

NOȚIUNI DE BAZĂ PRIVIND INSTRUMENTELE SOFTWARE

Sumar:

- Istoricul limbajelor de programare
- Etapele elaborării programelor de aplicații și rolul economiștilor
- Instrumente pentru dezvoltarea aplicațiilor
- Instrumente software pentru economiști
- Tendințe în instrumentele software

1.1 ISTORICUL LIMBAJELOR DE PROGRAMARE

Astăzi, aproape orice activitate dintr-o organizație se desfășoară cu ajutorul calculatorului. Nevoia de programe de aplicații s-a diversificat în permanență, iar odată cu ele și instrumentele software. Prin **instrument software** se înțelege un program folosit de informaticieni sau utilizatori pentru dezvoltarea programelor de aplicații.

Istoria instrumentelor software a început odată cu primele calculatoare, la începutul anilor '50. De atunci și până în anii '80 singura modalitate de dezvoltare a programelor de aplicații era reprezentată de limbajele de programare. Astăzi avem la dispoziție o multitudine de alte instrumente software. În afara limbajelor de programare pot fi enumerate sistemele de gestiune a bazelor de date, programe de calcul tabelar, produsele CASE, programele pentru gestiunea informațiilor personale și multe altele. Multe dintre acestea se adresează nespecialiștilor în programarea calculatoarelor.

Pentru a înțelege principalele evoluții și tendințe ale instrumentelor software, vom face cu o scurtă prezentare a istoricului limbajelor de programare.

Limbajele de programare constituie rezultatul unei evoluții, care a fost determinată atât de performanțele calculatoarelor electronice, cât și de progresele din teoria și practica programării. Se disting cinci categorii de limbaje de programare: *limbaje cod mașină*, *limbaje de asamblare*, *limbaje de nivel înalt*, *limbaje din generația a patra*, numite și *limbaje de nivel foarte înalt* și *limbaje naturale*.

Fiecare procesor are propriul set de instrucțiuni, care formează așa numitul **limbaj mașină**. În primii ani ai dezvoltării programelor, aceste limbaje constituiau singura variantă la îndemână. Toate instrucțiunile din program trebuiau să fie redactate în cod binar, adică sub forma unor șiruri formate din 0 și 1. De exemplu, o instrucțiune avea forma 0110 1010 0110 1011 și poate semnifica operațiunea de adunare. Astfel, programele se concretizau în secvențe de cifre binare pentru adrese, date și operații, programatorul cunoscând perfect operațiunile interne din unitatea centrală a calculatorului. O simplă funcție de prelucrare necesita lungi serii de instrucțiuni pentru a fi realizată.

Dezavantajele acestor limbaje de programare sunt astăzi evidente: dificultatea scrierii și, mai ales, a înțelegerii programelor; lipsa portabilității programelor, adică imposibilitatea executării lor pe alte tipuri de calculatoare decât cel pentru care au fost elaborate.

Limbajele de asamblare au permis reducerea dificultăților specifice programării în limbaj mașină, prin folosirea unor abrevieri alfabetic (mnemonice) în locul șirurilor de 0 și 1. De exemplu, instrucțiunea „ADD R1, R2, R4” semnifică adunarea valorilor conținute în registrele de memorie R1 și R2, iar rezultatul este scris în registrul R4. După cum se poate observa, codurile de operații și pozițiile din memorie sunt reprezentate prin simboluri, motiv pentru care aceste limbaje se mai numesc și *simbolice*. Ele simplifică mult programarea, deoarece abrevierile alfabetic sunt mai ușor de memorat decât instrucțiunile scrise în binar, oferind mai multă comoditate în scrierea sau citirea și înțelegerea programelor.

Odată scrise, programele în limbajele de ansamblare trebuie traduse în limbajul mașină, pentru a putea fi executate pe calculator. Această operațiune este realizată prin intermediul unor programe speciale numite *ansambloare*. Ele preiau *programele sursă* (cele scrise în limbajul de ansamblare) și le convertesc în *programe obiect executabile* (scrise în limbajul mașină). Operațiunea de ansamblare este necesară deoarece un calculator știe să interpreteze și să execute doar programele scrise în cod mașină.

Principalele limbaje din această categorie sunt: ASSIRIS, ASSEMBLER, MAGIRIS.

Dincolo de avantajele sale, limbajele de asamblare rămân orientate-mașină, deoarece comenzile sale corespund instrucțiunilor în limbaj mașină, conform tipului de calculator utilizat. O singură instrucțiune în limbaj de asamblare corespunde unei singure instrucțiuni în limbaj mașină și, ca atare, este nevoie de același număr de instrucțiuni pentru scrierea programelor în ambele cazuri. Ca și programele în cod mașină, programele scrise în limbaj de asamblare nu pot fi portate de pe un tip de calculator pe altul. Totuși, ele sunt încă utilizate astăzi, mai ales de către programatorii care dezvoltă software de sistem, pentru că ele permit utilizarea mai eficientă a resurselor calculatorului și sunt mai rapide la execuție.

În practica programării calculatoarelor, limbajele mașină și cele de asamblare sunt considerate *limbaje de nivel redus*.

O dată cu lansarea limbajului FORTRAN (1954) s-a trecut la o altă generație de **limbaje**, numite **de nivel înalt** sau **evolute**. Alte limbaje din această categorie sunt: COBOL, BASIC, C și C++, ADA, PROLOG.

Limbajele de nivel înalt au fost dezvoltate pentru a crește productivitatea muncii programatorilor. Ele sunt mai ușor de asimilat și de înțeles decât un limbaj de asamblare, deoarece regulile, normele și sintaxa sunt mai suple, ceea ce reduce riscul erorilor. Instrucțiunile în limbajele evolute, numite *enunțuri*, se aseamănă mult cu limbajul uman sau cu convențiile de scriere în matematică. De fapt, fiecare enunț constituie o macroinstrucțiune, adică ea realizează operațiuni specifice mai multor instrucțiuni în limbajul mașină. Iată câteva exemple de comenzi întâlnite în limbajele de nivel înalt: COMPUTE, DISPLAY, SUBSTRACT.

