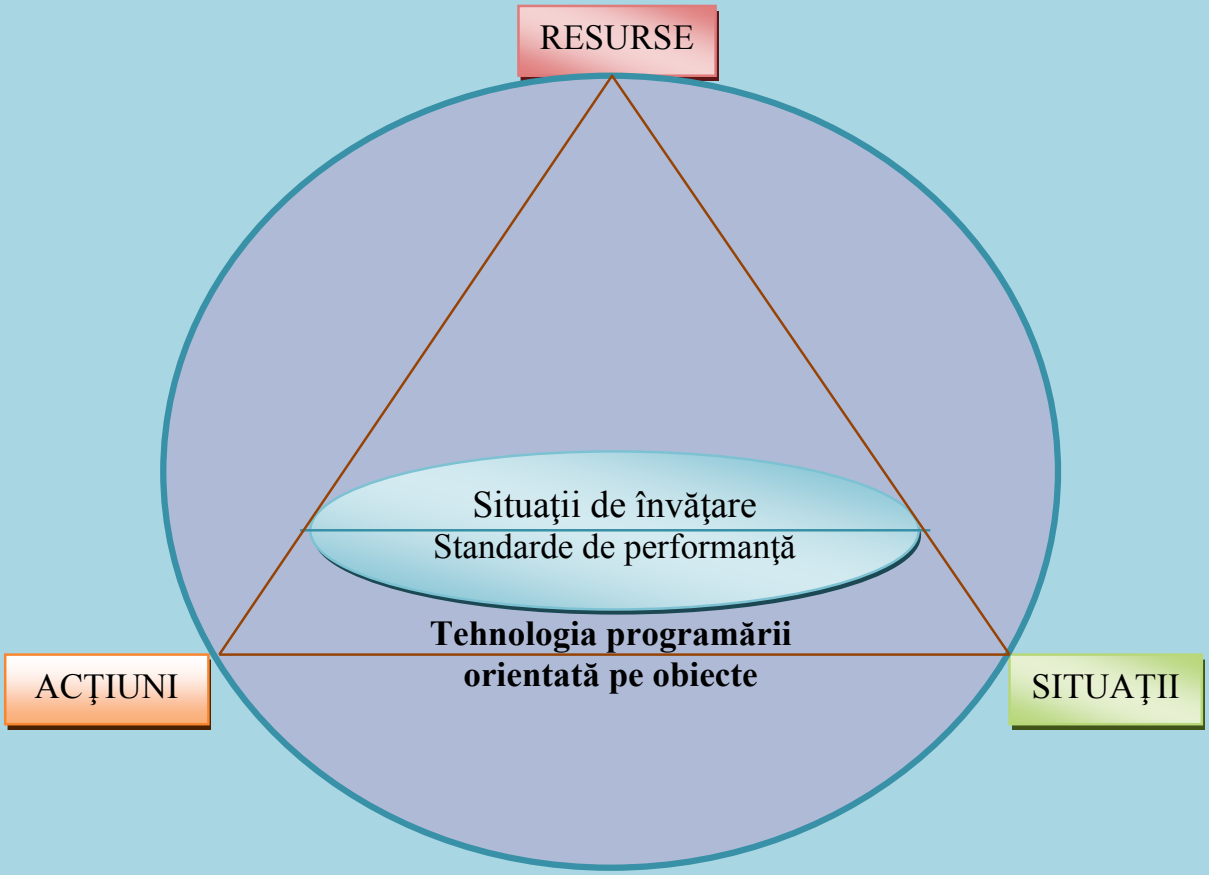
UNIVERSITATEA DE STAT TIRASPOL
Catedra Didactica Matematicii, Fizicii şi Informaticii

Ilie LUPU, Valeriu CABAC, **Silviu GÎNCU**

FORMAREA ŞI DEZVOLTAREA COMPETENŢEI DE PROGRAMARE ORIENTATĂ PE OBIECTE LA VIITORII PROFESORI DE INFORMATICĂ



Chişinău, 2013

UNIVERSITATEA DE STAT DIN TIRASPOL

Catedra Didactica Matematicii, Fizicii şi Informaticii

MONOGRAFIE

ILIE LUPU, VALERIU CABAC, SILVIU GÎNCU

FORMAREA ŞI DEZVOLTAREA COMPETENŢEI DE PROGRAMARE ORIENTATĂ PE OBIECTE LA VIITORII PROFESORI DE INFORMATICĂ

CHIŞINĂU, 2013

CZU 37.016.046:004
L 95

Colectivul de autori: Ilie LUPU, prof. univ., dr. hab. în pedagogie (Universitatea de Stat Tiraspol), Valeriu CABAC, dr. conf. (Universitatea de Stat "Alecu Russo" din Bălţi), Silviu GÎNCU, doctorand (Universitatea de Stat Tiraspol).

Aprobat pentru tipar de Senatul Universităţii de Stat Tiraspol

Descrierea CIP A CAMEREI NAŢIONALE A CĂRŢII

Lupu, Ilie.

Formarea şi dezvoltarea competenţei de programare orientată pe obiecte la viitorii profesori de informatică: Situaţii de învăţare. Standarte de performanţă: Tehnologia programării orientată pe obiecte / Ilie Lupu, Valeriu Cabac, Silviu Gîncu; Univ. de Stat Tiraspol, Catedra Didactica Matematicii, Fizicii şi Informaticii. – Chişinău : UST, 2013. – 150 p.

100 ex.

ISBN 978-9975-76-100-0.

Recenzenţi:

Anatol GREMALSCHI,
doctor habilitat, profesor universitar,
Universitatea Tehnică din Moldova
Liubov ZASTÎNCEANU,
doctor în pedagogie, conferenţiar universitar interimar,
Universitatea de Stat "Alecu Russo" din Bălţi

ISBN 978-9975-76-100-0

CUPRINS

INTRODUCERE

Apariţia şi dezvoltarea tehnicii de calcul a generat o adevărată revoluţie în societatea umană. Calculatorul a devenit un instrument obişnuit de lucru, iar tehnologiile de prelucrare a informaţiei au produs transformări în întreaga societate, pătrunzând în toate aspectele vieţii economice, sociale şi culturale. Aceste transformări se datorează în mare măsură aplicaţiilor prin intermediul cărora utilizatorului i se oferă posibilitatea de a transmite/primi informaţia către/de la maşina de calcul.

Sub aspect educaţional, învăţământul superior necesită a fi modernizat şi racordat la sistemul de formarea şi dezvoltare a competenţelor, menţionăm că în treapta preuniversitară, confom curricumului naţional elevilor le sunt formate competenţe transversale şi competenţe specifice pentru fiecare disciplină. În acest sens viitorul cadru didactic, pentru a forma competenţe, mai întâi trebuie ca el să le posede. Acesta este încă un argument în favoarea modernizării învăţământului superior.

