CNIV-2003, Noi tehnologii de e-learning, Conferința Națională de Învățământ Virtual,
Software educațional, Editura Universității din Bucureşti, 2003 (ISBN 973-575-822-9)

Gândirea algoritmică - o filosofie modernă a matematicii şi informaticii

Marin Vlada – Universitatea din Bucuresti,
vlada@fmi.unibuc.ro

Abstract

Dezvoltarea informaticii actuale se datorează cercetărilor, rezultatelor şi experienţelor din domeniile SISTEMELOR DE CALCUL, ALGORITMICII şi PROGRAMĂRII, dar mai ales a interdependenței acestor domenii prin aşa-numita triadă "SISTEM DE CALCUL - ALGORITMICĂ - PROGRAMARE". La baza acestei interdependenţe se află conceptul de ALGORITM, concept ce a construit pentru om o nouă filosofie: GĂNDIREA ALGORITMICĂ. Conceptul a fost introdus în matematica veche, utilizat de matematica modernă şi perfecţionat de informatică în utilizarea calculatoarelor moderne. Această gândire algoritmică a făcut posibilă apariția şi dezvoltarea Tehnologiei Informației (IT) ce reprezintă de fapt implementarea filosofiei procesării, gestionării şi comunicării informațiilor şi cunoştinţelor.

1. Introducere

Întâmplător sau nu, **deceniul 7** al secolului al XX-lea a fost unul al *marilor schimbări* în domeniul *informaticii* şi al *sistemelor de calcul*:

- **inventarea microprocesorului** ("*bijuteria de bază*" a actualelor sisteme de calcul) ca urmare a rezultatelor din trei domenii fundamentale : sisteme, circuite integrate şi microprogramare; a urmat construirea şi răspândirea pe scară largă a sistemelor de calcul de tip PC(Personal Computer), impulsionarea dezvoltării rețelelor de calculatoare, apariția şi dezvoltarea de noi sisteme de operare; performanţe sporite ale dispozitivelor I/O;
- **supremația şi răspândirea structurilor de control** în *algoritmică* şi *programare*; apariția *limbajului pseudocod* în reprezentarea şi elaborarea algoritmilor; conceperea şi scrierea primelor limbaje de programare care implementează *structurile de control* (Limbajele **Pascal** şi **C**), adaptarea continuă a limbajelor de programare prin implementarea *structurilor de control* , a *structurilor de date, a facilităților programării orientate spre obiecte (OOP- Object Oriented Programming)*;
- **succese spectaculoase în domeniul Inteligenței Artificiale** prin construirea primelor *sisteme expert*; conceperea şi scrierea primului *limbaj de programare logică* (Limbajul **Prolog**) ce oferă suportul *programării declarative*; dezvoltarea şi utilizarea largă a metodelor şi tehnicilor *Inteligenței Artificiale* în rezolvarea câtorva dintre cele mai dificile probleme;
- **delimitarea problemelor** rezolvate cu calculatorul(*probleme decidabile*) în două *clase distincte*: clasa problemelor rezolvate prin *metode imperative(procedurale)* şi clasa problemelor rezolvate prin *metode declarative*; delimitarea clasei *problemelor nedecidabile*; apariția şi dezvoltarea tehnicilor pentru comunicarea, căutarea şi vizualizarea la distanță a informațiilor.

Toate aceste aspecte au fost şi sunt într-o *interdependența* continuă ținând seama de particularitatea informaticii care oferă *sisteme de calcu*l performante şi *produse-program* competitive în rezolvarea problemelor. Utilizarea eficientă a sistemelor de calcul şi a produselor-program reclamă o instruire continuă, atât pentru *informaticieni-programatori*, cât şi pentru *utilizatori*.

