reprezentare a obiectelor aparţinând tipului de date al structurii în funcţie de obiectele altei structuri de date T. Fiecare operaţie a lui S trebuie definită în termenii operaţiilor lui T

Cele mai întâlnite tipuri primitive în limbajele moderne de programare sunt:

a) tipuri numerice: ele sunt reprezentări ale unor mulţimi de numere cunoscute din matematică, precum N, Z sau R. Vom avea în consecinţă date întregi fără semn, întregi cu semn sau reali. Numerele întregi se reprezintă intern în calculator sub forma unei succesiuni de biţi, cu ajutorul codului complement (cod complement faţă de 2). Numerele reale se reprezintă intern în calculator cu ajutorul reprezentării în virgulă mobilă (flotantă), care cuprinde semnul, mantisa şi exponentul. Pentru a reprezenta corect şi numerele subunitare, se utilizează reprezentarea cu semn, caracteristică şi exponent.

b) tipul caracter: este destinat reprezentării caracterelor alfanumerice. Tipul caracter modelează mulţimea finitãa a tuturor caracterelor afişabile. Din nefericire, nu există un set standard de caractere specific tuturor sistemelor de calcul. Din acest motiv, termenul "standard" trebuie interpretat în acest context ca referindu-se la sistemul de calcul particular pe care este executat programul în cauză. Cel mai folosit set "standard" de caractere este cel definit de Organizaţia Internaţională pentru Standarde (ISO) şi anume versiunea sa americană (codul ASCII), care constă din 128 de caractere, dintre care primele 33 sunt caractere de control şi celelalte 95 sunt caractere afişabile.Caracterele sunt codificate intern prin valori întregi numite coduri. Deoarece codul ASCII are 128 de caractere, pentru codificarea caracterelor sale sunt suficienţi 7 biţi. Cu toate acestea se foloseşte pentru codifiare un octet, deoarece memoria internă a sistemelor de calcul este structurată de obicei sub forma unei secvenţe de locaţii cu dimensiunea de un octet. Pentru seturile cu peste 256 de caractere, a fost introdusă specificaţia standard Unicode (cum este cazul limbii chineze sau japonenze), pentru reprezentarea căreia se folosesc doi octeţi.

c) tipul logic: este destinat reprezentării mulţimii valorilor logice fals şi adevărat.

d) tipul enumerare: este destinat reprezentării unei mulţimi finite de valori, fiecare valoare fiind desemnată printr-un nume simbolic. Se utilizează de obicei atunci când este necesară definirea unui tip de date prin indicarea explicită a elementelor sale. Elementele unui tip enumerare se reprezintă intern prin codificare cu ajutorul unor constante întregi.

e) tipul subdomeniu: este destinat reprezentării unui interval al unui tip ordonat liniar. Tipurile subdomeniu se utilizează de obicei pentru reprezentarea indicilor tablourilor.

f) tipul referinţă: este destinat reprezentării adreselor altor obiecte. Tipul referinţă se utilizează de obicei pentru implementarea unor obiecte abstracte complexe cum sunt listele şi arborii. Numeroase prelucrări se referă la relaţiile dintre obiectele prelucrate şi nu doar la valorile lor. În astfel de cazuri poate fi necesară reprezentarea explicită a acestor relaţii. Pentru aceasta anumite obiecte trebuie să se poată referi la alte obiecte. Este posibil chiar ca un acelaşi obiect să fie referit din două sau mai multe locuri. Întrucât copierea valorii obiectului în locul de unde a fost referit nu este o soluţie acceptabil din cauza consumului mare de memorie şi dificultăţilor de menţinere a consistenţei, rezultă că trebuie apelat la o altă tehnică şi anume de a defini în program obiecte care să poată referi alte obiecte. Pentru aceasta se folosesc tipul referinţă şi obiectele pointer. O referinţă la un obiect se implementează ca adresă a locaţiei de memorie, corespunzătoare obiectului respectiv.