Aşa cum informatica şi tehnologiile informaţionale reprezintă domenii cu cele mai înalte ritmuri de dezvoltare, este necesar de a asigura viitorul cadru didactic cu un şir de competenţe specifice în diferite domenii ale informaticii. Considerăm că formarea, la viitoarele cadre didactice, a competenţei de programare orientată pe obiecte va contribui esenţial la edificarea unei societăţi, capabile să facă faţă ritmului avansat de dezvoltare a tehnologiilor informaţionale.

Monografia conține principalele rezultate ale cercetării la tema: Formarea şi dezvoltarea competenţei de programare orientată pe obiecte la viitorii profesori de informatică.

Capitolul 1 „Aspecte teoretice în formarea şi dezvoltarea competenţei de programare orientată pe obiecte” conţine analiza publicaţiilor ştiinţifice referitoare la modalitatea de organizare a procesului didactic în abordarea prin competenţe, cât şi la modalităţile de formare a viitorilor specialişti, în special, a profesorilor de informatică. Sunt prezentate o serie de modele de predare a tehnologiei orientată pe obiecte studenţilor de la diferite specializări ale informaticii (tehnice, pedagogice ş.a.).

În Capitolul 2 „Bazele teoretico-metodice de formare şi dezvoltare a competenţei de programare orientată pe obiecte” este descris modelul de formare şi dezvoltare a competenţei de programare orientată pe obiecte la viitorii profesori de informatică, elaborat în cadrul cercetării. Conform acestui model, conţinuturile sunt structurate în şase unităţi de învăţare, iar studiul lor se realizează prin aplicarea diverselor strategii didactice, printre care: instruirea în bază de proiecte, studiul de caz, decompoziţia etc. Este descrisă metodologia utilizării modelului, conform căreia formarea şi dezvoltarea competenţei de programare orientaă pe obiecte se va realiza prin utilizarea a două mijloace: limbajul de programare orientat pe obiecte şi mediul de programare vizuală şi o gamă variată de situaţii specifice fiecărei unităţi de învăţare.

În ultimul capitol „Argumentarea experimentală a eficienţei aplicării modelului şi metodologiei elaborate” sunt descrise condiţiile de desfăşurare a experimentului pedagogic şi rezultatele obţinute. Este prezentat un model de organizarea a orelor practice (seminare şi laboratoare) şi o gamă variată de probleme care necesită a fi rezolvate de către studenţi.

Lucrarea este adresată cercetătorilor din domeniul didacticii, doctoranzilor şi masteranzilor de la specialităţile informatice, profesorilor de informatică.

Autorii
Chişinău-Bălţi
August 2013

1. ASPECTE TEORETICE ÎN FORMAREA ŞI DEZVOLTAREA COMPETENŢEI DE PROGRAMARE ORIENTATĂ PE OBIECTE

1.1. Bazele psihico-pedagogice de formare şi dezvoltare a competenţelor

Abordarea prin competenţe (APC) a procesului de învăţământ este o practică implementată în sistemele educaţionale naţionale, care tind să alinieze finalităţile învăţării la cerinţele pieţei de muncă. Într-un şir de ţări (Belgia francofonă, Canada, Franţa, Elveţia, Cili ş. a.) APC este implementată graţie unor reglementări legale. În sistemul de învăţământ din Republica Moldova APC a devenit oficială odată cu elaborarea şi implementarea curriculumului modernizat în anul 2010, însă implementarea APC s-a dovedit a fi anevoioasă. Cauza principală rezidă în faptul că însăşi noţiunea de competenţă nu s-a cristalizat, având semnificaţii diferite în diferite ţări şi la diferiţi autori. Afară de aceasta, în curriculum-ul modernizat din Republica Moldova a mai fost introdusă o noţiune ambiguă: noţiunea de *subcompetenţă*, care nu este definită şi care complică enorm proiectarea şi realizarea procesului de instruire în şcoală.

Termenul de „*competenţă*” provine din latinescul *competere* – a se întâlni, a fi capabil, a fi în stare. Din punct de vedere istoric, termenul a fost introdus în 1965 de către lingvistul N. Chomsky şi definit drept „aptitudinea de a produce şi înţelege un număr infinit de enunţuri, reguli, principii, acţiuni, moduri sau modele practice de a se comporta, strategii preferenţiale sau stiluri productive în profesie” [117].

Orientarea sistemului de învăţământ spre formarea de competenţe s-a produs treptat, urmând mai multe etape:

- Prima etapă este considerată perioada anilor 1960-1970, când termenul de competenţă este introdus în vocabularul pedagogic;
- A două etapă o reprezintă perioada anilor 1970-1990. În această perioadă noţiunea de competenţă este utilizată în deosebi în lingvistică, comunicare, totodată apare noţiunea de competenţă socială (J. Raven, А. К. Маркова, R. W. White);
- Începând cu anii 1990, este introdus termenul de competenţă profesională. Tot mai mulţi cercetători (А. Б. Хуторской, F.-M. Gerard,

X. Roegiers etc.) utilizează noțiunea de competenţa drept sinonim al profesionalismului.

În literatura de specialitate termenul de competenţă este întâlnit cu trei sensuri diferite:

a) *savoir-faire* disciplinar sau *competenţă disciplinară* - se are în vedere efectuarea în scris a unei operaţii aritmetice, construirea unui cerc cu ajutorul compasului etc.;
b) *savoir-faire* general (B. Rey, M. Romainville) - presupune argumentarea, exprimarea orală şi scrisă, verificarea etc.;
c) competenţa este considerată drept o contextualizare a achiziţiilor (cunoştinţe, priceperi şi deprinderi). Această abordare o întâlnim la autorii Ph. Perrenoud, G. Le Boterf, J.-M. De Ketele, X. Roegiers.

S. Marcus menţionează „pentru orice domeniu, competenţa reprezintă condiţia ce asigură eficienţa activităţii umane, iar exersarea eficace a activităţii stă la baza comportamentului şi este condiţionată de o serie de însuşiri caracteristice întregii structuri de interioritate a individului” [144, p. 16].

În literatura de specialitate numărul definiţiilor noţiunii de competenţă aproape este egal cu numărul autorilor. Aducem în continuare cele mai reprezentative, în opinia noastră, definiţii.