Pentru a fi executat, programul sursă, scris într-un limbaj de nivel înalt, trebuie convertit în program obiect, scris în limbaj mașină. Această operațiune poate fi realizată prin intermediul unor programe speciale, numite *interpretoare* sau *compilatoare*. Interpretoarele tratează programul sursă la nivel de linie de instrucțiuni, fiecare linie fiind executată imediat ce este convertită în cod obiect. În schimb, compilatoarele realizează conversia întregului program sursă în program obiect și după aceea el poate fi executat. Datorită manierei diferite de lucru, programele interpretate au o viteză de execuție mai mică decât cele compilate. După compilare mai trebuie urmată o etapă înainte de a fi executat programul. Este vorba despre *editarea de legături*, în care programul obiect este completat cu module preluate din bibliotecile sistemului de operare.

Principalele avantaje ale limbajelor de nivel înalt sunt:

- *Creșterea productivității muncii de programare*. Instrucțiunile din program sunt orientate pe activitate și nu pe calculatorul pe care se va executa programul, ceea ce ușurează munca programatorilor. Se mai spune că aceste

limbaje sunt orientate pe problemă. De exemplu, limbajul COBOL a fost conceput pentru dezvoltarea aplicațiilor economice, iar FORTRAN pentru scopuri științifice sau domeniul matematicii.

- *Programele scrise în limbajele de nivel înalt sunt portabile.* Prin utilizarea macroinstrucțiunilor în scrierea programelor sursă și traducerea lor în limbaj mașină, cu ajutorul interpretoarelor sau a compilatoarelor, programele vor putea fi executate pe diferite tipuri de calculatoare. Pentru a fi posibil acest lucru, fiecare tip de calculator va avea propriul interpretor/ compilator pentru fiecare limbaj de programare.
- *Posibilitatea scrierii de programe de către persoane mai puțin calificate decât în cazul limbajelor de asamblare.* Apropierea macroinstrucțiunilor de limbajul uman și orientarea lor pe problema de rezolvat le fac mai ușor de învățat.

În schimb, limbajele evaluate sunt mai puțin eficiente decât limbajele de asamblare, din punctul de vedere al utilizării resurselor calculatorului, și necesită mai mult timp de execuție. Aceste inconveniente n-au putut determina programatorii să nu utilizeze pe scară largă limbajele de nivel înalt în dezvoltarea aplicațiilor pentru companii.

Limbajele din generația a patra (cunoscute și sub acronimul **4LG**) descriu o mare varietate de limbaje care sunt mai neprocedurale și mai conversaționale decât cele de nivel înalt. Ele sunt extrem de cunoscute și utilizate astăzi, dar, în același timp, dificil de definit, chiar și încadrarea lor în categoria „limbaje” fiind discutabilă. Ele pot fi considerate ca limbaje sofisticate care permit creșterea productivității muncii programatorilor, prin facilitățile de dezvoltare a aplicațiilor pe care le oferă, sau ca limbaje mai directe și mai simple, care permit utilizatorilor să-și dezvolte propriile aplicații informatice.

Totuși, pot fi identificate câteva caracteristici comune majorității limbajelor 4GL¹:

- sunt *centrate în jurul bazelor de date*, în mod deosebit a celor relaționale, oferind facilități pentru stocarea și interogarea datelor din bază;
- sunt limbaje *declarative*, numite și *neprocedurale*, ceea ce înseamnă că accentul este pus pe ceea ce trebuie să facă aplicația și nu cum trebuie să facă. Acum este posibilă dezvoltarea unor mici aplicații informatice fără a mai fi necesară scrierea de programe. Utilizatorii se rezumă la a indica rezultatele vizate, de o manieră declarativă, astfel încât calculatorul să poată determina secvența de program necesară pentru obținerea acestor rezultate. Din acest motiv, se spune că limbajele din această generație au simplificat mult procesul de programare.
- pun la dispoziție o *interfață grafică prietenoasă*, bazată pe dialogul interactiv, cu care utilizatorii pot interacționa pentru a-și dezvolta propriile aplicații. Această interfață este referită prin acronimul GUI (Graphical User Interface – Interfață grafică utilizator);

¹ prelucrare după Curtis, G., Cobham, D. – *Business Information Systems. Analysis, Design and Practice*, fourth edition, Prentice Hall, 2002, p. 122

- *includ limbaje de programare de nivel înalt* pentru scrierea de proceduri prin care să se acopere anumite funcțiuni ale aplicațiilor care nu pot fi rezolvate prin instrumentele 4GL;

Apariția limbajelor 4GL a reprezentat răspunsul la neajunsurile limbajelor de nivel înalt. Deși acestea au permis o sporire considerabilă a productivității muncii programatorilor, în comparație cu limbajele de ansamblare, ea nu a fost suficientă în condițiile extinderii rapide a utilizării microcalculatoarelor din anii '80, a creșterii permanente a cerințelor utilizatorilor și a complexității aplicațiilor informatice. De fapt, prin introducerea limbajelor 4GL s-a urmărit rezolvarea a trei probleme: reducerea costurilor mari cu dezvoltarea programelor, nevoia de obținere a programelor în timp cât mai scurt și creșterea calității programelor, mai ales în ce privește includerea cerințelor funcționale ale utilizatorilor.

Prin urmare, aceste limbaje prezintă următoarele avantaje:

- posibilitatea dezvoltării de noi aplicații mai ieftine și mai rapid;
- ușurința întreținerii aplicațiilor;
- posibilitatea utilizatorilor de a-și dezvolta propriile aplicații. Odată cu apariția acestor limbaje s-a dezvoltat un nou concept – **utilizatorul final informatizat (end user computing)**, care face referire la faptul că utilizatorii cu mai puține cunoștințe informatice își pot crea și întreține singuri propriile aplicații.

