, fisier, program)
- Scopul sau – cerintele software pe care le implementeaza
- Componenetele subordonate: modulele apelate/ obiectele utilizate/ fisierele unei baze de date
- Interfata componentei: fluxul datelor si al controlului
- Dependentele sale: conditii care trebuie sa fie satisfacute inainte sau dupa executia sa, operatii interzise in timpul executiei componentei
- Prelucrarea interna a componentei, la nivelul cel mai coborat in limbaj natural
- Structurile de date interne

Planul testelor de integrare precizeaza ordinea in care vor fi integrate modulele in subsisteme si apoi subsistemele pana la nivel de sistem precum si testele care vor fi efectuate la integrare.

7.3. Proiectarea calităţii

Nu există nici în momentul de faţă o modalitate absolut garantată care să ateste că un proiect este fără nici o eroare. În funcţie de domeniul de aplicaţie şi de cerinţele proiectului, un proiect bun ar putea fi proiectul care permite producerea unui cod eficient, sau ar putea fi proiectul cu dimensiunea cea mai mică posibilă, pentru care implementarea să fie cât mai compactă, sau proiectul care asigură mentenanţa cea mai bună.

Un proiect mentenabil poate fi cu rapiditate adaptat pentru a i se adauga noi functionalităţi sau pentru a le modifica pe cele existente. Proiectul trebuie să fie inteligibil, iar schimbările să aibă un efect local. Componentele proiectului trebuie să aibă coeziune, ceea ce înseamnă că toate părţile componente să fie legate logic între ele. Ele trebuie să poată fi cu uşurinţă decuplate, cuplajul fiind o măsură a independenţei componentelor.

Multe preocupări şi cercetări ştiinţifice au avut în vedere stabilirea unor metrici de proiectare a calităţii, care să ateste în final că un proiect este bun. Cele mai multe asemenea metrici au fost dezvoltate în conjuncţie cu metodele structurate ale lui Yourdan. Caracteristicile de calitate sunt în mod egal aplicabile atât proiectării orientate pe obiect cât şi proiectării oriantate funcţional, şi sunt : coeziunea, cuplajul, inteligibilitatea, adaptabilitatea.

7.3.1. Coeziunea

Coeziunea unei componente este o măsură a cât de bine se potriveşte ea logic cu celelalte. O componentă poate implementa o singură funcţie logică sau o singură entitate logică şi toate părţile componentei trebuie să contribue la această implementare. Dacă componenta include părţi care nu sunt direct legate de funcţia ei logică (de exemplu, dacă există un grup de operaţii care se execută în acelaşi timp şi nu au legătură cu funcţia), aceasta înseamnă că gradul de coeziune este scazut.

Constantine şi Yourdan (1979) au identificat şapte niveluri de coeziune în ordine crescătoare a gradului, şi anume:

1). *Coeziune de coincidenţă*. Părţile componente nu sunt într-o relaţie, ci pur şi simplu sunt asamblate într-o singură componentă.

2). *Asociere logică.* Componentele care realizează funcţii similare, cum ar fi operaţii de intrare, tratarea erorilor,etc., sunt asamblate împreună într-o singură componentă.

3). *Coeziune temporală*. Toate componentele care sunt active la un moment de timp, cum ar fi cele necesare la pornire sau la oprire, trebuie să fie la un loc.

4). *Coeziunea procedurală*. Elementele dintr-o componentă trebuie să aibă o singura secvenţă de control.

5). *Coeziune de comunicare.* Toate elementele unei componente operează pe aceleaşi date de intrare sau produc aceleaşi date de ieşire.

6). *Coeziune secvenţială*. Ieşirea dintr-un element al componentei serveşte ca intrare pentru alt element.

7). *Coeziune funcţională.* Fiecare parte a componentei este necesară pentru execuţia unei singure funcţii.

Aceste clase de coeziune nu sunt în mod strict definite, iar Constantine şi Yourdan le-au ilustrat prin exemple. Nu este întotdeauna uşor să se facă o clasificare a unui sistem într-o anumită categorie de coeziune.