A. R. Gherghinescu defineşte competenţa drept „rezultantă a cunoştinţelor, deprinderilor, priceperilor, aptitudinilor şi trăsăturilor temperamental – caracterologice de care dispune individul în vederea îndeplinirii funcţiei sociale cu care este investit” [22, p.17];
B. În viziunea lui C. Cucoş competenţa reprezintă „ansambluri structurate de cunoştinţe şi deprinderi dobândite prin învăţare; apar ca structuri operante cu ajutorul cărora se pot rezolva, în contexte diverse, probleme caracteristice unui anumit domeniu” [19, p. 202];
C. Copilu consideră că competenţa se obţine în rezultatul „interacţiunii a 3C: Competenţa = Cunoştinţe, Capacităţi, Comportament (C=CCC)” [14, p. 154];
D. După M. Ştefan competenţa reprezintă „un sistem de cunoştinţe, abilităţi, deprinderi şi atitudini, bine structurate şi temeinic însuşite, care asigură elevului posibilitatea de a identifica şi de a rezolva în mod eficient problemele dintr-un anumit domeniu” [56, p. 57];

În baza definiţiilor expuse (A-D) competenţa poate fi reprezentată grafic (fig. 1.1) sub forma a trei vectori: *cunoştinţe*, *capacităţi* şi *atitudini*.

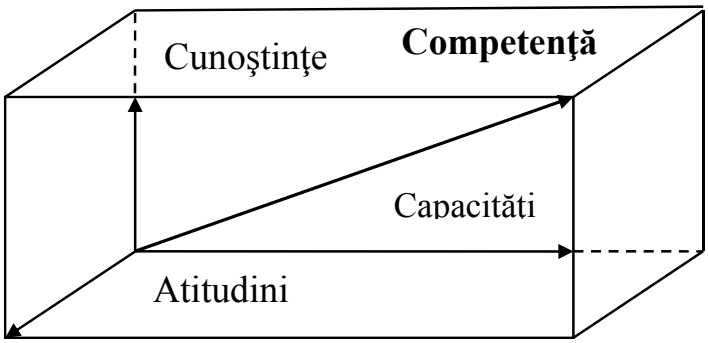


Fig.1.1. Reprezentarea grafică a competenţei.

Cunoştinţele reprezintă totalitatea noţiunilor, ideilor, informaţiilor pe care le are cineva într-un domeniu oarecare.

Capacităţile se definesc ca „însuşiri individuale, care oferă posibilitatea reuşitei, într-un anumit domeniu de activitate" [16, p.159]. Capacităţile se constituie din operaţiunile mentale, mecanismele de gândire ale unui individ, atunci când acesta îşi exersează inteligenţa.

Atitudinile sunt modalităţi de raportare comportamentală şi afectivă la diferite aspecte ale realităţii. Ele sunt raporturi semnificative ale fiinţei umane faţă de fenomenele lumii şi înglobează cunoştinţele şi capacităţile studentului.

O altă abordare a competenţei o întâlnim la Ph. Jonnaert, O. Lebedev, X. Roegiers, V. Cabac etc. care, la definirea termenului de competenţă, utilizează trei elemente indispensabile: **situaţii, resurse**, **acţiuni**. Astfel competenţa este definită:

1. Din punctul de vedere al ***situaţiei*** în care este plasat individul şi care urmează a fi rezolvată/tratată. Situaţia reprezintă circumstanţele în care se află individul şi este sursa competenţei: *numai în situaţie persoana îşi poate demonstra competenţa*. Concomitent, situaţia este criteriul competenţei: persoana este declarată competentă numai dacă ea a tratat cu succes situaţia. Legătura dintre situaţie şi competenţă este într-atât de strânsă, încât competenţa, indirect, poate fi definită printr-un ansamblu de situaţii, care de deosebesc nesemnificativ una de alta. Ph. Jonnaert afirmă: „competenţa şi *situaţia* se imbrică într-o relaţie dialectică: o competenţă se exercită numaidecât într-o *situaţie*, deci

competenţa este situată aşa precum şi *resursele* interne pe care le angajează, şi *resursele* externe pe care le utilizează.” [143].

2. Din punct de vedere al ***resurselor*** ce necesită a fi mobilizate. Anume mobilizarea resurselor necesare presupune competenţa. Resursele pot fi de două tipuri: interne (cunoştinţe, capacităţi, etc.) şi externe (un manual, un sit, un produs software etc.). D. Masciotra menţionează: „o resursă este resursă dacă şi numai dacă persoana care dispune de resursă dată, o poate utiliza şi această resursă este un mijloc efectiv în ameliorarea situaţiei” [144]. X. Roegiers defineşte competenţa „un ansamblu integrat de *resurse* mobilizat în vederea rezolvării unei *situaţii* semnificative care aparţine unui ansamblu de *situaţii* – problemă” [52, p.44];
3. Din punct de vedere al ***acţiunilor*** ce necesită a fi întreprinse. În acest caz se are în vedere acţiunea unei persoane. O astfel de abordare o întâlnim la autorii germani şi ruşi. De exemplu, germanul R. Arnold susţine: „competenţa se referă la capacitatea unei persoane de *a acţiona*” [148, p. 10]. O. Lebedev defineşte competenţa drept “capacitatea de *a acţiona* în *situaţie* de incertitudine” [97].

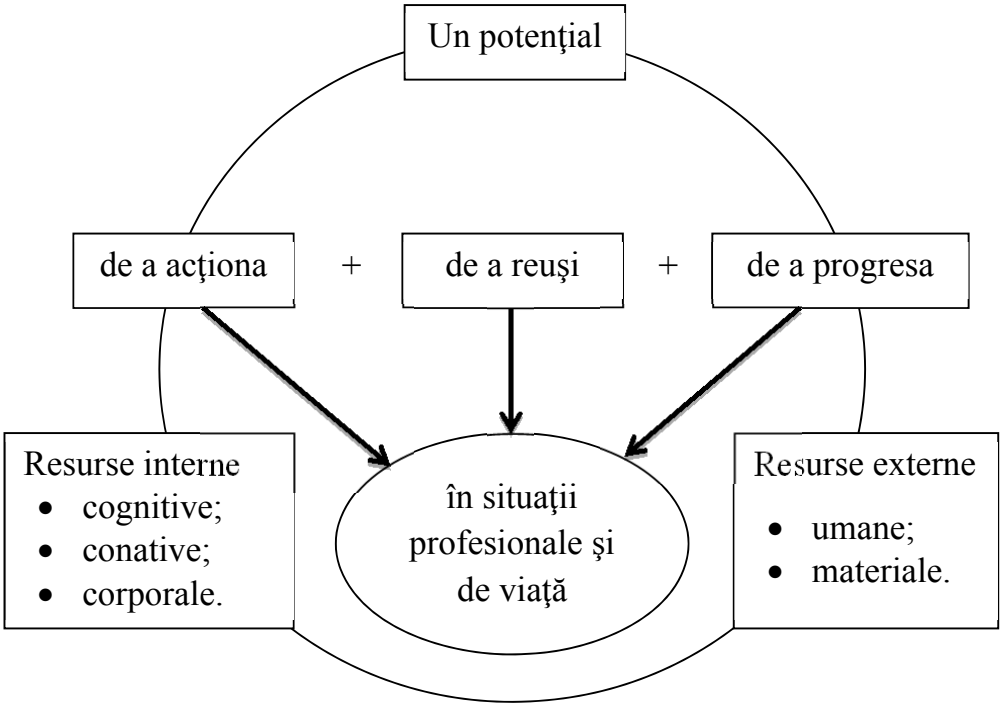


Fig.1.2. Competenţa ca potenţial de acţiune în baza resurselor [20, p. 8]