În evoluția limbajelor de programare de până acum, apariția unei generații noi a determinat renunțarea, în bună măsură, la limbajele din generația anterioară, dar nu și dispariția lor. Acum, ne putem întreba dacă apariția și dezvoltarea limbajelor 4GL vor avea același efect asupra limbajelor de nivel înalt. Răspunsul ferm este **nu**, atât pentru prezent, cât și pentru viitorul imediat. Explicațiile sunt următoarele:

- aplicațiile dezvoltate în 4GL sunt mai puțin eficiente decât limbajele de nivel înalt, din punctul de vedere al utilizării resurselor calculatorului, aspect care devine mai evident cu cât crește complexitatea aplicațiilor sau volumul datelor de prelucrat. Viteza de execuție a programelor generate automat de aceste limbaje, pe baza specificațiilor utilizatorilor, este mai mică, fapt care se răsfrânge în timpi de răspuns mai mari, dar și pretenții sporite în ce privește puterea de calcul. Unii chiar se întreabă dacă reducerea costurilor și a timpului de dezvoltare a aplicațiilor nu sunt depășite de costurile suplimentare cu achiziția echipamentelor necesare pentru obținerea de performanțe echivalente limbajelor din a treia generație.
- aplicațiile complexe solicitate astăzi de firme nu pot fi dezvoltate integral cu limbajele 4GL. Facilitățile oferite sunt încă limitate, ceea ce impune apelarea la limbajele de generația a treia, mai ales în cazul aplicațiilor pentru firmele medii și mari.
- utilizarea limbajelor 4GL de către utilizatori poate conduce la lipsa standardizării în dezvoltarea sistemelor informatice la nivelul organizației, ceea ce poate crea deficiențe în desfășurarea activității angajaților.

Cel mai tipic exemplu de limbaj 4GL este SQL. El îndeplinește ambele caracteristici ale acestei generații de limbaje: este utilizat de profesioniști, în vederea creșterii productivității muncii lor, dar și de utilizatorii finali, în scopul dezvoltării propriilor aplicații. SQL este orientat spre dezvoltarea aplicațiilor cu baze de date relaționale, fiind inclus de marea majoritatea a SGBD-urilor, precum ACCESS, Visual FoxPro, Oracle. El oferă facilități pentru crearea, actualizarea și interogarea bazelor de date

În categoria limbajelor 4GL sunt incluse și SGBD-ul ACCESS, programele de calcul tabelar EXCEL și LOTUS 1-2-3, limbajele de interogare a bazelor de date QBE și RPG IV, generatorul de rapoarte Crystal Reports.

Ultimul pas în evoluția limbajelor de programare îl reprezintă **limbajele naturale**, referite uneori și ca **limbaje de programare de generația a cincea**². Ele își propun utilizarea limbajului uman în programarea calculatoarelor, cele mai cunoscute exemple fiind INTELLECT și ELF. Ele sunt adesea folosite ca interfețe pentru instrumentele 4GL.

Principalul lor neajuns este legat de traducerea programelor din limbajul natural în limbaj mașină, o operațiune extrem de complexă și costisitoare din punctul de vedere al resurselor hardware solicitate. Ele sunt integrate cu aplicațiile de inteligență artificială, incluse și ele în această generație. Un astfel de exemplu îl reprezintă LISP.

Deși limbajele naturale sunt încă în faza de început a dezvoltării lor, ele reflectă foarte bine caracteristica principală a evoluției limbajelor de programare – apropierea de limbajul uman. Această caracteristică, precum și altele sunt evidențiate figura 1.1.



Generatia					
	I	II	III	IV	V
Caracteristici	Limbaj masina	Limbaj de ansamblare	Limbaj de nivel înalt	Limbaj neprocedural	Limbaj natural
Apropierea de limbajul uman	 <div>Progres</div> 				
Productivitate	<div>Progres</div>				
Eficienta utilizarii resurselor	<div>Diminuare</div>				
Portabilitate	Nu	Nu	Da	Da	Da
Utilizarea de macroinstrucțiuni	Nu	Nu	Da	Da	Da
Utilizarea simbolurilor	Nu	Da	Da	Da	Da

Figura 1.1 Caracteristicile evoluției limbajelor de programare

Sursa: adaptare după Turban, E., McLean, E., Wetherbe, J. – *Information Technology for Management*, John Wiley&Sons, 2001, p. 735

² unii autori includ limbajele naturale tot în categoria 4GL. Vezi O'Brien, J.A. – *Introduction ti Information Systems. Essentials for the Internetworked E-Business Enterprise*, McGraw-Hill, 2001, p.158

1.2 ETAPELE ELABORĂRII PROGRAMELOR DE APLICAȚII ȘI ROLUL ECONOMIȘTILOR

Dezvoltarea programelor de aplicații (sau a sistemelor informaționale) nu se limitează la scrierea programelor, ci presupune parcurgerea unui proces mai amplu, format din mai multe etape. Ansamblul tuturor acestor etape formează **ciclul de viață al programelor de aplicații**. Astăzi există numeroase metodologii de dezvoltare a programelor, ce diferă între ele prin numărul etapelor, conținutul și organizarea lor. În acest paragraf dorim să descriem succint etapele comune majorității metodologiilor și să punem în evidență rolul economiștilor de-a lungul procesului de dezvoltare.

Procesul de dezvoltare începe cu **etapa de microanaliză**. Această etapă are drept țință definirea clară a problemei sau oportunității ivite, precum și a scopului urmărit, în sensul că trebuie să se înțeleagă ce se întâmplă la nivelul firmei, ce obiective are firma și cum noua aplicație poate conduce la atingerea lor. Este important să se identifice exact care sunt avantajele pe care le aduce noua aplicație, ce activități economice sprijină și ce probleme pot fi rezolvate cu ajutorul ei. Printre motivele de inițiere a dezvoltării unei noi aplicații putem enumera: modernizarea sistemului informațional în vederea valorificării avantajelor oferite de noile tehnologii informaționale; modificarea legislației; ineficiența unor activități derulate în cadrul firmei, datorată slabei integrări a sistemului informațional; aplicațiile existente nu mai fac față cerințelor, datorită extinderii activității firmei; sprijinirea realizării obiectivelor strategice.

La finalul acestei etape se elaborează un plan de dezvoltare, care este supus spre analiză conducerii. Dacă se obține aprobarea, proiectul poate fi inițiat, în sensul că vor fi alocate fondurile și resursele necesare și se va organiza echipa de dezvoltare.

Cel mai adesea, economiștii sunt cei care inițiază dezvoltarea de noi aplicații. Chiar dacă inițiativa aparține altor categorii de personal, rolul economiștilor rămâne important, deoarece ei trebuie să elaboreze planul de dezvoltare, în care se face o estimare a cheltuielilor și beneficiilor potențiale ale aplicației propuse.