INGINERIA PROGRAMĂRII

Şabloane de proiectare. Definitii, categorii, utilizare.
(Bădică, A., Ingineria programării, Note de curs, pag. 201-204)

Proiectarea orientată pe obiecte a software-ului presupune identificarea de obiecte, abstractizarea lor în clase de granularitate potrivită, definirea interfeţelor şi ierarhiilor de moştenire, stabilirea relaţiilor între aceste clase. Soluţia trebuie să rezolve problema şi să fie în acelaşi timp suficient de flexibilă pentru a rezista la noi cerinţe şi probleme ce pot apare în timp.

Există grupări de clase sau obiecte care se repetă în cele mai diferite sisteme. Acestea rezolvă probleme specifice, folosirea lor fac proiectele mai flexibile, mai elegante, reutilizabile. Un proiectant care stăpâneşte un set de asemenea şabloane le poate aplica imediat la noile proiecte fără a mai fi nevoit să le redescopere.

Şabloanele ce se pot refolosi pot fi general valabile sau specifice unui domeniu, de exemplu pentru probleme de concurenţă, sisteme distribuite, programare în timp real, etc. Şabloanele utilizate în sistemele OO pot fi clasificate într-o ierarhie după cum urmează:

- Idiomuri
- Mecanisme
- Cadre (frameworks).

Un **idiom** este legat de un anumit limbaj de programare şi reprezintă o convenţie general acceptată de utilizare a limbajului respectiv. Exemple tipice de idiomuri pot fi găsite în cadrul limbajelor C/C++. Returnarea unei valori întregi care să semnifice succesul sau eşecul unei funcţii este un idiom din C (adoptat şi de C++, pe langă generarea excepţiilor). Importanţa acestui idiom constă în faptul că el reprezintă un anumit stil acceptat de comunitatea utilizatorilor de C şi orice programator care citeşte o secvenţă C recunoaşte imediat această convenţie. Încălcarea acestui idiom are drept consecinţă producerea de cod greu de înţeles, chiar dacă este corect. Practic fiecare limbaj de programare îşi are propriile sale idiomuri. La fel şi o echipă de programatori îşi poate stabili un set de obiceiuri, în funcţie de experienţa şi cultura pe care le posedă. Se poate spune că un idiom este o formă de reutilizare pe scară mică.

Un **mecanism** este o structură în cadrul căruia obiectele colaborează în vederea obţinerii unui anumit comportament ce satisface o anumită cerinţă a problemei. Mecanismele reprezintă decizii de proiectare privind modul în care cooperează colecţiile de obiecte. Ele se mai numesc şi sabloane de proiectare (design patterns).

Majoritatea sistemelor OO includ mecanisme referitoare la:

- persistenţa obiectelor
- controlul stocării
- controlul proceselor
- transmisia/recepţia mesajelor
- distribuirea şi migrarea obiectelor
- conectarea în reţele (networking)
- tranzacţii
- evenimente
- modul de prezentare ("look & feel") al aplicaţiei.

Un **cadru** reprezintă o colecţie de clase care oferă un set de servicii pentru un domeniu particular. Cadrul exportă un număr de clase şi mecanisme pe care utilizatorii le pot adapta. Cadrele sunt forme de reutilizare pe scară largă. Cele mai răspândite tipuri de cadre sunt cele destinate creării de interfeţe grafice.