V. Cabac oferă o definiţie „mai completă” a noţiunii de competenţă, ea fiind definită ca „o structură dinamică, formată în rezultatul învăţării, activităţii profesionale şi practicii trăite, care organizează activitatea unei persoane, plasate într-o *situaţie*, într-un context determinat, prin alegerea, mobilizarea şi coordonarea unui ansamblu diversificat de *resurse* pentru *tratarea* reuşită a *situaţiei*” [9, p.133].

În viziunea N. Deinego competenţa este un *potenţial de acţiune* a unei persoane. Această acţiune constă în realizarea unei *sarcini* complexe, prin mobilizarea *resurselor* disponibile în diverse *situaţii:*

Din perspectiva cercetării noastre considerăm că:

A. persoana îşi poate demonstra competenţa numai în *situaţie*;
B. pentru demonstrarea competenţei persoana va apela la un şir de *resurse*;
C. competenţa presupune *acţiune*.

În acest context, ***competenţa*** *reprezintă un ansamblu de resurse bine integrate prin intermediul cărora individul va putea acţiona pentru soluţionarea/tratarea anumitor situaţii specifice*. În cadrul cercetării noastre noțiunea de competență va fi utilizată în sensul definiției de mai sus..

Învăţământul axat pe formarea de competenţe

Conform J. Bianka „*Educaţia pe bază de competenţe* tinde să fie o formă a educaţiei care elaborează curriculum-ul pe baza analizei unui potenţial sau actual rol în societatea modernă şi care încearcă să certifice progresul cursantului pornind de la performanţa dovedită în unele sau în toate aspectele acelui rol. Creează oportunităţi pentru cursanţi, legate de experienţa lor într-un mediu de învăţare propice (care este de preferat practicii profesionale), prin care cursantul poate dezvolta aptitudini integrate, orientate spre performanţă, pentru a rezolva probleme în practică” [2, p.7].

M. Miled consideră că abordarea învăţământului prin formarea de competenţe are la bază mai multe principii: „(a) definirea şi dezvoltarea competenţelor pentru o inserţie socio-profesională adecvată sau pentru formarea unor capacităţi mentale utile în diverse situaţii; (b) Integrarea cunoştinţelor în loc de achiziţia lor separată; (c) orientarea învăţării spre sarcini complexe; (d) selectarea unor situaţii motivatoare şi stimulatoare pentru student pentru a face cunoştinţele semnificative şi operatorii; (e)

realizarea evaluării într-un mod explicit, prin propunerea unor sarcini complexe" [146, p.130-131].

După introducerea conceptului de competenţă în educaţie, acesta s-a dovedit a fi prea elastic, creând în unele situaţii anumite confuzii. M. Bocoş identifică şase elemente comune ale conceptului de competenţă: „(1) Competenţele sunt dependente de context; (2) Competenţele sunt indivizibile; (3) Competenţele sunt expuse la schimbare; (4) Competenţele sunt racordate obiectivelor, activităţilor şi sarcinilor; (5) Competenţele necesită procese de învăţare şi de dezvoltare bine definite; (6) Competenţele sunt interdependente." [3, p.12].

X. Roegiers constată 5 indicatori ai competenţei: „(1) *Mobilizarea unui ansamblu de resurse* se manifestă prin specificarea cunoştinţelor, capacităţilor necesare pentru formarea competenţei; (1) *Caracterul finalizat* se manifestă prin modul în care vor fi mobilizate resursele; (2) *Relaţia cu un ansamblu de situaţii* se manifestă prin faptul că mobilizarea resurselor are loc în cadrul unui anumit ansamblu de situaţii; (3) *Caracterul disciplinar* se manifestă prin aceea că competenţa este adeseori definită printr-o categorie de situaţii specifice disciplinei; (4) *Evaluabilitatea*: competenţa poate fi evaluată prin calitatea îndeplinirii sarcinii" [155, p. 68-70].

Competenţele pot fi clasificate după mai multe criterii. După criteriul de *transferabilitate* deosebim competenţe *generale* sau generice şi competenţele *specifice*. Competenţele generale pot fi transferate dintr-un domeniu în altul, dintr-o disciplină şcolară în alta, iar competenţele specifice pot fi dezvoltate şi sunt *tipice* unui domeniu/discipline şcolare. Din această cauză competenţele generale se mai numesc *transversale* - ele sunt comune pentru toate sau aproape pentru toate disciplinele şcolare. Din numărul competenţelor generale Consiliul Europei a identificat un şir de competenţe considerate strict necesare viitorilor cetăţeni europeni. Aceste competenţe au primit denumirea de competenţe-cheie.

Competenţele cheie-transversale „reprezintă un pachet transferabil şi multifuncţional de cunoştinţe, deprinderi/abilităţi şi atitudini de care au nevoie toţi indivizii pentru împlinirea şi dezvoltarea personală, pentru incluziune socială şi inserţie profesională. Acestea trebuie dezvoltate până la finalizarea educaţiei obligatorii şi trebuie să acţioneze ca un fundament

pentru învăţarea în continuare, ca parte a învăţării pe parcursul întregii vieţi" [174, p. 11].

În raportul de progres al grupului de lucru B (2003), au fost stabilite 8 tipuri de competenţe. Curriculumul modernizat din Republica Moldova prevede formarea la elevi a 10 competenţe-cheie care au fost definite pe baza competenţelor-cheie propuse de Comisia Europeană:

1. Competenţe de învăţare / de a învăţa să înveţi;
2. Competenţe de comunicare în limba maternă / limba de stat;
3. Competenţe de comunicare într-o limbă străină;
4. Competenţe acţional-strategice;
5. Competenţe de autocunoaştere şi autorealizare;
6. Competenţe interpersonale, civice, morale;
7. Competenţe de bază în matematică, ştiinţe şi tehnologie.
8. Competenţe digitale în domeniul tehnologiilor informaţionale şi comunicaţionale (TIC);
9. Competenţe culturale, interculturale (de a recepta şi a crea valori);
10. Competenţe antreprenoriale.