După inițierea proiectului de dezvoltare a noului program de aplicație, se merge mai departe cu **etapa de analiză**, prin care se urmărește identificarea și descrierea cerințelor funcționale și informaționale. La sfârșitul ei trebuie să fie foarte clar ce funcții va realiza noua aplicație. Astfel, în această etapă sunt urmărite trei obiective principale:

- descrierea sistemului informațional existent, fiind analizate documentele primare, fluxurile de informații din sistem, prelucrările și rapoartele;
- identificarea și definirea cerințelor pentru noua aplicație;
- documentarea detaliată a tuturor cerințelor, funcționale și nefuncționale.

Analiza sistemului este o activitate esențială în aflarea situației existente și a ceea ce se dorește în viitor. Informațiile privind sistemul curent și cerințele pentru noul sistem se pot obține prin intermediul mai multor tehnici, cum ar fi: observarea a ceea ce fac utilizatorii, intervievarea lor sau sondarea pe bază de chestionare, analiza documentelor și a procedurilor de lucru, a regulilor economice și a responsabilităților fiecărui loc de muncă care este

influențat sau influențează funcționarea sistemului, a documentației sistemul informatic existent.

Dar nu este suficientă simpla culegere a informațiilor. Analistii trebuie să le revizuiască, să le analizeze și să le structureze astfel încât cerințele noii aplicații să fie ușor înțelese de către cei care se vor ocupa de proiectarea și scrierea programelor. Cerințele pot transpuse sub formă grafică, cu ajutorul diferitelor instrumente.

În finalul etapei de analiză, se pregătește o *documentație* necesară *proiectării*, prin care sunt surprinse toate aspectele privind modul de funcționare a sistemului curent, căror nevoi și cerințe trebuie să răspundă noua aplicație, ce funcții trebuie să cuprindă și ce obiective trebuie să sprijine.

În această etapă, economiștii joacă un rol covârșitor. Ei sunt cei care trebuie să descrie documentele primare, prelucrările realizate în sistem, procedurile de control, modalitățile de obținere a informațiilor și prezentare a lor în rapoarte. Informaticienii nu au cum să înțeleagă mecanismele economice desfășurate în cadrul firmei. Provocarea majoră în rândul economiștilor constă în structurarea tuturor informațiilor despre sistem într-o formă care să faciliteze comunicarea cu informaticienii. De aceea, ei trebuie să stăpânească instrumentele grafice de reprezentare a sistemului.

În etapa de **proiectare logică** se urmărește dezvoltarea arhitecturii aplicației, pregătirea specificațiilor de proiectare a interfețelor utilizator și a bazei de date, a controalelor și procedurilor de realizare a copiilor de siguranță, a programelor, independent de platformele pe care urmează să fie implementată aplicația. Aceste activități se bazează pe utilizarea informațiilor culese și a modelelor create în timpul analizei.

Proiectarea logică se derulează prin intermediul a trei pași sau subfaze:

- *proiectarea formularelor/formatelor (pentru culegerea datelor) și a rapoartelor;*
- *proiectarea interfețelor și a dialogurilor, pentru evidențierea modului de comunicare a utilizatorului cu programele și echipamentele;*
- *proiectarea logică a bazelor de date, prin care este concepută schema relațională a bazei de date, sub forma unui ansamblu de tabele normalizate între care există legături.*

Și în această etapă economiștii pot juca un rol important. Înțelegerea corectă a modului de funcționare a sistemului informațional este crucială în proiectarea interfețelor utilizator, a documentelor și rapoartelor, chiar și a bazei de date. Însă, pentru a face față acestor sarcini, economistul trebuie să dețină cunoștințe suplimentare de informatică, precum: modelul relațional și normalizarea bazelor de date, tipurile de obiecte regăsite în formularele, limbajul SQL.

După parcurgerea etapei de proiectare logică se trece la o nouă etapă, **proiectarea fizică**. Această etapă are un caracter preponderent tehnic, fiind orientată spre platformele pe

care va fi implementată aplicația. Ca atare, și specificațiile acestei etape vor avea un pronunțat caracter tehnic, economiștii intervenind rareori în derularea acestor activități.

Două dintre activitățile importante ale acestei etape privesc proiectarea fizică a bazei de date și proiectarea modulelor de program. De exemplu, la proiectarea fizică a bazei de date se vor lua în considerare facilitățile de stocare și accesare a datelor din SGBD-ul ales, precum tipul de organizare a fișierelor și utilizarea indecșilor.

După finalizarea proiectării, se trece la **etapa de implementare**, în timpul căreia sistemul este construit, testat și instalat. Obiectivul activităților specifice nu este numai de asigurare a funcționării aplicației, în concordanță cu nevoile identificate, ci și a faptului că utilizatorii sunt instruiți astfel încât firma să beneficieze de rezultatele prevăzute prin exploatarea corespunzătoare a acesteia. Principalele activități care se desfășoară în cadrul etapei sunt:

- scrierea și testarea programelor;
- construirea bazei de date;
- conversia datelor din vechea aplicație în formatul cerut de noua aplicație;
- instalarea aplicației pe calculatoare;
- instruirea utilizatorilor pentru oferirea tuturor explicațiilor și cunoștințelor necesare exploatării aplicației la parametrii la care a fost proiectat;
- elaborarea documentației aplicației și a manualelor de utilizare, exploatare și întreținere.

La finalul acestei etape, programul de aplicație se poate considera a fi funcțional, și se trece la **etapa de exploatare și întreținere**. În această etapă el trebuie supus revizuirilor periodice pentru a se asigura întreținerea acestuia, fie pentru corectarea eventualelor erori apărute în exploatarea lui, fie pentru îmbunătățirea caracteristicilor sau funcțiilor, ca răspuns la modificarea unor cerințe organizaționale. Această etapă este, de obicei, cea mai costisitoare, având în vedere că timpul pe care și-l petrec specialiștii din departamentele informatice reprezintă cam 48-60% din totalul timpului alocat pentru dezvoltarea sistemului.

1.3 INSTRUMENTE PENTRU DEZVOLTAREA APLICAȚIILOR

Enumerarea și descrierea instrumentelor software poate porni de la cele trei componente ale programelor de aplicații:

- **interfața utilizator**, adică o serie de componente (formulare pentru introducerea datelor, ferestre de dialog, meniuri, rapoarte) cu ajutorul cărora utilizatorul să poată iniția diferite operațiuni de culegere a datelor, prelucrare și obținere a informațiilor solicitate;
- **modulele de program**, care realizează prelucrările din sistem, cum ar fi calculul salariilor, întocmirea graficelor de rambursare etc.;
- **baza de date**, în care sunt organizate și stocate toate datele necesare aplicației.