Metoda lui Yourdan este aplicabilă şi este evident că cea mai bună formă de coeziune a unei unităţi este funcţia.Totuşi, cel mai mare grad de coeziune îl au sistemele orientate pe obiecte şi acest lucru reprezintă de fapt o caracteristică a acestora. Într-adevar, unul dintre principalele avantaje ale acestei modalităţi de proiectare este acela că obiectele creează sistemului o coeziune naturală.

Un obiect coerent (care are coeziune) este acela în care este reprezentată o singură entitate şi în care sunt incluse toate operaţiile pentru acea entitate. În acest context se poate defini următoarea clasa de coeziune:

-*Coeziune de obiect*. Fiecare operaţie are ca rezultat acea funcţionalitate care să permită obiectului să poată fi modificat, inspectat sau utilizat ca sursă de servicii.

Coeziunea este o caracteristică de dorit deoarece aceasta înseamnă că un modul reprezintă o singură parte din soluţia problemei. Dacă este necesar să se modifice sistemul, acestă parte există într-un singur loc şi orice trebuie făcut cu ea se află încapsulat într-o singură unitate.

Dacă funcţionalitatea este furnizată de un sistem-obiect care utilizează moştenirea din super-clase, coeziunea obiectului care moşteneşte atribute şi operaţii este redusă. Nu este posibil mult timp să se considere obiectul ca o unitate distinctă. Toate super-clasele vor putea de asemenea să fie inspectate dacă funcţionalitatea unui obiect nu este în totalitate înţeleasă.

7.3.2. Cuplajul

Cuplajul este legat de coeziune şi este un indicator al tăriei intercorelării dintre unităţile programului.

Sistemele cu cuplaj ridicat au interconectări puternice cu unităţile din program care depind unele de altele. Sistemele cu cuplaj slab sunt construite din unităţi independente sau aproape independente.

Ca reguli generale: toate modulele au cuplaj ridicat dacă ele utilizează variabile în comun sau dacă ele îşi schimbă informaţia de control. Constantine si Yourdan au numit această *cuplare comună* sau *cuplare prin control*. Cuplajul slab este obţinut prin asigurarea ca, atât timp cât este posibil, informaţia reprezentativă este păstrată în interiorul unei componente, iar interfaţa de date cu alte unităţi se realizează numai prin lista de parametri. Dacă este nevoie să fie partajate datele, această partajare poate fi controlată, cum este în ADA în cazul “package”. Aceasta se numeşte cuplare prin date. **Fig. 7.5 si 7.6** ilustrează cuplajul puternic, respectiv cuplajul slab al modulelor.

O problema a cuplajului provine din faptul că se pot cupla diverse module prin valori de variabile. Orice schimbare a acestor valori presupune o schimbare în program.

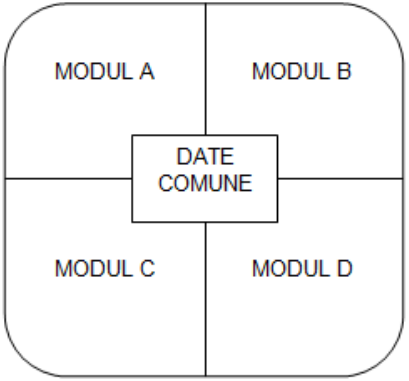

Fig.7.5. Cuplaj puternic

Se pare ca principalul avantaj al proiectării orientate pe obiect rezidă din faptul că proiectele rezultate manifestă un cuplaj slab.

Principala caracteristică a acestei abordari orientate pe obiect este tocmai aceea că “ascunde” ceea ce este în interiorul obiectului astfel încât sistemul nu trebuie să aibă o stare partiţionată cu nici un alt obiect, iar un obiect poate fi la rândul lui înlocuit de altul, cu aceeaşi interfaţă.