Din competenţele-cheie derivă *competenţele specifice* care reprezintă detalieri pentru fiecare obiect de studiu al celor 10 competenţe-cheie. În treapta liceală, acestea se formează pe durata unei trepte de învăţământ.

În treapta universitară accentul este pus pe formarea competenţelor profesionale. Prin **competenţă profesională** se înţelege capacitatea dovedită de a selecta, combina şi utiliza adecvat cunoştinţe, abilităţi şi alte achiziţii (valori şi atitudini), în vederea rezolvării cu succes a unei anumite categorii de situaţii de muncă sau de învăţare, circumscrise profesiei respective, în condiţii de eficacitate şi eficienţă. Competenţele profesionale reprezintă un ansamblu integrat şi dinamic de cunoştinţe (cunoaştere, înţelegere şi utilizare a limbajului specific, explicare şi interpretare) şi abilităţi (aplicare, transfer şi rezolvare de probleme, reflecţie critică şi constructivă, creativitate şi inovare).

Noțiunea de competență completează noțiunea de calificare (certificată printr-o diplomă), care mult timp a servit drept bază pentru angajare în câmpul muncii. "O *calificare* este dobândită atunci când un organism abilitat constată că nivelul de învățare la care a ajuns o persoană a atins un anumit standard al capacităților de cunoaștere, deprinderilor și competențelor

generale. Standardul rezultatelor învățării este confirmat prin intermediul unui proces de evaluare sau prin finalizarea cu succes a unui program de studiu. Învățarea și evaluarea în vederea obținerii unei calificări se poate realiza printr-un program de studiu și/sau prin experiența la locul de muncă. O calificare conferă recunoașterea oficială a valorii rezultatelor învățării pentru piața muncii, precum și pentru educația și formarea profesională continuă. O calificare conferă un drept legal de a practica o ocupație/meserie/profesie" [72, p. 13].

Fiecare calificare este dezvoltată într-un anumit ciclu de studii (licenţă, masterat, doctorat) şi se exprimă prin:

- *Competenţe profesionale generale*, care se dezvoltă în cadrul mai larg al domeniului de studii;
- *Competenţe profesionale specifice*, care se dezvoltă în cadrul mai restrâns al unui program de studii. Se mai întâlnesc sub denumirea de competenţe de specialitate.

Din punctul de vedere al competenţei profesionale putem menţiona un șir de concepte.

Б. С. Гершунский constată: „Categoria competenţelor profesionale este definită de nivelul de pregătire, de experienţa acumulată, de abilităţile individuale, şi nu în ultimul rând, de dorinţa fiecăruia de a se autodepăşi" [83, p. 74].

Э. Ф. Зеер, О. Н. Шахматова constată ca prin „competenţă profesională se are în vedere o combinaţie de cunoştinţe şi aptitudini profesionale, precum şi modul în care sunt desfăşurate activităţile profesionale" [91, p. 46].

Competenţa de specialitate a cadrului didactic face referire la trei capacităţi principale:

- Cunoaşterea materiei;
- Capacitatea de a stabili legături între teorie şi practică;
- Capacitatea de înnoire a conţinuturilor, în consens cu noile achiziţii ale ştiinţei domeniului (dar şi din domenii adiacente).

Competenţa de specialitate se manifestă în trei aspecte: teoretic, operaţional, creator.

Tabelul 1.1. Aspecte ale competenţei de specialitate.

Aspectul teoretic	*Aspectul operaţional*	*Aspectul creator*
- de a asimila conţinutul ştiinţific propriu disciplinelor de învăţământ predate şi a metodelor, tehnicilor de informare; - de a realiza corelaţii inter/ intra şi pluridisciplinare ale conţinuturilor; - de a actualiza, prelucra, esenţializa, ilustra, reprezenta şi dezvolta conţinutul; - de a surprinde valenţele formative şi educative ale conţinutului.	- de a structura asimilarea conţinuturilor astfel încât să dezvolte structuri operatorii, afective motivaţionale, volitive, atitudinale; - de a dirija asimilarea tehnicilor de activitate intelectuală odată cu informaţiile; - de a forma modul de gândire specific disciplinei respective de învăţământ şi modul de gândire sistemic; - de a valoriza conţinutul obiectului de învăţământ, structurând comportamente raportate la valori; - de a comunica fluent, expresiv, coerent.	- de a adapta conţinuturile specificului dezvoltării psihice stadiale a educabililor; - de a stimula dezvoltarea maximală a potenţialului fiecărui educabil, prin asimilarea conţinuturilor; - de a promova învăţarea participativă, anticipativă, societală, creatoare; de a dirija surprinderea problemelor şi dezvoltarea lor; - de a dezvolta conţinuturile şi strategiile de asimilare.

Modele de formare a competenţelor

În literatura de specialitate există mai multe modele de formare şi dezvoltare a competenţelor. Vom analiza, în continuare, câteva dintre modelele existente.

În viziunea lui C. Chauvigné şi D. Vandroz formarea competenţei profesionale presupune parcurgerea mai multor etape pe parcursul cărora

studentul dezvoltă competenţe la diferite nivele: „*Tehnician* – presupune a utiliza metode şi tehnici cunoscute; *Inginer* – posibilitatea de a oferi răspunsuri noi, în funcţie de situaţie; *Expert* - capacităţi metacognitive de reflecţie asupra practicii sale pentru a concepe acţiuni originale” [116, p. 2]. Acest concept reprezintă mai mult un nivel de calificare, decât un model de formare.

Alt concept de formare a competenţelor se bazează pe logica construcţiei. R. Wittorski scoate în evidenţă cinci modalităţi de dezvoltare a lor [141, p. 29]:

a) Dezvoltarea bazată pe *logica integrării* - presupune o învăţare realizată până la începutul exercitării activităţii profesionale;
b) Dezvoltarea bazată pe *logica acţiunii* - presupune o învăţare de tip aplicativ în situaţii profesionale, bazată pe metoda încercărilor şi erorilor într-un context unic. Acest model este caracteristic învăţământului medicinal, de exemplu, recoltarea sângelui; în timpul studiilor studenţii nu efectuează această operaţie în practică, această abilitate se dezvoltă abia după finalizarea studiilor.
c) Dezvoltarea bazată pe *logica acţiunii generative* - învăţarea este realizată în situaţii profesionale; această modalitate se bazează pe metoda încercărilor şi erorilor în diferite contexte.
d) Dezvoltarea bazată pe *logica reflecţiei asupra acţiunii* – învăţarea este realizată după exersarea activităţii profesionale prin formalizare, explicitare şi analiza practicii. Acest model este caracteristic situaţiilor când la nivel practic o operaţie poate fi efectuată, dar în acelaşi timp ea este efectuată fără un suport teoretic.
e) Dezvoltarea bazată pe *logica reflecţiei până la acţiune* - presupune o învăţare realizată înaintea exersării unei activităţi profesionale prin reflecţii anticipative asupra schimbărilor ce pot interveni în activitate.