Dezvoltarea ***gândirii algoritmice*** trebuie luată în considerare ca obiectiv în instruire, atunci când se învață *algoritmică* (metode şi tehnici), dar şi când se învață *programarea* (limbaje de programare). Practica instruirii elevilor şi studenților a demonstrat că invățarea unui limbaj de programare este în general "*mai uşoară*" decât invățarea elaborării algoritmilor (algoritmică). Acest lucru se poate justifica prin faptul că elaborarea unui *algoritm* este echivalentă cu implementarea(reprezentarea) *raționamentelor(procese demonstrative)* deduse din **metode şi tehnici** utilizate în rezolvarea unei probleme. **Rezolvarea problemelor** necesită nu numai *cunoştințe clare şi precise*, dar şi *capacitate de sinteză* şi *control* şi mai ales *capacitate de creație*. Dacă vrem să facem o analogie, un *programator* poate fi "*compozitorul*" ce realizează o "*lucrare muzicală*". În toate etapele instruirii se va avea în vedere *interdependența*:

*"*SISTEM DE CALCUL – ALGORITMICĂ – PROGRAMARE*"*

Succesele unui *concurent* la Olimpiadele de Informatică depind de cunoaşterea utilizării unui limbaj de programare modern, dar mai ales depind de « *bogăția* » şi stapânirea cunoştințelor în *elaborarea algoritmilor*. Şi mai există încă ceva: *experința acumulată* în activitatea de rezolvare a problemelor prin formarea unei *gândiri algoritmice* solide şi consistente. Acelaşi lucru se poate afirma şi despre succesele unui *programator* ce lucrează într-o firmă de *produse software*.

2. Conceptul de algoritm şi algoritmizarea

Complexitatea problemelor care necesită descrierea mai multor **procese de calcul** complexe a determinat folosirea noțiunii de **algoritm** în activitatea de rezolvare a problemelor. Multe procese naturale, multe activități umane, pot fi descrise într-o **formă algoritmică** prin definirea unor *informații* şi *acțiuni* clare şi precise, eliminându-se ambiguitățile în *interpretare* şi în *operații*. **Algoritmizarea** este o cerință fundamentală în rezolvarea oricărei probleme cu ajutorul calculatorului.

Experiența a demonstrat că *nu orice problemă* poate fi rezolvată prin *algoritmizarea rezolvării*, adică prin descrierea unui algoritm de rezolvare. Aşa s-a delimitat *clasa problemelor decidabile* (o problemă este decidabilă dacă există un algoritm pentru rezolvarea ei) de *clasa problemelor nedecidabile* (o problemă este nedecidabilă dacă nu există un algoritm pentru rezolvarea ei).

Un algoritm implementează diverse metode şi tehnici de rezolvare care au fost descoperite sau definitivate într-un anumit moment în evoluția ştiințifică a domeniului respectiv. Există algoritmi ce urmează metode dezvoltate înainte de apariția calculatoarelor, dar cele mai multe probleme cer **abordări noi**. Chiar dacă ne gândim numai la "**problema celor 4 culori**" , care a fost rezolvată (*în anul 1977*) doar prin utilizarea calculatorului şi prin utilizarea unei metode noi (metoda *Backtracking*), facem dovada acestei afirmații.

În etapa actuală de dezvoltare ştiinţifică şi tehnică, **rezolvarea unei probleme** dintr-un domeniu (*matematică, informatică*, *fizică, chimie, etc.*) reprezintă o *activitate de creaţie, un raţionament* prin construirea, generarea, descrierea unui:

- **proces demonstrativ**(*demonstraţia*) care să arate existenţa unei soluţii sau a mai multor soluţii şi/sau să determine efectiv *soluţiile exacte*;
- **proces computaţional**(*algoritmul*) care să codifice un *proces demonstrativ*, o metodă sau o tehnică de rezolvare în scopul *determinării (eventual aproximative)* a soluţiilor exacte.

În general, în procesul (activitatea) de *rezolvare a unei probleme* dintr-un anumit domeniu (ştiinţific, economic, social, etc.), este necesară evidenţierea *ipotezei* (condiţiile, stările iniţiale, parametrii iniţiali) şi a *concluziei* (cerinţele, obiectivele, scopurile) din analiza şi studiul *enunţului* problemei. *Procesul de rezolvare* (raţionamentul) constă în utilizarea selectivă a *legilor*, *teoremelor*, *propoziţiilor*, etc. din domeniul problemei, pentru ca pornind de la *ipoteză*(**I**), prin aplicarea succesivă a legilor, teoremelor, etc. să se obţină *concluzia*(**C**) problemei. Legătura dintre *ipoteză*, *concluzie* şi *procesul de rezolvare* (*raţionamentul* - **R**) determină o *structură* asemănătoare conceptului de *program*, şi anume (*Ipoteză* - *Date de intrare*; *Concluzia* - *Date de ieşire*) :