Pentru dezvoltarea acestor componente se apelează la instrumente diferite, pe care încercăm să le prezentăm în continuare.

Mediile de programare sunt seturi de programe care includ următoarele funcții: introducerea și editarea programului sursă, instrumente pentru transformarea programului sursă în programe executabile sau biblioteci de funcții și instrumente pentru depanarea programelor. Într-un mediu de dezvoltare se pot folosi unul sau mai multe *limbaje de programare*. Exemple de medii de programare sunt: MS Visual Studio, Oracle JDeveloper, Oracle PowerObjects, Java, Borland Jbuilder, Borland Delphi, IBM Visual Age, Sybase PowerBuilder. În MS Visual Studio pot fi folosite mai multe limbaje de programare, cum ar fi: Visual Basic, C# sau C++.

Mediile de programare contribuie la realizarea în cadrul unei aplicații nu doar a modulelor de programe pentru prelucrarea datelor, ci și a interfeței utilizator și a legăturii cu bazele de date.

Pentru stocarea și gestionarea volumelor mari de date se apelează *sistemele de gestiune a bazelor de date*. Astăzi marea majoritate a aplicațiilor stochează datele în baze de date. Bazele de date cel mai des întâlnite sunt cele organizate după modelul relațional: Oracle, DB2 produs de IBM, Sybase SQL Server, MS SQL Server, MS Access, MS Visual FoxPro.

În condițiile globalizării afacerilor și a dezvoltării rețelelor de calculatoare se simte din ce în ce mai mult nevoia distribuirii resurselor informaționale ale companiei, respectiv date și/ sau programe, pe mai multe calculatoare. Dezvoltarea unei aplicații distribuite impune apelarea la *arhitectura client – server*, care presupune conceperea aplicației în termenii a două categorii de componente: componente client, care solicită anumite servicii și componente server, care furnizează servicii. Cele două categorii de componente vor fi rezidente pe două sau mai multe calculatoare.

Comunicarea între două componente aflate pe calculatoare diferite sau mascarea eterogenității platformelor hardware și software sunt rezolvate prin intermediul soluțiilor *middleware*. Ele sunt un set de programe care nu au doar rolul de a ascunde natura eterogenă a echipamentelor, sistemelor de operare sau a limbajelor de programare folosite în sistem, ci și de a oferi un mediu de programare mai comod. Cele mai cunoscute modele middleware sunt RPC (Remote Procedure Call), RMI (Remote Method Invocation), ODBC (Open Database Connectivity) și CORBA (Common Object Request Broker Architecture).

Indiferent de arhitectura software dorită, în cazul în care firma nu dispune de resurse financiare care să permită achiziționarea unor produse program necesare, se poate apela la soluția *soluții freeware*. Așa cum există aplicații financiar-contabile gratuite (de exemplu SAGA), tot așa există instrumente de dezvoltare software gratuite. Cel mai răspândit limbaj de programare folosit în medii de programare gratuite este Java. Un SGBD gratuit, care reușește să ajungă la un nivel de complexitate a prelucrării datelor comparabil cu cel al celor mai bine vândute SGBD-uri este PostgreSQL. Spre deosebire de alte SGBD-uri gratuite (MySQL), PostgreSQL este capabil să rezolve interogări complexe adresate bazelor de date pe care le gestionează. La capitolul SGBD-uri gratuite începe să concureze și unul dintre cele mai performante SGBD-uri relaționale din lume, Oracle, prin lansarea versiunii Oracle Database 10g Express Edition.

Dezvoltarea aplicațiilor Web presupune folosirea unor platforme diferite de mediile de programare specificate anterior. Pentru realizarea rapidă a unor site-uri Web se poate apela

la programul MS FrontPage. Ușurința în utilizarea acestui instrument software este dată de asemănarea interfeței sale cu interfețele programelor din cadrul MS Office, de faptul că utilizatorul nu este nevoit să învețe limbajul HTML și de folosirea unor șabloane.

Site-urile Web capătă o scalabilitate și o dinamică deosebită atunci când folosesc baze de date în care își salvează o parte din conținut. Trecerea la site-uri mai complexe nu presupune doar folosirea resurselor unei baze de date, ci și elemente grafice care să atragă atenția utilizatorului. Ca exemple de instrumente pentru construirea site-urilor Web complexe putem aminti Dreamweaver, ColdFusion, Flash și RoboDemo. Aplicațiile Web pot conține elemente de limbaj împrumutate din mediile de programare tradiționale, cum ar fi VB Scripts sau Java Scripts.

1.4 INSTRUMENTE SOFTWARE PENTRU ECONOMIȘTI

Economiștilor li se oferă astăzi o serie de instrumente software cu ajutorul cărora pot să-și dezvolte propriile aplicații informatice, e drept, de complexitate mai redusă. În acest paragraf dorim să trecem în revistă astfel de instrumente, cu referire la cele utilizate în dezvoltarea aplicațiilor economice. În acest sens, dorim să reținem atenția asupra instrumentelor CASE, programelor de calcul tabelar, sistemelor de gestiune a bazelor de date (SGBD), instrumentelor pentru crearea de pagini Web și a celor pentru organizarea informațiilor personale.

CASE (Computer Aided Software Engineering) reprezintă un instrument pentru automatizarea procesului de dezvoltare a programelor. El permite realizarea rapidă a unor programe de calitate, prin sprijinul care-l oferă tuturor activităților de dezvoltare, de la planificarea sistemelor informaționale până la generarea automată și testarea programelor.

CASE se bazează pe instrumente grafice și simple, orientate spre utilizatorii fără prea multe cunoștințe tehnice, fapt ce permite implicarea lor activă în procesul de dezvoltare a programelor. Economiștii au posibilitatea de a realiza variate sarcini, precum:

- planificarea resurselor financiare alocate proiectelor de sisteme informaționale,
- redactarea cerințelor funcționale ale aplicației,
- participarea activă la întocmirea specificațiilor de proiectare a programelor,
- generarea automată a programelor sau a bazei de date fără a avea cunoștințe tehnice privind limbajul de programare sau SGBD-ul folosit,
- testarea programelor,
- generarea automată a documentației pentru aplicația dezvoltată.