Moştenirea în sistemele orientate pe obiect are o formă definită de cuplaj.

Obiectele care moştenesc atribute şi operaţii sunt cuplate cu super-clasele lor, iar schimbările făcute în superclase trebuie operate cu grijă, deoarece acestea se propagă în toate clasele care moştenesc acea super-clasă.

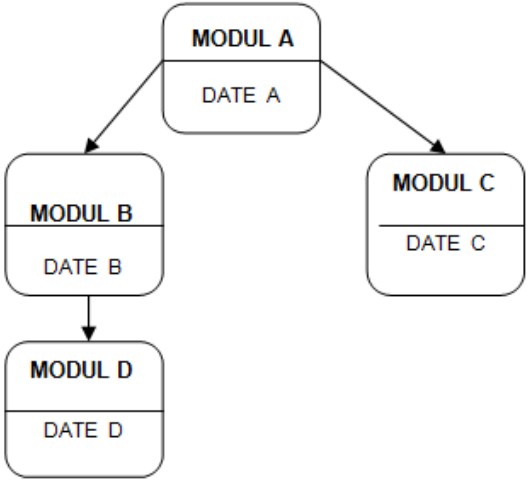

Fig.7.6 Cuplaj slab

7.3.3. Inteligibilitatea

Schimbarea componentei unui proiect are drept implicaţie ca persoana responsabilă de acea schimbare să înţeleagă operaţia facută. Acest lucru se referă la câteva caracteristici ale componentei şi anume:

(1) Coeziunea. Poate acea componenta să fie înţeleasă fără alte referiri la celelalte componente?

(2) Numele. Sunt numele utilizate în înţelesul componentei?

(3) Documentare. Este componenta documentată astfel încât să reflecte o legatură clară între entităţile din lumea reala şi componentă?

(4) Complexitate. Cât de mare este complexitatea algoritmilor utilizaţi în implementarea componentei? În acest context se utilizează complexitatea într-o manieră neformală.

Complexitatea înaltă implică multe relaţii între diferite componente ale proiectului şi o structură logică complexă, care poate implica secvenţe “if-then-else” imbricate.

Complexitatea componentelor este greu de înţeles şi de aceea proiectantul trebuie să se străduiască să conceapă componente pe cât posibil mai simple.

Cel mai mare efort în stabilirea de metrici pentru calitatea proiectului este concentrat pe încercarea de a asigura complexitatea componentei, obţinându-se astfel o măsură a inteligibilităţii componentei. Complexitatea afectează înţelegerea, dar sunt şi alţi factori care o afectează, cum ar fi organizarea datelor şi stilul în care proiectul este descris.

Măsurile complexităţii pot numai să furnizeze un indicator al inteligibilităţii componentei. Moştenirea în proiectele orientate pe obiect afectează înţelegerea.

Moştenirea este folosită pentru a ascunde detalii de proiectare, iar proiectul este uşor de înţeles.

Pe de altă parte, utilizarea moştenirii solicită celui care citeşte proiectul să privească la multe clase diferite de obiecte din ierarhia de moştenire, iar înţelegerea proiectului este redusă.

7.3.4. Adaptabilitatea

Pentru ca un proiect să fie bine întreţinut, el trebuie să poată fi cu uşurinţă adaptat diverselor cerinţe survenite ulterior. Aceasta implică, subînţeles, ca toate componentele să fie cuplate slab. Mai mult decât atât, adaptabilitatea înseamnă ca proiectul să fie bine documentat, documentaţia

componentelor să poată fi înţeleasă cu uşurinţă în concordanţă cu implementarea, iar implementarea să fie scrisă într-o formă accesibilă.

Un proiect adaptabil trebuie să aibă un nivel înalt de vizibilitate şi trebuie să existe o relaţie clară între diferitele niveluri ale proiectului. Trebuie să fie posibil pentru cititorul proiectului să găsească relaţia dintre reprezentarea ca o diagramă de structuri şi reprezentarea transformării datelor (Fig.7.7).