Un sablon reprezintă o soluţie comună a unei probleme într-un anumit context. Importanţa şabloanelor (standardelor) în construirea sistemelor complexe a fost de mult recunoscută în alte discipline. În cadrul comunităţii proiectanţilor de software OO (orientate pe obiecte) ideea de a aplica şabloane se pare că a fost inspirată de propunerea unui arhitect, Christopher Alexander, care a lansat iniţiativa folosirii unui limbaj bazat pe sabloane pentru proiectarea clădirilor şi a oraselor. Acesta afirma că: "Fiecare şablon descrie o problemă care apare mereu în domeniul nostru de activitate şi indică esenţa soluţiei acelei probleme într-un mod care permite utilizarea soluţiei de nenumărate ori în contexte diferite". Deşi în domeniul sistemelor OO soluţiile sunt exprimate în termeni de obiecte şi interfeţe (în loc de ziduri, uşi, grinzi etc), esenţa noţiunii de sablon este aceeaşi, adică de soluţie a unei probleme într-un context dat. Un şablon este descris de patru elemente:

- **Nume**: foloseşte pentru identificare; descrie sintetic problema rezolvată de şablon şi soluţia.

- **Problema**: descrie când se aplică şablonul; se descrie problema şi contextul.
- **Solutia**: descrie elementele care intră în rezolvare, relaţiile între ele, responsabilităţile lor şi colaborările între ele.
- **Consecinţe si compromisuri**: implicaţiile folosirii şablonului, costuri şi beneficii. Acestea pot privi impactul asupra flexibilităţii, extensibilităţii sau portabilităţii sistemului, după cum pot să se refere la aspecte ale implementării sau limbajului de programare utilizat. Compromisurile sunt de cele mai multe ori legate de spaţiu şi timp.

  Şabloanele sunt la diferite niveluri de abstractizare, au granularităţi diferite.

Criterii de clasificare:

- **Scop** - şabloanele pot fi creaţionale, structurale sau comportamentale.
  - Şabloanele **creaţionale** (creational patterns) privesc modul de creare al obiectelor.
  - Şabloanele **structurale** (structural patterns) se referă la compoziţia claselor sau al obiectelor.
  - Şabloanele **comportamentale** (behavioral patterns) caracterizează modul în care obiectele şi clasele interacţionează şi îşi distribuie responsabilităţile.
- **Domeniu de aplicare -** şabloanele se pot aplica obiectelor sau claselor.
  - **Şabloanele claselor** se referă la relaţii dintre clase, relaţii stabilite prin moştenire şi care sunt statice (fixate la compilare).
    - Şabloanele creaţionale ale claselor acoperă situaţiile în care o parte din procesul creării unui obiect cade în sarcina subclaselor.
    - Şabloanele structurale ale claselor descriu modul de utilizare a moştenirii în scopul compunerii claselor.
    - Şabloanele comportamentale ale claselor utilizează moştenirea pentru descrierea unor algoritmi şi fluxuri de control.
  - **Şabloanele obiectelor** se referă la relaţiile dintre obiecte, relaţii care au un caracter dinamic.
    - Şabloanele creaţionale ale obiectelor acoperă situaţiile în care o parte din procesul creării unui obiect cade în sarcina unui alt obiect.
    - Şabloanele structurale ale obiectelor descriu căile prin care se asamblează obiecte.
    - Şabloanele comportamentale ale obiectelor descriu modul în care un grup de obiecte cooperează pentru a îndeplini o sarcină ce nu ar putea fi efectuată de un singur obiect.

În tabelul de mai jos sunt incluse cele mai importante şabloane, clasificate după criteriile enumerate anterior:

Scop / Domeniu de aplicare	Creationale	Structurale	Comportamentale
Clasa	Factory Method	Adapter (clasa) Interface Marker Interface	Interpreter Template Method
Obiect	Immutable Abstract Factory Builder Prototype Singleton	Delegation Adapter (obiect) Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

Şabloanele de proiectare rezolvă multe din problemele cu care se confruntă proiectanţii. Câteva din aceste probleme sunt urmatoarele:

- Găsirea obiectelor adecvate

Un obiect, care intră în alcătuirea programelor OO, împachetează atât **date**, cât şi **metode (operaţii)** ce operează asupra datelor. Obiectul execută o operaţie când primeşte o **cerere (mesaj)** de la un **client**. Partea dificilă în proiectarea unui sistem OO este descompunerea sistemului în obiecte. Aceasta deoarece procesul este influenţat de mai mulţi factori care acţionează adesea în mod contradictoriu: încapsularea, granularitatea, dependenţele, flexibilitatea, performanţele, evoluţia, gradul de reutilizare etc. Şabloanele ne pot ajuta în identificarea unor abstracţiuni mai puţin evidente şi a obiectelor care le pot reprezenta. De exemplu, obiectele care reprezintă procese sau algoritmi nu apar în natură, dar ele nu pot lipsi dintr-un proiect. Şablonul Strategy descrie modul de implementare a unor familii interschimbabile de algoritmi. Şablonul State reprezintă fiecare stare a unei entităţi sub forma unui obiect. Asemenea obiecte sunt rareori descoperite în timpul analizei sau chiar a stadiului incipient al proiectării.

- Determinarea granularităţii obiectelor

Obiectele ce compun un sistem pot varia enorm ca mărime şi ca număr. Ele pot reprezenta practic orice începând de la componente hardware până la aplicaţii întregi. Cum decidem ce ar trebui sa fie un obiect? Există şabloane care acoperă şi acest aspect. Astfel, şablonul Facade descrie modul în care subsisteme complete pot fi reprezentate ca obiecte, şablonul Flyweight arată cum se poate gestiona un număr uriaş de obiecte la nivelurile cele mai fine de granularitate. Alte şabloane descriu căile prin care un obiect poate fi descompus în obiecte mai mici. Abstract Factory şi Builder reprezintă obiecte a căror unică responsabilitate este crearea de alte obiecte. Visitor şi Command reprezintă obiecte a căror unică responsabilitate este implementarea unui mesaj către alt obiect sau grup de obiecte.

- Specificarea interfeţelor obiectelor

Pentru fiecare operaţie declarată într-un obiect se precizează **numele**, obiectele pe care le ia ca **parametrii** şi **valoarea returnată**; aceste elemente formează **semnătura** operaţiei. Mulţimea tuturor semnăturilor corespunzătoare operaţiilor dintr-un obiect reprezintă **interfaţa** obiectului. Interfaţa unui obiect descrie complet setul mesajelor care pot fi trimise spre obiectul respectiv. Interfeţele sunt lucruri fundamentale în sistemele OO. Obiectele sunt cunoscute doar prin intermediul interfetelor lor. O interfaţă nu dă nici un detaliu relativ la implementarea unui obiect, iar obiecte distincte pot implementa în mod diferit o aceeasi cerere. Obiecte diferite care pot recepţiona cereri identice pot avea implementări diferite ale operaţiilor ce vor satisface cererile respective. În acest context, şabloanele de proiectare ne ajută la: definirea interfeţelor; identificarea elementelor care NU trebuie să apară într-o interfaţă. Astfel, şablonul Memento descrie modul de încapsulare şi salvare a stării interne a unui obiect pentru ca starea respectivă să poată fi restaurată ulterior. Şabloanele specifică de asemenea şi relaţii între interfeţe.

- Specificarea implementării obiectelor

Implementarea unui obiect este definită prin intermediul **clasei** obiectului. Clasa unui obiect specifică datele interne ale obiectului şi definiţiile operaţiilor pe care acesta le poate executa. Obiectele sunt create prin **instanţierea** unei clase; se mai spune că un obiect este o **instanţă** a unei clase. Procesul de instanţiere a unei clase presupune alocarea de memorie pentru datele interne ale obiectului respectiv şi asocierea operaţiilor cu aceste date. Pe baza unor clase existente se pot defini noi clase, folosind **moştenirea claselor**. O **subclasă** moşteneşte de la una sau mai multe **clase părinte** (superclase) toate datele şi operaţiile definite în acestea din urmă. O **clasă abstractă** are drept scop principal definirea unei interfeţe comune pentru subclasele sale. Implementarea operaţiilor unei clase abstracte este pasată parţial sau în întregime subclaselor sale. De aceea, o clasă abstractă nu poate fi instanţiată. Operaţiile declarate într-o clasă abstractă, dar neimplementate se numesc **operaţii abstracte**. Clasele care nu sunt abstracte se numesc **clase concrete**.