Un grup de cercetători din Belgia (X. Rogiers, J.-M. De Ketele, F.-M. Gérard) [155] a elaborat modelul moderat de dezvoltare a competențelor. Acest model este elaborat în baza abordării integrative. Modelul include:

1. *Structurarea resurselor*. Această etapă presupune asimilarea tuturor resurselor necesare pentru soluţionarea/tratarea unei anumite situaţii.
2. *Integrarea resurselor*. Odată formate, resursele necesită a fi integrate în situaţii concrete.

3. *Transfer sau adaptarea* la situaţii noi. În cadrul acestei etape sunt propuse situaţii care diferă, dar nu considerabil de situaţiile din cadrul etapei a doua. Astfel, dacă situaţia este rezolvată/tratată, atunci transferul este realizat. Prin urmare, se poate menţiona că studentul îşi poate manifesta competenţa în anumite situaţii.

În viziunea N. Deinego dezvoltarea competenţelor se realizează prin parcurgerea a cinci etape: „(1) explorarea – conştientizarea şi analiza familiei de situaţii în care va fi exersată competenţa; (2) învăţarea de bază – completarea resurselor, structurarea conţinuturilor; (3) integrarea – antrenamentul; (4) transferul – adaptarea competenţei la situaţii noi din aceiaşi familie de situaţii; (5) îmbogăţirea – dezvoltarea de mai departe a competenţei" [20, p. 18].

Deşi modelul propus de către N. Deinego vine să completeze modelul moderat, el nu poate fi aplicat integral pentru formarea şi dezvoltarea competenţei de POO. Deci modelul moderat urmează a fi analizat şi adaptat astfel, încât să poată fi utilizat în cadrul cercetării noastre.

1.2. Necesităţile sistemului educaţional în dezvoltarea competenţei de programare orientată pe obiecte

Educaţia constituie o componentă indispensabilă în formarea unei societăţi bazată pe cunoaştere. În sens mai larg, ea poate fi concepută, ca o interacţiune dintre individ, societate şi întreaga viaţă socială. Edificarea societăţii bazate pe cunoaștere se realizează prin intermediul sistemului de învăţământ. În Republica Moldova acesta este reglementat prin lege: „un proces organizat de instruire şi educare, prin care persoana atinge un anumit nivel de pregătire fizică, intelectuală şi spirituală, stabilit de stat, şi obţine certificatul respectiv de studii. Sistemul de învăţământ cuprinde reţeaua instituţiilor de învăţământ de diverse tipuri şi forme de proprietate, programele de studii, tehnologiile şi standardele educaţionale de stat de diferite niveluri şi orientări, precum şi organele de conducere a învăţământului, instituţiile şi întreprinderile subordonate acestora" [67]. Conform Art. 12 sistemul de învăţământ este organizat pe niveluri şi trepte:

I Învăţământul preşcolar;

II Învăţământul primar;

III Învăţământul secundar:

1) Învăţământul secundar general:

a) Învăţământul gimnazial;

b) Învăţământul liceal;

2) Învăţământul secundar profesional;

IV Învăţământul mediu de specialitate (colegiu);

V Învăţământul superior;

VI Învăţământul postuniversitar.

Prin intermediul sistemului de învățământ elevului/studentului îi sunt formate un şir de competenţe transversale şi competenţe specifice pentru fiecare disciplină de studiu. Spre deosebire de alte discipline de studiu, conținutul Informatici, în ultimul timp s-a dezvoltat considerabil. Astfel, în cadrul orelor de informatică, activitatea profesorului necesită a fi orientată spre formarea de competenţe la elevi, astfel încât prin dobândirea lor, ei să facă faţă diferitor situaţii, care necesită a fi rezolvate, inclusiv a celor întîlnite în activitatea cotidiană.

Conform Curriculumul modernizat „Informatica participă la formarea şi dezvoltarea generală a personalităţii (dezvoltarea gândirii logice şi algoritmice); formarea de competenţe digitale. Integrarea persoanei în mediul informatizat al societăţii moderne este posibilă numai în cazul deţinerii cunoştinţelor informatice fundamentale şi abilităţilor de utilizare instrumentală şi de comunicare cu calculatorul şi prin intermediul acestuia – totalitatea de competenţe care se conţin în noţiunea de cultură informaţională” [62, p. 4].

Informatica, ca disciplină de studiu, a fost introdusă în diverse țări în perioada anilor 1970-1985. De exemplu, în Franţa şi U.R.S.S disciplina informatica avea drept scop formarea unui stil de gândire operaţional, pe când în Argentina şi S.U.A. era considerată resursă pentru predarea altor discipline, în special a matematicii.

Apariţia unei noi discipline a generat necesitatea de a forma cadre didactice. Conform [41, p. 259] formarea profesorilor de informatică din sistemul de învăţământ din Republica Moldova a fost realizată în mod etapizat, după cum urmează.

Prima etapă s-a realizat în perioada 1985–1987 prin recalificarea profesorilor care predau disciplinele cu profil real, în special matematică şi

fizică. Formarea lor s-a redus la cursuri de scurtă durată (două săptămâni). Din punct de vedere al conţinuturilor, programele de studii prevedeau: familiarizarea cu structura calculatorului; noţiune de algoritm, program; algoritmul de funcţionare a procesorului; instrucţiuni ale unui limbaj de programare (BASIC).

Începând cu anul 1987 în instituţiile de învăţământ din fosta U.R.S.S., inclusiv în cele din R.M. a apărut specialitatea „Informatica" iniţial ca specialitate suplimentară, apoi ca disciplină de bază. Această etapă se caracterizează prin studierea mai multor compartimente precum: tehnica de calcul şi programare; metode de modelare matematică; radioelectronica; sisteme automatizate de dirijare; aplicarea calculatorului în procesul de învăţământ.

În perioada anilor 1992-2005 în instituţiile de învăţământ superior au apărut mai multe specialităţi ce ţin de informatică şi totodată datorită autonomiei universitare, de rând cu disciplinele fundamentale şi disciplinele de specialitate, au fost incluse un set de discipline care asigurau pregătirea psihopedagogică a viitorului profesor de informatică.