De asemenea, trebuie să fie uşor să se încorporeze schimbările făcute din proiect în toate documentele proiectului. Pentru o adaptabilitate optimă componenta trebuie să se conţină numai pe sine.

Adaptabilitatea unei componente poate să implice schimbări numai ale unei părţi a componentei care se referă la funcţiile externe, astfel ca specificarea acestor funcţii externe să fie de asemenea luată în consideraţie de către cel care face modificarea.

Unul dintre principalele avantaje ale moştenirii sistemelor orientate pe obiect este acela că în acest caz componentele pot fi adaptate cu uşurinţă.

Mecanismul de adaptare nu se referă la modificarea componentei, ci la crearea unei noi componente care moşteneşte atributele şi operaţiile componentei originale.

Numai atributele şi operaţiile care trebuie schimbate sunt modificate. Această adaptabilitate simplă este motivul pentru care limbajele orientate pe obiect sunt atât de eficiente pentru prototipizarea rapidă.

Totuşi, pentru sistemele cu viaţă scurtă, trebuie avut în vedere că moştenirea devine o problemă atunci când sunt operate din ce în ce mai multe schimbări şi reţeaua de moştenire devine din ce în ce mai complexă.

Funcţionalitatea este adesea distribuită în diverse puncte ale reţelei, iar componentele sunt în acest caz greu de înţeles.

Experienţa în programarea orientată pe obiect a arătat că reţeaua de moştenire trebuie periodic revizuită şi restructurată pentru a se reduce complexitatea şi duplicarea funcţionalităţii. În mod clar, aceasta face să crească costul schimbării sistemului.

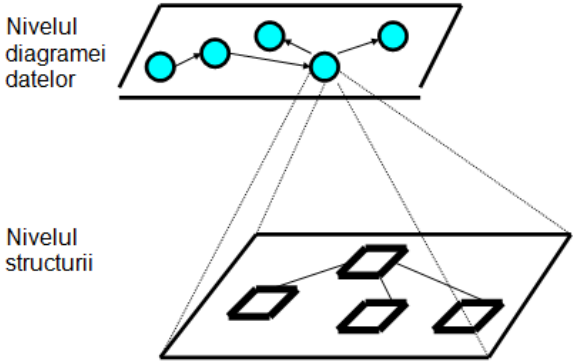

Fig.7.7 Diagrama de structură şi reprezentare a datelor

7.4. Modularizarea proiectului

7.4.1. Principii de modularizare:

- Modulele trebuie sa fie simple si cat mai independente unul de altul:
    - O modificare a unui modul are influenta minima asupra altor componente
    - O schimbare mica a cerintelor nu conduce la modificari majore ale arhitecturii software (gruparea cerintelor corelate in acelasi modul)
    - Efectul unei conditii de eroare este izolat in modulul are a generat-o
    - Un modul poate fi inteles ca o entitate de sine-statatoare
- Modulele trebuie sa “ascunda” modul de implementare a functiilor descrise de interfata lor, de ex. cum sunt memorate datele cu care lucreaza (tablou, lista, arbore, in memorie sau intr-un fisier)

7.4.2. Reguli de proiectare a modulelor:

- **Minimizarea cuplarii** intre module:
    - Minimizarea numarului de elemente prin care comunica modulele;
    - Evitarea cuplarii prin structuri de date
    - Evitarea cuplarii prin variabile “steag” (cuplarea prin control)
    - Evitarea cuplarii prin date globale
- **Maximizarea coeziunii interne** a fiecarei componente: elementele grupate intr-o componenta trebuie sa fie corelate, de ex. sa contribuie la aceeasi prelucrare
- **Restrangerea numarului de module apelate (fan-out)** de un modul:
- **Maximizarea numarului de module care utilizeaza un modul (fan-in)** – incurajaza reutilizarea (Fig. 7.8)

  “Fan-in” mare: un numar mare de module depind de el

„Fan-out“ mare: modulul depinde de multe module

- **Factorizare**: functionalitatile comune sunt definite in module reutlizabile.