Este important să inţelegem diferenţa între clasa unui obiect şi tipul obiectului. Clasa defineşte starea internă a obiectului şi implementarea operaţiilor lui. Tipul obiectului se

referă doar la interfaţa obiectului - setul cererilor la care obiectul poate răspunde. Un obiect poate avea mai multe tipuri, iar obiecte de clase diferite pot avea acelaşi tip. Este de asemenea important să înţelegem diferenţa dintre moştenirea de clasă şi moştenirea de interfaţă (subtipizare). Moştenirea de clasă presupune că implementarea unui obiect este definită în termenii implementării altui obiect. Moştenirea de interfaţă (subtyping) este un mecanism prin care un obiect poate fi utilizat în locul altuia. Multe dintre şabloanele de proiectare se bazează pe această distincţie.

- Mecanisme ale reutilizării

Problema este cum obţinem un software flexibil şi reutilizabil folosind concepte ca obiect, clasă, interfaţă, moştenire. Şabloanele constituie un răspuns.

Ingineria programării – definiţie, principii, etape
(Bădică, A., Ingineria programării, Note de curs, pag. 2-5)

Un produs program, ca obiect utilitar distinct şi identificabil (de exemplu, un interpretor, un compilator, un sistem de operare, un joc, un sistem informatic pentru gestiune), este rezultatul unui proces creativ uman, ca multe alte produse rezultate din activitatea umană. Există însă câteva caracteristici ale programelor care le deosebesc de alte obiecte şi anume:

- un program este un element logic şi nu fizic, acesta fiind în esenţă dezvoltat, nu fabricat în sensul clasic, fapt ce influenţează distribuirea costurilor pe fazele procesului de producţie;
- întreţinerea programelor implică un grad mai mare de complexitate, deoarece nu există piese de rezervă pentru programe. Fiecare defect indică o eroare în proiectare sau în procesul de implementare.

IP urmăreşte stabilirea şi utilizarea de principii inginereşti robuste pentru obţinerea cu costuri cât mai mici a unor produse software fiabile şi care să ruleze eficient pe calculatoarele existente.

Această definiţie nu s-a modificat ulterior prea mult. Spre exemplu, definiţia dată de Glosarul Standard de Termeni din Ingineria Programării al IEEE (`83) este următoarea:

IP reprezintă abordarea sistematică în dezvoltarea, operarea, întreţinerea şi retragerea programelor.

Caracteristicile domeniului ingineriei programării sunt următoarele:

i) IP se referă la dezvoltarea de programe mari.

Programarea-în-mic = dezvoltarea unui program de către o persoană, într-o perioadă relativ scurtă de timp (de regulă mai mică de 6 luni). Apelează de regulă la tehnicile convenţionale de programare: programare modulară, programare structurată, diagrame de structură, etc.) *Programarea-în-mare* = dezvoltarea unui program de către un grup de persoane, într-o perioadă relativ mare de timp. (de regulă mai mare de 6 luni).

ii) un obiectiv principal al IP este stăpânirea complexităţii. Acest lucru se realizează prin aplicarea teoremei fundamentale a ingineriei programării. Să presupunem că M(P) = măsura complexităţii problemei P şi C(P) = costul rezolvării lui P. În mod evident, dacă M(P) > M(Q) atunci C(P) > C(Q).

Pentru două probleme P şi Q, C(P+Q) > C(P) + C(Q).