Implicarea directă a economiștilor în procesul de dezvoltare va determina sporirea șanselor de reușită datorită calității mai bune a programelor dar și creșterii probabilității de acceptare de către ei a noilor programe. Cele mai cunoscute instrumente CASE sunt Oracle Designer și Visible Analyst.

SGBD-urile sunt, fără îndoială, cele mai populare instrumente software în dezvoltarea aplicațiilor economice. Datorită acestui fapt, furnizorii au pus la dispoziția utilizatorilor nespecializați astfel de produse, cele mai utilizate fiind ACCESS, Visual FoxPro,

Corel Paradox sau Lotus Approach. Prin instrumentele grafice furnizate, ele se disting de produsele destinate profesioniștilor, precum Oracle. Apelând la un astfel de produs, utilizatorii vor fi în măsură să dezvolte aplicații de complexitate relativ redusă, fără a avea cunoștințe de specialitate.

Principalele facilități oferite de aceste SGBD-uri sunt:

- *Crearea și actualizarea bazei de date.* Utilizatorii pot defini structura datelor, sub forma tabelelor și a relațiilor dintre acestea, dar și aspectele de comportament a datelor, sub forma restricțiilor de integritate. De asemenea, ei pot adăuga, modifica sau șterge date fie prin intermediul unor ferestre, fie apelând la comenzile limbajului SQL.
- *Interogarea bazei de date.* Accesarea datelor din bază se poate face prin intermediul unor facilități grafice de interogare, dar și prin utilizarea frazei SELECT SQL.
- *Dezvoltarea de aplicații.* Utilizatorii au la dispoziție o serie de instrumente prin care pot crea formulare pentru introducerea datelor în bază, rapoarte pentru extragerea datelor și prezentarea informațiilor într-o formă plăcută, meniuri pentru exploatarea mai ușoară a aplicației, pagini Web. Prin integrarea lor se pot obține aplicații destul de complexe.

Programele de calcul tabelar, precum EXCEL, LOTUS 1-2-3, QuatroPro au reprezentat prima categorie de instrumente software destinate utilizatorilor neinformaticieni. Ele pot fi utilizate de economiști în activitățile de planificare și elaborarea bugetelor, în analize complexe privind activitatea firmei, în procesul decizional etc. Aceste programe sunt considerate instrumente elementare de asistare a procesului decizional.

Programele de calcul tabelar organizează datele în centralizatoare formate din linii și coloane, la intersecția cărora se găsesc căsuțe în care pot fi introduse date numerice și de tip text, precum și formule complexe de calcul. De asemenea, în centralizatoare pot fi create și afișate grafice. În momentul introducerii datelor în centralizatoare, calculele din căsuțele cu formule sunt efectuate automat, iar rezultatele sunt afișate imediat.

Principalele facilități de lucru sunt:

- o gamă largă de funcții predefinite, cum sunt cele financiare, statistice, matematice etc., pe care utilizatorii le pot folosi în elaborarea unor modele de calcul complexe;
- reprezentarea grafică a datelor din centralizatoare;
- regăsirea ușoară a datelor din centralizatoare;
- posibilitatea realizării de operații specifice bazelor de date, precum extragerea datelor în funcție de anumite criterii, ordonarea datelor, adăugarea, modificarea și ștergerea de date etc.
- efectuarea de analize de tip „What-if” („Ce se întâmplă dacă”). De exemplu, putem găsi răspunsul la întrebarea „Ce se întâmplă cu cota de piață dacă vom crește cheltuielile de promovare cu 10%?” Pentru a afla răspunsul la această întrebare, este suficient să introducem într-un centralizator relația de calcul și

datele privind factorii de influență. Simularea poate continua prin considerarea a unui procent de 15%.

- automatizarea unor sarcini de lucru repetitive, prin crearea de macro-programe.

Programele pentru gestiunea informațiilor personale au rolul de a ajuta utilizatorii în stocarea, organizarea și extragerea informațiilor privind clienții, diferite activități, întâlniri, ședințe etc. De exemplu, informațiile pot fi extrase și prezentate sub forma unui calendar electronic, o listă de întâlniri sau o planificare în timp a activităților unui proiect. Cele mai cunoscute astfel de programe sunt Lotus Organizer și Microsoft Outlook.

Furnizorii de software destinate nespecialiștilor oferă astăzi pachete integrate de programe sub forma unor **suite de programe**. Astfel de pachete includ diferite instrumente software care sprijină utilizatorii în creșterea productivității muncii lor. De regulă, într-o suită se regăsesc un SGBD, un program de calcul tabelar și un program pentru gestiunea informațiilor personale, la care se adaugă un procesor de texte și unul pentru grafică. În funcție de furnizor, suitele pot include și alte programe. Cele mai cunoscute suite, precum și câteva dintre componentele lor sunt prezentate în tabelul 1.1.

Tabelul 1.1 Principalele suite de programe și componentele lor

Pachete (suite)	Microsoft Office	Lotus SmartSuite	Corel WordPerfect Office
Calcul tabelar	Excel	Lotus 1-2-3	QuatroPro
SGBD	Access	Approach	Paradox
Gestiunea informațiilor personale	Outlook	Organizer	Corel Central

Avantajele principale ale suitelor de programe constau în costul mai redus față de cumpărarea individuală a fiecărui produs inclus, oferirea unei interfețe grafice uniforme, adică aceleași ferestre, meniuri, butoane, linii de stare etc., ceea ce le face mai ușor de învățat și, nu în ultimul rând, posibilitatea integrării aplicațiilor dezvoltate cu instrumente software diferite. De exemplu, datele din baza de date pot fi ușor exportate într-o aplicație dezvoltată în programul de calcul tabelar. În schimb, ele sunt criticate pentru faptul că sunt prea mari și ocupă prea mult spațiu pe calculator, mai ales că unele componente nu sunt folosite niciodată de cei mai mulți utilizatori.

1.5 TENDINȚE ÎN INSTRUMENTELE SOFTWARE

Progresele extrem de rapide înregistrate în ultimii ani în domeniul tehnologiilor informaționale și de comunicații își pun amprenta și asupra aplicațiilor informatice economice și a instrumentelor software pentru dezvoltarea acestora. Desprinderea unor tendințe în evoluția instrumentelor software este o încercare dificilă datorită numeroasele evoluții înregistrate în prezent, unele contradictorii.