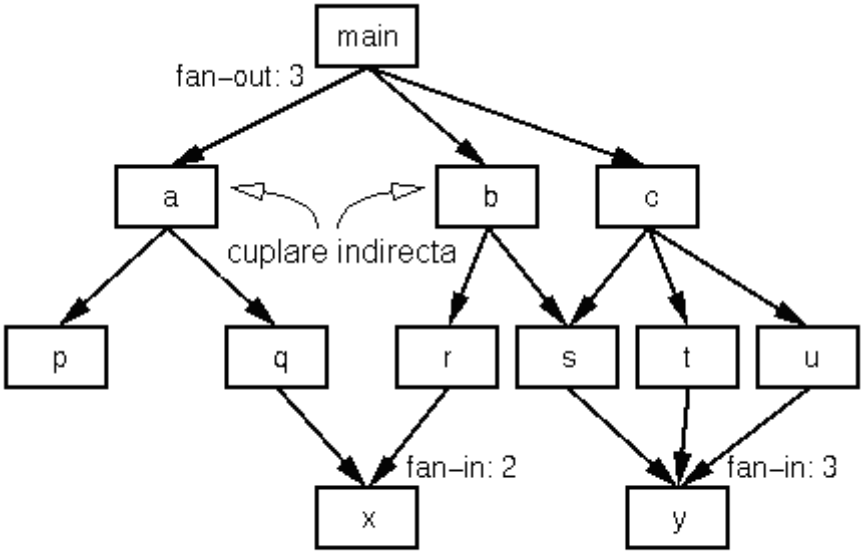

Fig.7.8. Exemplu de diagramă de structură

Documentul de proiectare arhitecturala (ADD)

Şablonul documentului in standardele ESA:

a.	Abstract	
b.	Table of Contents	
c.	Document Status Sheet	Status sheet for configuration control.
d.	Document Change Records since previous issue	A list of document changes.
1. Introduction		
1.1	Purpose	The purpose of this particular ADD and its intended readership.
1.2	Scope	Scope of the software. Identifies the product by name, explains what the software will do.
1.3	List of definitions	The definitions of all used terms, acronyms and abbreviations.
1.4	List of references	All applicable documents.
1.5	Overview	Short description of the rest of the ADD and how it is organized.
2. System overview		Short introduction to system context and design. Background of the project.
3. System context (for each external interface ...)		
3.n	External interface definition	The relationship with external system n.
4. System design		

4.1	Design method	Name and reference of the method used.
4.2	Decomposition description	Overview of components: decomposition, dependency or interface view.
5. Component descriptions (for each component ...)		
5.n	Component identifier	A unique identifier.
5.n.1	Type	Task, procedure, package, program, file, ...
5.n.2	Purpose	Software requirements implemented.
5.n.3	Function	What the component does.
5.n.4	Subordinates	Child components (modules called, files composed of, classes used).
5.n.5	Dependencies	Components to be executed before/after, excluded operations during execution.
5.n.6	Interfaces	Data and control flow in and out.
5.n.7	Resources	Needed to perform the function.
5.n.8	References	To other documents.
5.n.9	Processing	Internal control and data flow.
5.n.10	Data	Internal data.
6. Feasibility and resource estimates		A summary of computer resources needed to build, operate and maintain the software.
7. Requirements traceability matrix		A table showing how each software requirement of the SRD is linked to components in the ADD.

7.4.3. Proiectarea de detaliu

- Se efectuează la nivelul modulelor definite in proiectarea arhitecturala.
- Poate avea loc in paralel, pentru diferite module.
- Detaliază modelul de proiectare arhitecturala:
  - pot fi definite module de nivel mai coborât
  - se detaliază componenta claselor: atributele si funcţiile membre
  - se aleg biblioteci utilizate in implementare
  - se încurajează reutilizarea
  - sunt descrişi algoritmii