Este clar că deoarece M(P+Q) include şi măsura interacţiunii dintre P şi Q, avem M(P+Q) > M(P) + M(Q) de unde rezultă enunţul teoremei.

iii) Programarea-în-mare presupune cooperarea între oameni în general şi între programatori în special. Se au în vedere:

- distribuţia muncii
- metodele de comunicare
- responsabilităţile

Succesul unui proiect depinde în mare măsură de disciplina participanţilor (manageri, analişti, programatori).

iv) IP are în vedere evoluţia programelor. Orice program modelează o parte a realităţii şi, deoarece realitatea evoluează, programele evoluează şi ele, deci trebuie alocate costuri pentru evoluţie chiar după livrarea programului
v) IP are în vedere minimizarea costurilor dezvoltării şi întreţinerii programelor. Costurile se referă la consumurile de:

- timp
- efort
- resurse umane
- resurse financiare

vi) IP are în vedere ca programele să corespundă cerinţelor reale ale utilizatorilor, adică să fie:

- funcţionale
- fiabile
- prietenoase
- uşor de instalat

Concluzie: activitatea de programare (codificare) este inclusă în IP şi incluziunea este strictă.

Un obiectiv principal al IP este obţinerea de produse program (PP). Un PP este rezultatul unui proces creativ uman, în care este înglobată o mulţime eterogenă de elemente: imaginaţie şi rigoare, cunoştinţe şi experienţă, inteligenţă şi tehnică de calcul, care împreună cu un calculator (parte hardware) efectuează toate prelucrările dorite de utilizator.

Dificultatea elaborării PP rezidă în caracterul logic, abstract al acestora. Astfel, spre deosebire de alte obiecte produse de oameni, un PP nu este fabricat, ci este dezvoltat.

Produsele program pot fi clasificate în:
i) produse generice (PPG): sunt PP ale unei companii de dezvoltare care sunt vândute pe piaţa liberă oricărui client interesat
ii) produse personalizate (PPP): sunt produse program destinate unui client particular. Ele sunt dezvoltate special pentru acel client de către un contractor

Aceste categorii de PP diferă semnificativ în specificaţii:
i) specificaţiile PPG sunt produse intern de departamentul de marketing al companiei respective şi ele reflectă ceea ce cred membrii acestui departament că s-ar vinde (sunt flexibile şi neprescriptive)
ii) specificaţiile PPP sunt baza contractului între client şi contractor. Sunt foarte detaliate şi se negociază foarte atent.

Atributele PP se referă la caracteristicile unui PP odată instalat şi pus în exploatare. Ele nu se referă la serviciile furnizate de PP, ci la utilitatea şi calitatea acestora.

Atributele esenţiale ale unui PP sunt:
i) mentenabilitate: reflectă capacitatea unui PP de a evolua pentru a satisface cerinţele în continuă schimbare ale clienţilor
ii) credibilitate: reflectă nivelul de încrederea într-un PP. Include caracteristici ca: fiabilitate, siguranţă, securitate. Un PP credibil nu trebuie să cauzeze pagube fizice sau economice în cazul apariţiei unor defecte în funcţionare
iii) eficienţă: reflectă măsura în care un PP foloseşte resursele sistemului (memorie şi timp procesor)
iv) utilizabilitate: reflectă măsura în care un PP este uşor de folosit şi bine documentat

Există două abordări principal şi principial diferite în dezvoltarea programelor. Uneori ele sunt numite paradigme de dezvoltare. Acestea sunt:

- paradigma structurată, predominantă în anii `80
- paradigma orientată pe obiect, predominantă în anii `90

Paradigma structurată se bazează pe abstractizarea noţiunilor de procedură, funcţie şi structură de bloc din limbaje de programare cum sunt Algol şi Pascal. Uneori această paradigmă se numeşte funcţională deoarece sugerează rezolvarea unei probleme prin descompunere funcţională. PS pleacă de la comportamentul dorit al PP pe care îl descompune utilizând un procedeu de sus în jos - „top-down”, până la obţinerea unor elemente ce pot fi implementate direct. Dezavantajul

este că rezultatele obţinute prin aplicarea metodelor acestei paradigme sunt puternic dependente de problemă şi au astfel un grad redus de reutilizabilitate, chiar şi la probleme asemănătoare.
Paradigma orientată pe obiect inversează metodologiile structurate, accentul fiind pus pe identificarea obiectelor din domeniul problemei şi nu pe descompunerea funcţională a problemei. Avantajul este că la orice nivel în cadrul procesului de dezvoltare pot fi identificate elemente candidate la reutilizare.