Ca o tendință mai generală în ce privește dezvoltarea aplicațiilor informatice, se observă predilecția companiilor către achiziția programelor de aplicații de la furnizorii specializați, în detrimentul dezvoltării lor în cadrul campaniei, cu forțe proprii sau prin

angajarea de specialiști. În acest context, două acronime au devenit extrem de populare: **ERP** și **ASP**.

La mijlocul anilor '90, multe organizații au conștientizat că nu mai pot rezista pe piață dacă vor continua să utilizeze vechile sisteme informatice neintegrate. Nevoii de integrare i-au răspuns sistemele **ERP (Enterprise Resource Planning)**. ERP reprezintă o mega-aplicație multi-modulară care integrează procesele economice și optimizează resursele disponibile ale organizației, reunind toate funcțiunile sale într-o singură soluție software. ERP elimină barierele departamentale, prin integrarea tuturor sistemelor informaționale tranzacționale într-o singură bază de date, accesibilă oricui din organizație³. Nu în ultimul rând, se urmărește eliminarea granițelor organizației cu partenerii săi de afaceri. Două aplicații, adesea incluse în soluțiile ERP, stau drept mărturie: **CRM (Customer Relationship Management)**, centrată pe gestiunea relațiilor cu clienții, și **SCM (Supply Chain Management)**, orientată spre gestiunea relațiilor cu furnizorii.

ERP nu realizează doar integrarea funcțională, ci și pe cea tehnologică. Sub egida sa sunt reunite diverse tehnologii, precum: groupware, EDI (Electronic Data Interchange), Internet, Intranet, data warehouse (depozite de date) etc.

Principalele avantaje obținute prin apelarea la soluțiile ERP constau în: reducerea costurilor producției și a stocurilor, planificarea integrală a resurselor întreprinderii, îmbunătățirea productivității globale, maximizarea profitului prin flexibilitate și reactivitate sporită la cerințele pieței. Aceste avantaje nu derivă doar din integrarea funcțională și tehnologică, ci și ca urmare a expertizei de care sunt însoțite soluțiile ERP. Furnizorii de astfel de sisteme desfășoară ample activități de cercetare pentru a găsi cele mai bune practici în diferite domenii de afaceri.

Dincolo de aceste avantaje, apelarea pe scară largă la soluțiile ERP este deocamdată limitată, situație explicabilă prin trei mari neajunsuri: prețul exorbitant de mare, timpul îndelungat de implementare și adaptabilitatea redusă la condițiile particulare din firmă. Soluțiile ERP implică riscuri destul de mari pentru firme, legate de volumul mare al investițiilor inițiale, costuri ascunse semnificative, incertitudini privind adaptabilitatea ei și responsabilitățile sporite încredințate personalului.

Cel de-al doilea acronim, ASP, a devenit cunoscut odată cu tendința, din ce în ce mai accentuată, de externalizare a serviciilor informaționale, în scopul reducerii costurilor și îmbunătățirii performanțelor acestora. Externalizarea serviciilor informaționale cuprinde o paletă largă de posibilități, de la externalizarea integrală a lor și până la externalizarea dezvoltării unei părți a sistemului informațional. Externalizarea serviciilor informaționale este astăzi posibilă prin apelarea la furnizorii specializați, numiți **ASP (Application Service Providers)**. ASP reprezintă o companie care dezvoltă și furnizează servicii informaționale folosite în comun de mai mulți utilizatori, care plătesc un abonament sau taxe de folosire, serviciile fiind furnizate dintr-o locație centrală prin Internet sau printr-o rețea privată. Un ASP permite clienților săi accesul la un mediu de aplicații complet, preocupându-se de investițiile necesare în licențe de aplicații, servere, angajați și alte resurse.

³ Fotache D., Hurbean, L. – *Soluții informatice integrate pentru gestiunea afacerilor – ERP*, Ed. Economică, București, 2004, p. 16

Apelarea la serviciile unor astfel de furnizori oferă numeroase beneficii, dintre care amintim:

- investiții inițiale modeste și predictibilitatea costurilor, deoarece pentru aceste servicii se plătește o taxă fixă pe principiul “plătești pe măsură ce utilizezi”;
- posibilitatea de a fi mereu în pas cu progresele tehnologice;
- posibilitatea închirierii aplicațiilor scumpe, inaccesibile companiilor mici și mijlocii sau a celor cu investiții inițiale importante;
- acces la suport tehnic și consultanță de specialitate pentru servicii cu înalt nivel tehnologic;
- obținerea de aplicații funcționale într-un interval de timp foarte scurt.

În ultimul timp se manifestă tendința extinderii serviciilor furnizate, conceptul ASP fiind înlocuit cu **XSP**. Spre deosebire de ASP, XSP (X semnifică servicii generice) nu oferă doar aplicații, ci și tehnologii, sisteme și procese economice. Actorii de pe această piață vor fi marile companii (de exemplu Microsoft, Oracle, Sun), care sunt capabile să realizeze investiții foarte mari pentru crearea infrastructurii necesare oferirii de servicii în rețea bazate pe Internet. Modelul XSP promite numeroase beneficii: firmele vor putea să cumpere doar funcționalitatea pe care o doresc și atunci când o doresc, reducând astfel investițiile în tehnologiile informaționale; transferul responsabilității și riscurilor menținerii permanente în pas cu noile tehnologii; firmele pot activa pe piață nu doar pe post de consumator de servicii ci și cel de furnizor; ușurința modificării proceselor și a relațiilor de parteneriat în funcție de condițiile de pe piață.

Dincolo de aceste beneficii, unele probleme rămân încă nerezolvate sau, cel puțin, neclare: securitatea serviciilor informaționale disponibile pe piață; protecția datelor sensibile ale clienților; posibilitatea furnizării de aplicații critice pe web în condiții de siguranță sporită; portabilitatea aplicațiilor și interoperabilitatea componentelor sale.

În ce privește strict limbajele de programare, reținem două tendințe: limbajele orientate-obiect, limbajele orientate-Internet și limbajele naturale. Ele nu sunt tocmai noi, însă în prezent se înregistrează o extindere a utilizării lor, în încercările obsedante de apropiere de limbajul uman și de îmbunătățire a productivității muncii programatorilor.