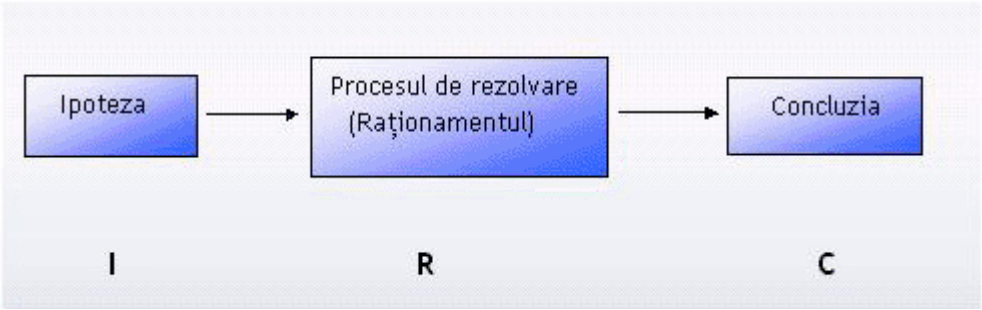


Figura 1. Rezolvarea unei probleme

De exemplu, *procesul de rezolvare* a unei probleme poate fi descris astfel: există un număr de *teoreme* $T_1, \ldots, T_n$ determinate de *ipotezele* $I_1, \ldots, I_n$ şi de *concluziile* $C_1, \ldots, C_n$; acestea sunt selectate şi apoi aplicate astfel încât să se poată realiza identificările:

$$\mathbf{I} \supseteq I_1, C_1 \supseteq I_2, \ldots, C_n \supseteq \mathbf{C}, \text{ adică:}$$

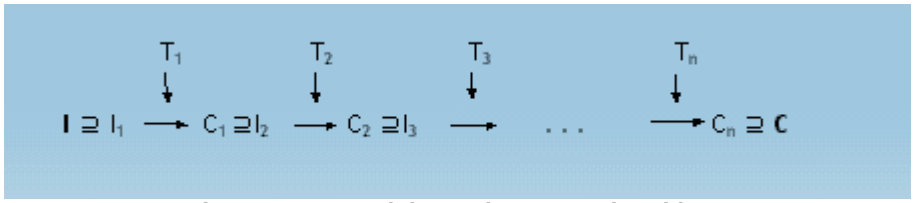


Figura 2. Precesul de rezolvare a unei probleme

Această prezentare poate fi înţeleasă dacă, de exemplu, facem analogia cu rezolvarea *problemelor de geometrie* care necesită delimitarea *ipotezei* şi a *concluziei,* ca apoi să se utilizeze numai acele *teoreme, propoziţii, proprietăţi* care, pornind de la ipoteză, prin aplicarea succesivă, să se obţină concluzia cerută. Evident, din experinţa

căpătată în *rezolvarea problemelor*, se poate afirma faptul că selectarea teoremelor şi aplicarea lor se poate realiza doar prin "*stăpânirea*" domeniului respectiv la nivel de specialist sau de expert. Este cunoscut faptul că omul poate rezolva o problemă în trei moduri (metode evidențiate de *Inteligența Artificială*):

- pornind de la *ipoteză* şi obținând *concluzia* ("*metoda înlănțuirii înainte*");
- pornind de la *concluzie* şi obținând *ipoteza* ("*metoda înlănțuirii înapoi*");
- pornind simultan, de la *ipoteză* şi *concluzie* ("*metoda mixtă*").

În toate *ştiintele*(matematică,fizică, chimie, etc.) există multe probleme care, chiar dacă au fost studiate într-o anumită etapă, abordarea lor cu *metode şi tehnici moderne* poate să trezească oricând interesul tinerilor, al cercetătorilor în general. Aceasta cu atât mai mult cu cât astăzi, prin utilizarea unor metode specifice *tehnologiei informatiei* şi prin utilizarea *calculatorului*, se pot concepe tehnici care altădată erau imposibil de imaginat.

Din punct de vedere *metodologic*, trebuie să ne obişnuim să reformulăm problemele în mod explicit şi adecvat rezolvării lor matematice sau- când este posibil – cu ajutorul calculatorului. Pentru acest lucru trebuie să cunoaştem "*limitele*" **gândirii demonstrative** şi în acelaşi timp "*limitele*" **gândirii algoritmice**, apoi să cunoaştem performanțele calculatoarelor.