Elaborarea unui PP, cunoscută şi ca *ciclul de viaţă al unui PP* (CVPP), reprezintă mulţimea tuturor activităţilor împreună cu rezultatele asociate lor, care au ca efect final obţinerea unui PP. Aceste activităţi includ:

i) *analiza cerinţelor*. Are ca obiectiv obţinerea unei descrieri cât mai complete a problemei de rezolvat şi a cerinţelor impuse de şi către mediul în care va funcţiona PP. Rezultatul acestei activiţăţi este specificarea cerinţelor.

ii) *proiectarea.* Are ca obiectiv obţinerea unui model al PP care, dacă este transpus într-un limbaj de programare, rezolvă problema dată.

Observaţie: deşi activităţile i) şi ii) sunt privite deseori ca o introducere plictisitoare în programare, codificarea fiind luată drept „adevărata problemă", această atitudine are un efect negativ asupra calităţii unui PP.

Proiectarea se face la un nivel abstract, independent de detaliile de implementare, iar rezultatul său este concretizat într-o specificaţie tehnică sau proiect.

iii) *implementarea*. Are ca obiectiv transpunerea proiectului într-un limbaj de progrmare. Accentul este pus în primul rând pe obţinerea unui program bine documentat, fiabil, lizibil, flexibil şi corect şi mai puţin pe aspecte secundare ca obţinerea unui program foarte eficient sau aplicarea unor trucuri de programare. Rezultatul acestei activităţi este un program executabil. Activităţile ii) şi iii) se grupează deseori sub denumirea generică de dezvoltare.

iv) *testarea*. Este greşit să se considere testarea ca etapă succesoare implementării, deoarece acest lucru ar insemna să ne preocupăm de ea numai după ce programul a fost implementat. Corect este ca testarea să fie efectuată după fiecare activitate în parte, sub forma verificării şi validării.

Verificarea = stabilirea corectitudinii tranziţiei din etapa i în etapa i+1.
Validarea = stabilirea faptului că nu ne-am îndepărtat de cerinţele stabilite iniţial.

v) *întreţinerea*. Are ca obiective îndepărtarea erorilor raportate în faza de exploatare a PP şi evoluţia PP pentru a satisface eventuale cerinţe noi.

Activităţile de întreţinere se clasifică în:

- *întreţinere corectivă* = înlăturarea erorilor
- *întreţinere adaptivă* = adaptarea PP la schimbările mediului în care operează: hardware nou, o nouă versiune a SO sau SGBD utilizat, etc.
- *întreţinere perfectivă* = adaptarea PP la noile cerinţe ale utilizatorilor
- *întreţinere preventivă* = actualizarea documentaţiei, îmbunătăţirea structurii modulare, adăugarea de comentarii, etc. Are ca obiectiv îmbunătăţirea mentenabilităţii viitoare.

Maniera în care procesul de elaborare al unui PP se descompune în activităţi şi subactivităţi cât şi modul de corelare în timp al acestora definesc un model al ciclului de viaţă al unui PP.

CVPP are următoarele atribute importante:

- *inteligibilitate*: indică măsura în care CVPP este explicit definit şi uşor de înţeles.
- *vizibilitate*: indică măsura în care rezultatele fiecărei activităţi a CVPP sunt suficient de clare astfel încât progresul să fie vizibil din exterior.
- *suportabilitate*: indică măsura în care activităţile CVPP pot fi automatizate (suportate de unelte CASE).
- *acceptabilitate*: indică măsura în care CVPP este acceptabil şi utilizabil de către inginerii responsabili de obţinerea PP.
- *fiabilitate*: indică măsura în care erorile în cadrul CVPP să poată fi evitate su înlăturate înainte de a se transforma în erori ale PP.