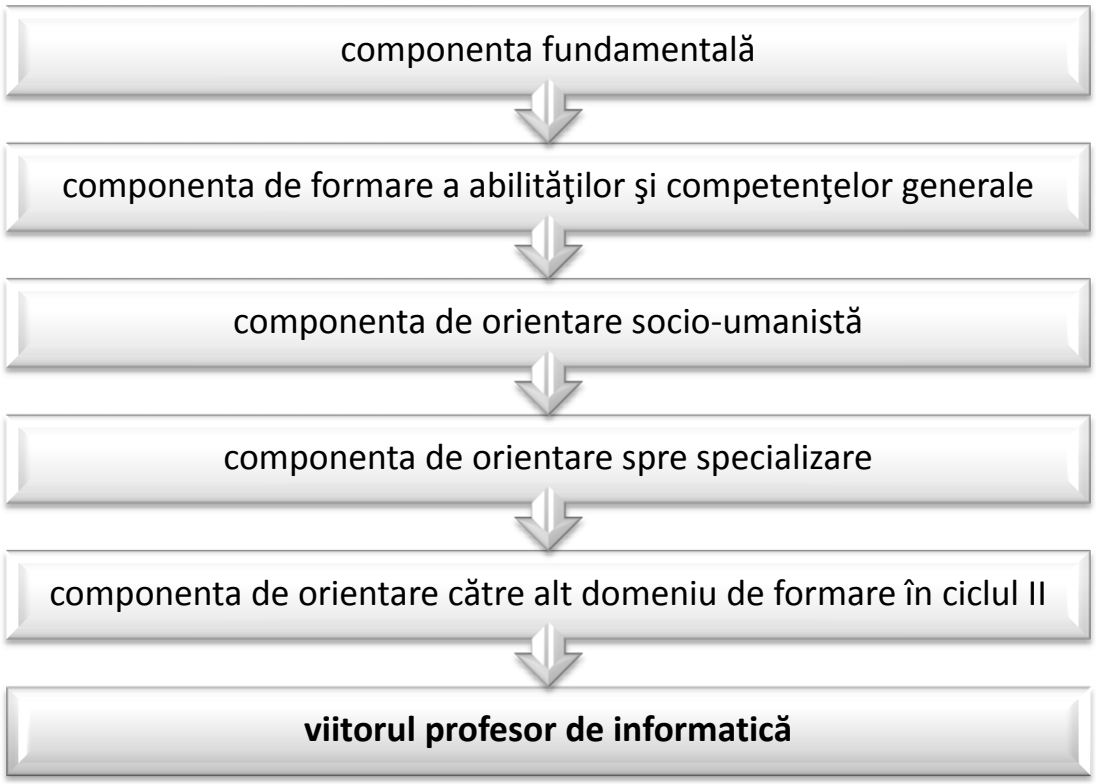


Fig. 1.3. Categorii de discipline care formează viitorul profesor de informatică

A patra etapă, din 2005, este realizată în sistemul cu două cicluri licenţă-masterat. În ambele categorii de planuri disciplinele, ce formează viitorul profesor de informatică, sunt divizate în 5 componente (Fig. 1.3).

Un compartiment important al informatici ţine de procesarea informaţiei. Aceasta se realizează conform schemei:

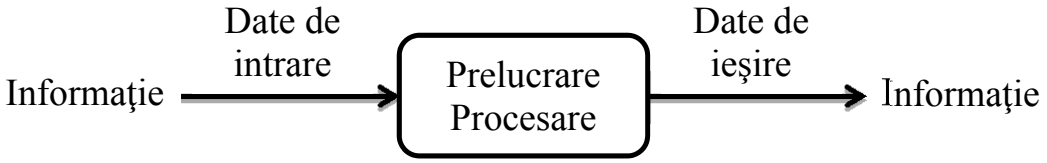


Fig. 1.4. Schema de procesare a informaţiei.

Informaţia este prelucrată prin intermediul programelor. Eficienţa lor, depinde, în mare măsură, de modul de organizare a datelor de intrare. Printre primele paradigme de organizare a codului unui program a fost paradigma *programării structurate,* apărută după anul 1970. Această paradigmă are la bază *teorema de structură*, furnizată de C. Boehm şi G. Jacopini „orice algoritm, având o intrare şi o ieşire, poate fi reprezentat ca o combinaţie de trei structuri de control: *secvenţa* – succesiune de două sau mai multe operaţii; *decizia* – alegerea unei operaţii dintre două alternative posibile; *ciclul* – repetarea unei operaţii atât timp cât o anumită condiţie este îndeplinită" [40, p. 27]. Datorită diversităţii datelor de intrare, în baza acestei paradigme era dificil, aproape imposibil de a elabora programe care să prelucreze un număr relativ mare de date cu o structură diferită. Pentru prelucrarea lor, s-a impus elaborarea unui nou model de programare, orientat pe obiecte, capabil să depăşească limitările programării structurate şi să realizeze o abstractizare adecvată a datelor. Diferenţa dintre programarea structurată şi cea orientată pe obiecte poate fi formulată astfel: „Dacă procedurile şi funcţiile sunt verbe, iar blocurile de date sunt substantive, un program procedural este organizat în jurul verbelor, în timp ce un program orientat pe obiecte este organizat în jurul substantivelor" [112]. Ideea dominantă a programării orientată pe obiecte presupune unirea datelor cu subprogramele care prelucrează aceste date într-un tot întreg, numit *obiect*. De fapt, concepţia de obiect a intervenit pentru crearea unor noi tipuri de date, care nu sunt predefinite într-un limbaj de programare (de exemplu, tipurile de date număr complex şi graf nu sunt definite în majoritatea

limbajelor de programare). La baza acestui model sunt mai multe concepte, care se bazează pe:

a) modelul lumii reale sau pe o parte a acestuia, care poate fi descris ca o totalitate de obiecte care colaborează între ele;
b) obiectul este descris ca:
    1. un set de parametri, valorile cărora determină starea obiectului;
    2. un set de operaţii pe care le poate realiza obiectul;
c) interacţiunea dintre obiecte se realizează printr-un schimb de mesaje, în baza cărora pot fi îndeplinite o serie de acţiuni ori poate fi modificată starea obiectului;
d) obiectele care sunt descrise de acelaşi set de parametri şi pot îndeplini acelaşi set de operaţii, reprezintă o clasă;
e) programul reprezintă un model al lumii reale sau al unei acţiuni exacte din acest model.