Să amintim aici istoria problemei celor 4 culori care începe în anul 1852 , când studentul englez *Francis Guthrie* a enunțat urmatoarea problemă pentru colorarea unei hărți:

"*Sunt suficiente 4 culori pentru a colora o hartă ce reprezintă diverse țări, cu condiția ca oricare două țări vecine (ce au frontiera comună) să fie colorate cu culori diferite*".

Mulți matematicieni cu renume au studiat această problemă (**A. de Morgan, A. Cayley, A. B. Kempe, H. Heesch, K. Appel, W. Hakel**), dar rezolvarea ei completă nu a fost posibilă decât în anul 1977, (*K. Appel, W. Hakel, "The four color problem " în Mathematics Today , L. A. Steen(ed.) , Springer Verlag , 1978*) şi acest lucru numai după ce s-a utilizat calculatorul, deoarece fiind o *problemă combinatorială* cu spațiul soluțiilor de ordin foarte mare, a fost necesară conceperea unei *metode algoritmice* pentru căutarea eficientă cu ajutorul calculatorului. Teoretic, problema a fost rezolvată în cazul a *5 culori*. De altfel, rezolvarea acestei probleme a impus în Informatică o metodă devenită clasică în domeniul elaborării algoritmilor: *metoda backtracking*.

În zilele noastre apar tot mai des probleme a căror ***rezolvare*** (*proces demonstrativ*) este rezultatul îmbinării *metodelor pur matematice* cu ***metodele algoritmice*** (*computaționale*). Uneori, rezultate computaționale pot oferi surse de inspirație pentru metodele matematice. Informaticienii au şi venit în întâmpinarea matematicienilor şi specialiştilor prin elaborarea diverselor *sisteme de programe* care realizează *calcule numerice*, *logice*, *simbolice*, *reprezentări spațiale*, etc. cu ajutorul cărora pot fi dobândite cunoştinte matematice sau se pot rezolva probleme din diverse domenii. Totuşi, aceste aspecte nu trebuie să conducă la neglijarea procesului demonstrativ. Cele mai utilizate astfel de *sisteme de programe* sunt: MATHEMATICA, STATISTICA, MATLAB, MATHCAD, DERIVE, MAPLE, ORIGIN, SLIDEWRITE, WORKPLACE, EUREKA, etc.

Este cunoscut faptul că în ştiințele naturii se obțin rezultate deosebite prin îmbinarea metodelor teoretice cu metodele experimentale. Pentru *matematică(şi alte ştiințe)*,

informatica(tehnologia informației) este cea care oferă rezultate experimentale prin utilizarea calculatorului în rezolvarea problemelor.

Metoda algoritmică va fi un experiment pentru justificarea rezultatelor *procesului demonstrativ* de la *metoda matematică*, dar în acelaşi timp poate să fie o metodă de rezolvare independentă. Metoda algoritmică trebuie să substituie rezultatele obținute pe cale teoretică cu metode computaționale eficiente ținând seama de limitele proceselor algoritmice şi de performanțele calculatoarelor. Este adevărat că uneori metodele matematice pot conduce la simplificarea metodelor algoritmice şi invers. *Elevii* care participă la olimpiadele şcolare de informatică au constatat că, de multe ori, o analiză profundă a abordării matematice, poate să conducă la obținerea unei metode algoritmice eficiente. Participanții la olimpiadele de matematică pot constata acelaşi lucru la o analiză profundă a abordării algoritmice. De aici, importanța atât a **proceselor demonstrative** (*metode matematice*) , cât şi a **proceselor computaționale** (*metode algoritmice*).

În funcție de natura metodelor/tehnicilor implementate în procesele computaționale, *algoritmii* pot fi: *numerici, seminumerici, formali, combinatoriali, neuronali, de căutare, de sortare, recursivi, de rescriere, secvențiali, paraleli, determinişti, nedeterminişti, probabilişti, aleatori, euristici, de tip Monte Carlo, genetici, de simulare, computational geometry***,** etc.

Întreaga activitate de cercetare şi elaborare de software din domeniul *Tehnologiei Informației* este determinată de inventarea, conceperea, elaborarea, testarea, şi implementarea de *algoritmi performanți* şi *utili*. Marea diversitate a algoritmilor şi marea aplicabilitate a acestora în toate domeniile, face ca această *temă* sa fie mereu actuală şi într-o continuă schimbare şi perfecționare.