Limbajele orientate-obiect modelează sistemul informațional sub forma unui set de obiecte care interacționează între ele prin schimbul de mesaje. Un obiect integrează datele și acțiunile (procedurile) care pot fi executate asupra datelor, numite metode. Astfel, spre deosebire de limbajele anterioare, datele și programele de prelucrare a acestora nu mai sunt separate. Într-o bancă, contul de card al unui client reprezintă un obiect. Numele clientului, adresa sa, numărul contului, numărul cardului, soldul contului, limita de credit reprezintă datele, referite și ca **proprietăți**, iar retragerea de numerar, depunerea, transferul bancar, schimbarea cardului reprezintă acțiuni, numite **metode**, care descriu comportamentul obiectului.

Una dintre caracteristicile acestor limbaje se referă la faptul că datele unui obiect sunt ascunse față de alte părți ale programului, proprietate numită **încapsulare**. Datele pot fi manipulate numai din interiorul obiectului, prin intermediul metodelor disponibile. Un alt obiect sau un utilizator care dorește să modifice anumite date vor trebui să apeleze una dintre

metodele obiectului. În acest fel, fiecare obiect reprezintă o unitate independentă de program, ce poate fi utilizată în diferite moduri fără a fi necesară modificarea programului. Această caracteristică permite reducerea drastică a timpului și costului de scriere a programelor, prin reutilizarea programelor în alte aplicații asemănătoare.

Reutilizabilitatea programelor este facilitată de alte două concepte specifice limbajelor orientate-obiect: **clasă** și **moștenire**. Obiectele cu proprietăți similare sunt reunite în aceeași clasă. Clasele sunt organizate ierarhic în superclase și subclase. Moștenirea permite definirea unei clase de obiecte ca un caz particular al altei clase mai generale, numită superclasă. O clasă va moșteni proprietățile și metodele superclasei, însă va putea modifica definițiile acestora sau adăuga proprietăți și metode noi. De exemplu, STUDENT poate fi considerat o clasă în sistemul de evidență a studenților, iar un anumit student va reprezenta o instanță a acestei clase, adică un obiect. Clasa STUDENT poate fi considerată o subclasă a FIINTA_UMANA, aceasta fiind superclasa. La rândul său, STUDENT poate avea ca subclasă STUDENT_BURSIER. Această clasă va avea, în plus față de student, proprietatea TipBursa. Oricum, proprietățile și metodele clasei STUDENT sunt moștenite de STUDENT_BURSIER.

Un alt mare avantaj al limbajelor orientate-obiect, pe lângă reutilizabilitate, se referă la faptul sunt mai apropiate de modul în care utilizatorii văd lumea. Se consideră că este mult mai naturală descrierea realității înconjurătoare în termenii claselor de obiecte, cu proprietăți și comportament, decât sub forma unor funcții de prelucrare și a datelor, așa cum cereau limbajele anterioare.

Limbajele orientate-obiect au apărut în anii '70, odată cu Smalltalk. Astăzi, cele mai utilizate limbaje sunt Java și C#.

Dezvoltarea limbajelor orientate-obiect a determinat apariția unei alte tehnologii, numită **programarea vizuală**. Aceste limbaje dispun de un mediu grafic ce permite programatorilor să dezvolte aplicații prin manipularea directă a unor imagini, în loc de scrierea de programe. Visual Basic este cel mai cunoscut astfel de limbaj, însă tot aici se înscriu DELPHI, Power Objects și Visual C++.

Limbajele de programare orientate-obiect au influențat și alte domenii ale sistemelor informaționale, astfel că astăzi există metode de analiză și proiectare orientate-obiect, sisteme de gestiune a bazelor de date orientate obiect (SGBDOO), instrumente CASE orientate-obiect.

Limbajele orientate-Internet se referă la tehnologiile care stau la baza dezvoltării aplicațiilor Web, cele mai importante fiind limbajele HTML și XML.

HTML (HyperText Markup Language) reprezintă un limbaj de descriere utilizat pentru crearea paginilor Web sub forma documentelor hypertext sau hypermedia. El este derivat din *SGML (Standard Generalized Markup Language)*, o metodă de reprezentare a limbajelor de formatare a documentelor, ceea ce permite ca la crearea documentelor să se separe informația de modul de prezentare a ei. Aceasta înseamnă că documente care conțin aceeași informație pot fi diferite prin modul de prezentare, adică tipul și mărimea fontului, spațierea paragrafelor etc., fără ca informația să fie modificată. HTML utilizează instrucțiuni numite *tag-uri*, pentru a descrie cum vor fi plasate în document informațiile de tip text, grafică, video și sunet și pentru a crea legături dinamice (numite și hiper-legături) către alte

documente și obiecte stocate pe același calculator sau pe altul. Aceste legături permit utilizatorului să deschidă un alt document printr-un simplu clic pe un text subliniat sau pe o imagine.

Popularitatea înregistrată de acest limbaj a determinat principalii furnizori de software să-l includă în produsele lor. Astăzi, majoritatea procesoarelor de text, programelor de calcul tabelar, programelor de procesare grafică sau sistemelor de gestiune a bazelor de date oferă facilități pentru generarea automată a documentelor în format HTML. De asemenea, după cum am văzut în paragraful anterior, au fost puse la dispoziția utilizatorilor instrumente software speciale pentru dezvoltarea paginilor Web, fără a fi nevoie să se scrie programe HTML. Cele mai cunoscute sunt Microsoft FrontPage și Lotus FastSite.

XML (eXtensible Markup Language) nu este un limbaj de descriere a formatului unei pagini Web, ci un limbaj de descriere a conținutului paginilor Web. De exemplu, informațiile privind zborurile aeriene de pe pagina Web a unei agenții de turism vor fi descrise prin intermediul tag-urilor XML. El este derivat tot din standardul SGML.

Exemple de teste grilă

1. Care dintre următoarele instrumente software se adresează economiștilor, ca utilizator final informatizat?
 - a) Programe de calcul tabelar
 - b) Limbajul Java
 - c) Produsele de tip CASE
 - d) Limbajele de ansamblare (numite și simbolice)
2. Evoluția limbajelor de programare reflectă progresele înregistrate în privința:
 - a) Eficienței utilizării resurselor calculatoarelor
 - b) Productivității muncii programatorilor
 - c) Apropierii de limbajul uman
 - d) Memorarea datelor