- *robusteţe*: indică măsura în care proiectul poate continua în eventualitatea apariţiei unor situaţii neaşteptate în cadrul CVPP.
- *mentenabilitate*: indică măsura în care CVPP poate evolua la apariţia unor schimbări în cadrul companiei sau la depistarea unor posibile îmbunătăţiri.
- *rapiditate*: indică cât de repede pote fi furnizat PP prin aplicarea CVPP.

PROGRAMAREÎN INTERNET

Programare pe parte de client
(Bădică, A., Programare în Internet, Note de curs, pag. 32-35)

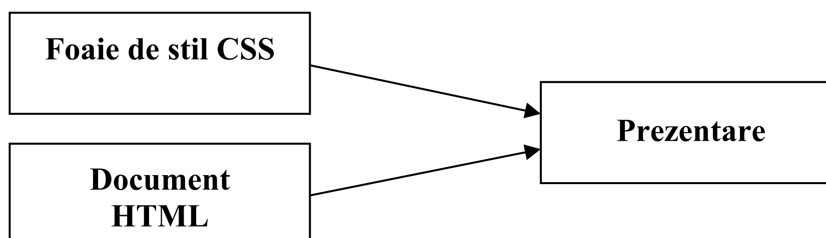
Termenul de programare pe partea de client (engl.client side programming) se referă la executarea de programe la client în scopul creşterii gradului de interactivitate a paginilor WWW. Programele destinate a fi executate la client se pot transmite de la server către client fie în format sursa, fie în format obiect.

Termenul Dynamic HTML – DHTML nu se referă la o anumită versiune sau facilitate a HTML. El desemnează acele facilităţi din diversele variante sau extensii ale HTML care ajută la crearea de conţinut dinamic.

Cele mai populare elemente ale DHTML sunt:

- Foile de stil (engl.Cascading Style Sheets - CSS).
- Scripting-ul la client (elementul SCRIPT). Un exemplu este JavaScript. Se referă la încorporarea în cadrul unei pagini a unor programe în format sursă. Ele vor fi executate la client.
- Obiectele (elementul OBJECT). Se referă la încorporarea in cadrul unei pagini a unor programe în format obiect. Ele vor fi executate la client.
- Modelul obiectelor document (engl.Document Object Model – DOM). Acesta este liantul dintre elementele anterioare şi limbajul de marcare HTML sau XML.
- Eventuale mecanisme particulare specifice programului navigator.

Se recomandă separarea prezentării (margini, culori, fonturi) de conţinutul şi structura documentului (antet, pagini, paragrafe, titluri, secţiuni). Pentru aceasta se pot folosi foi de stil (engl.Cascading Style Sheets – CSS).



Prezentarea unui document este în general determinată de mai mulţi factori:

- Intenţiile proiectantului paginii
- Reguli generale pe care trebuie să le urmeze proiectantul
- Preferinţele utilizatorului
- Limitările cauzate de terminalul pe care se vizualizează prezentarea

Din acest motiv apare necesitatea utilizării mai multor foi de stil – cascadarea foilor de stil (CSS). CSS specifică stilul de prezentare al unui document HTML. Există trei modalităţi de a mixa CSS si HTML: i) în cadrul unui element HTML; ii) în antetul unui document HTML; iii) într-un fişier separat.

O foaie de stil este constituită dintr-o multime de reguli care se aplică unui document HTML în scopul generării unei prezentări a documentului.

O regulă de stil conţine o parte de condiţii (stângă) numită şi selector şi o parte de acţiuni (dreaptă) numită şi declaraţie. Exemplu: P {color:green}, unde:

- P este un selector