3. Gândirea algoritmică şi gândirea obiectuală

Practica *rezolvării problemelor* folosind un *limbaj de programare* a determinat de-a lungul timpului diverse abordări în funcție de performanța limbajului de programare, performanța calculatorului şi nu în ultimul rând, în funcție de metodele şi tehnicile avansate privind implementarea raționamentelor pentru demonstrațiile corespunzătoare problemelor.

Rezolvarea teoretică a unei probleme nu garantează şi *rezolvarea ei practică* cu calculatorul. În general, un limbaj de programare este menit să faciliteze rezolvarea unor *clase de probleme* şi se pretează mai bine anumitor *tipuri de algoritmi*. Este nevoie de experiență în utilizarea şi cunoaşterea calculatorului, de competență şi intuiție, este nevoie de inspirație şi creație. În astfel de situatii *este nevoie de cunoaşterea mai multor limbaje de programare* pentru a alege *limbajul de programare* adecvat pentru *clasa de probleme* din care face parte problema de rezolvat. Experiența a arătat că, atunci când nu este ales limbajul de programare corespunzător, dacă totusi se ajunge să se rezolve problema, s-a făcut risipă de resurse (timp / memorie / finanțe, etc.) şi, prin urmare, eficiența şi performanța au avut de suferit.

Astăzi, perfomanța unui *informatician-programator* este determinată de experiența şi competența obținută în desfaşurarea celor două etape (ANALIZĂ, PROGRAMARE) :

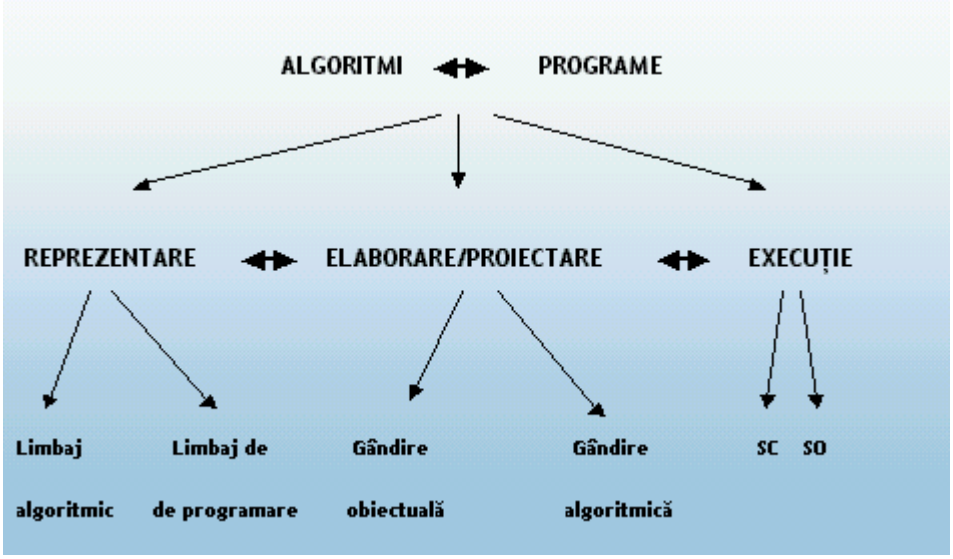
- *etapa gândirii obiectuale* (ANALIZĂ - PROIECTARE) – modul de analiză şi abstactizare a problemelor prin definirea corectă a obiectelor, a tipurilor de obiecte, a relațiilor între obiecte şi a operatorilor specifici(*elaborarea UAP ; etapa de concepție şi analiză- proiectare*) ;
- *etapa gândirii algoritmice* (PROGRAMARE - EXECUȚIE) – alegerea şi aplicarea corectă a unor metode de rezolvare prin precizarea exactă a operatorilor de prelucrare a obiectelor, reprezentarea corectă a strategiilor algoritmice, reprezentarea codificată a obiectelor şi a prelucrărilor conform unui limbaj de programare (*elaborarea algoritmului şi programului ; etapa de programare - codificare implementare şi execuție*).

În prezent este tot mai des invocată *reprezentarea problemelor* folosind concepte **OOP** (**O**bject **O**riented **P**rogramming). Conceptul de *obiect* (M. Minsky, *The Society of Mind*, Touchstone Books, New York, 1986) are un rol important în *ştiința cunoasterii* şi *educației*. Un *obiect* modelează o *entitate* din **lumea reală** sau **virtuală**. În *activitatea de rezolvare a problemelor* trebuie să se *identifice/definească* obiectele din cadrul problemelor ce provin din diferite *domenii*: *ştiințifice, economice, sociale*, etc. *Identificarea* obiectelor este echivalentă cu determinarea entităților şi conceptelor care reprezintă *forme fizice/grafice, fapte, evenimente, procese, stări*, etc. Un *obiect* este caracterizat în mod unic prin *identificare*, *comportament*(caracteristică dinamică), şi *stare*(caracteristică statică). În esență, rezolvarea unei probleme se va exprima printr-o *codificare* a *universului problemei* şi a *raționamentelor* pentru procesul demonstrativ.

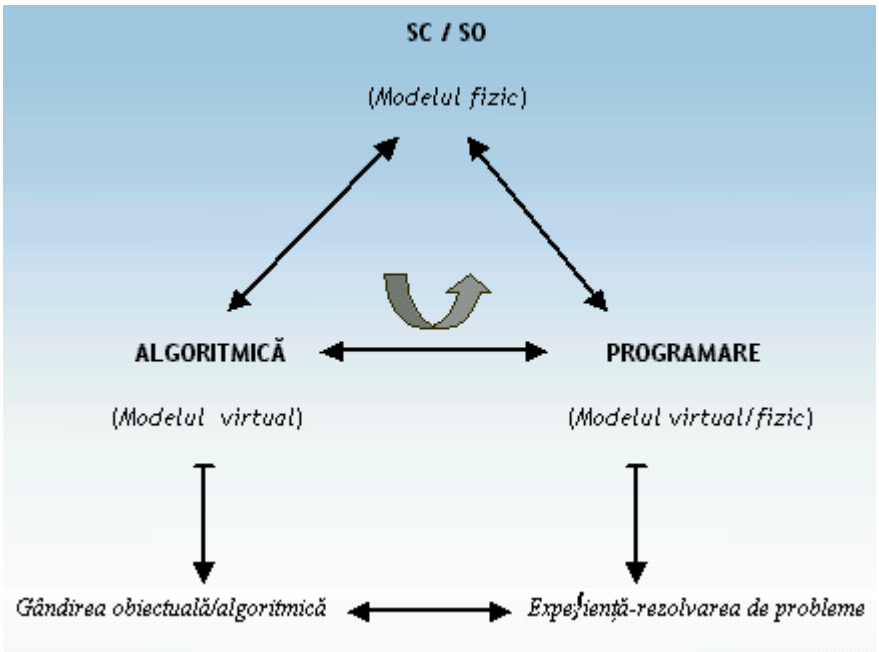
Practica *dezvoltării aplicațiilor software*(care necesită rezolvarea diverselor tipuri de probleme) a scos la iveală urmatoarele faze importante (*gândirea obiectuală* :fazele 1 şi 2= *analiză-proiectare*; *gândirea algoritmică*: fazele 3-6= *programare-execuție*):

1. *specificarea problemelor* – descrierea clară şi precisă a problemelor indiferent din ce domeniu provin acestea ;
2. *proiectarea soluțiilor* – includerea problemelor în clasa de probleme corespunzătoare şi alegerea modului de reprezentare a problemelor prin formularea *etapelor* şi *procedeelor* corespunzătoare pentru procesele de rezolvare;
3. *implementarea soluțiilor – elaborarea algoritmilor* şi *codificarea* acestora într-un limbaj de programare modern;
4. *analiza soluțiilor – eficiența soluțiilor* raportată la resursele utilizate: *memorie, timp, utilizarea dispozitivelor I/O*, etc.;
5. *testarea şi depanarea – verificarea execuției programului* cu diverse seturi de date de intrare pentru a putea răspunde rezolvării oricărei probleme pentru care aplicația a fost elaborată;
6. *actualizarea şi întreținerea* – adaptarea soluțiilor implementate pentru *eliminarea erorilor* în rezolvarea unei anumite probleme şi *compatibilitatea* cu sistemul de calcul şi sistemul de operare folosite.

Conform acestor faze iese în evidență *interdependența* între urmatoarele *activități* importante: REPREZENTARE – ELABORARE / PROIECTARE – EXECUȚIE, şi anume se poate sintetiza prin următoarea schemă(SC=*sistem de calcul*, SO=*sistem de operare*):



De asemenea, *interdependența* precizată mai sus se transmite şi între componentele de pe nivelul inferior din schema arborescentă alăturată. *Competența* şi *experiența* în *rezolvarea problemelor* se pot obține doar dacă permanent se are în vedere această interdependență şi dacă se întreprind eforturi pentru î*nsuşirea de noi cunoştințe* şi pentru *conoaşterea* corespunzătoare a tuturor aspectelor privind MODELUL FIZIC , respectiv MODELUL VIRTUAL, aspecte determinate de interdependența SISTEM DE CALCUL – ALGORITMICĂ – PROGRAMARE:



Practica şi experienţa elaborării programelor pentru rezolvarea problemelor scot în evidenţă următoarele aspecte foarte importante:

- **Modelul fizic**- acest model este dat de *sistemul de calcul* şi *sistemul de operare*, model ce trebuie luat în considerare când se proiectează şi se elaborează o aplicaţie; acest aspect reclamă competenţă în domeniul sistemelor de calcul şi perfecţionare continuă pentru cel care proiectează şi elaborează aplicaţia;
- **Modelul virtual** – acest model este dat *de gândirea obiectuală şi algoritmică*, de modul de *reprezentare a algoritmilor*, de *maşina virtuală* pe care trebuie să se *execute* algoritmul elaborat; în timp, acest model a suferit schimbări majore deoarece a fost tot timpul influenţat de *modelul fizic* şi de *clasa problemelor* ce urmau să fie rezolvate;
- **Modelul program** – acest model este reprezentat de o îmbinare între *modelul fizic*(SC/SO) şi *modelul virtual* (Algoritmul); întotdeauna un program se elaborează într-un *limbaj de programare* care trebuie să respecte *restricţiile modelului fizic* (sistemul de calcul şi sistemul de operare) şi *restricţiile modelului virtual* (algoritmul)

4. Bibliografie

[1] Appel, K. and Haken, W. "Every Planar Map is Four-Colorable, II: Reducibility." *Illinois J. Math.* 21, 91-567, 1977.
[2] Appel, K. and Haken, W. "The Solution of the Four-Color Map Problem." *Sci. Amer.* 237, 108-121, 1977.
[3] Brassard, G. and Bratley, P. *Fundamentals of Algorithmics.* Englewood Cliffs, NJ: Prentice-Hall, 1995.
[4] Cristea, V., C. Giumale, E, Kalisz, Al. Paunoiu, Limbajul C standard*, Ed. Teora,* Bucureşti, 1992.
[5] Knuth, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms, 3rd ed.* Reading, MA: Addison-Wesley, 1997.
[6] Knuth, D. E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, 3rd ed.* Reading, MA: Addison-Wesley, 1998.
[7] Knuth, D. E. *The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed.* Reading, MA: Addison-Wesley, 1998.
[8] Chabert, J.-L. (Ed.). *A History of Algorithms: From the Pebble to the Microchip.* New York: Springer-Verlag, 1999.
[9] Popovici, M. D., Popovici, M. I., C++. Tehnologia orientată spre obiecte. Aplicaţii, *Ed. Teora,* Bucureşti, 2000.
[10] Vlada*,* M., Informatică*, Universitatea din Bucureşti, Ed. Ars Docendi,* Bucureşti, 1999.
[11] Vlada, M., Poligoane stelate. Problema lui Hopf şi Pannwitz*, Gazeta de matematică, nr. 8/1995, pag. 339-348.*
[12] Vlada, M., Rezolvarea problemelor folosind Eureka, *software educaţional*, www.unibuc.ro/eBooks/informatica/eureka/, *Universitatea din Bucuresti*, 2003.
[13] Vlada, M., Concepul de algoritm-abordare modernă, *Gazeta de informatică*, vol. 13/2 şi 3, pp. 25-30, pp. 35-39, *Agora*, Cluj Napoca, 2003.
[14] Zaharia*,* M. D., Structuri de date şi algoritmi. Exemple în limbajele C şi C++, *Ed. Albastră,* Cluj-Napoca, 2002.
[15] http://www-groups.dcs.st-and.ac.uk/~history/BigPictures/Guthrie.jpeg