Cunoaşterea de către viitorii profesori de informatică a acestei paradigme de programare, este necesară, din mai multe aspecte:

1. Datorită elementelor care stau la baza ei (fig. 1.5).

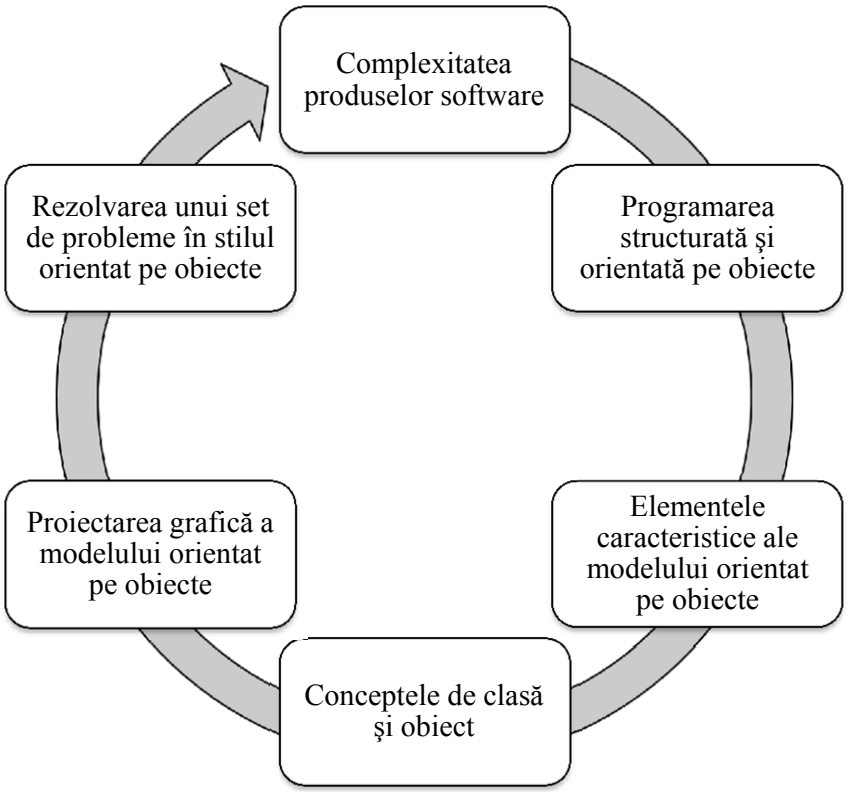


Fig. 1.5. Elementele caracteristice POO.

O problemă, la care oferă răspuns programarea orientată pe obiect (POO), o reprezintă *complexitatea produselor software*. Conform G. Booch, factorii principali care influenţează complexitatea unui produs software sunt: complexitatea domeniul cercetat; impedimentele întâlnite în procesul de elaborare a unei aplicaţii; necesitatea asigurării unei flexibilităţi suficiente a programului; compatibilitatea programului cu alte programe, sisteme. Aceasta permite de a forma la student o nouă concepţie ce ţine de programare. Programarea nu înseamnă numai cunoaşterea unor structuri de date şi a unor algoritmi de prelucrare a lor, dar şi abilitatea de a dezvolta un design mai bun, divizat în mai multe componente aflate într-o conexiune permanentă.

Punctul forte al acestui model de programare constă în faptul că studentului îi sunt aduse argumente precum că problema-cheie în dezvoltarea unui produs software o constituie proiectarea sistemului, reacţia acestuia la solicitările altor programe, cât şi utilizarea adecvată a resurselor tehnice. Modelul orientat pe obiecte oferă un şir de avantaje în soluţionarea problemelor de complexitate sporită şi nu numai. G. Booch [112] prezintă cinci dintre ele:

a. Modelul orientat pe obiecte permite de a opera cu obiecte şi de a utiliza limbaje de POO;
b. Utilizarea modelului orientat pe obiecte mărește fiabilitatea produsului realizat;
c. În modelul orientat pe obiect produsul evoluează treptat, prin completarea acestuia pe parcurs;
d. Se reduce riscul de a dezvolta sisteme complexe;
e. Modelul orientat pe obiecte este conceput după percepţia umană.

Un alt aspect al POO îl reprezintă modul de rezolvare a unei probleme: dacă în *programarea structurată* problema este rezolvată prin integrarea mai multor algoritmi, atunci în cazul *programării orientată pe obiecte* problema este descompusă în mai multe probleme mai mici reprezentate sub formă de obiecte. Avantajul acestei metode îl constituie modul de creare a obiectului/obiectelor. Ele pot fi create de persoane diferite, aflate în diferite locuri, fără a avea o comunicare permanentă, pe când în cazul programării structurate, la rezolvarea unei probleme, numărul de persoane care participă

la rezolvarea ei este limitat. Datorită mecanismelor POO codul programului poate fi reutilizat, reducând considerabil dimensiunea acestuia.

Modelul orientat pe obiecte constă dintr-un set de obiecte definite printr-un set de atribute (date, metode etc.) care comunică datorită relaţiilor dintre ele. La baza acestui model se regăsesc mai multe concepte: abstractizare, incapsulare, modularitate, ierarhizare, tipizare, paralelism, persistenţă dintre care primele patru sunt esenţiale. Le vom explicita în continuare.

Abstractizare - procesul care captează caracteristicile esenţiale ale unui obiect, caracteristici ce diferenţiază acest obiect de toate celelalte tipuri de obiecte. Prin abstractizare se rezolvă problemele cu un grad sporit de complexitate. C. A. R. Hoare susţine: „abstractizarea se manifestă prin determinarea similarităţii dintre anumite obiecte, situaţii sau procese din lumea reală şi în luarea deciziilor, ignorând pentru moment deosebirile dintre obiecte" [120]. E. Seidewitz şi M. Stark menţionează: „există o serie de abstracţiuni de obiecte care pot să corespundă aproape exact realităţilor dintr-un anumit domeniu, iar pentru unele obiecte, acestea nu au dreptul la existenţă" [165]. Aceste tipuri de abstracţii includ: abstractizarea entităţilor obiectului; abstractizarea comportamentului unui obiect; abstractizarea maşinii virtuale; abstractizarea arbitrară.

Incapsulare - procesul mental executat pe o abstractizare în care elementele structurale ale abstractizării sunt izolate de cele ce constituie comportamentul său. Serveşte la separarea interfeţei vizibile a unei abstractizări faţă de implementarea sa.

Modularitate - procesul de divizare a unui sistem complex în părţi (module) manevrabile. Conform cercetării lui B. Liskov „modularitatea nu este alt ceva decât divizarea programului în mai multe unităţi care, fiind compilate separat, comunică între ele" [142];

Ierarhizare - procesul de clasificare pe nivele de abstractizare (moştenire, agregare). Conceptele de *clasă şi obiect* sunt esenţiale în programarea orientată pe obiecte. Ele se află într-o strânsă legătură: dacă un obiect reprezintă o entitate, atunci prin intermediul clasei sunt definite abstracţiunile entităţii.

Abstracţiunile reprezintă:

a) variabile (date) prin intermediul cărora se păstrează informaţia despre un obiect (de exemplu, obiectul *student* va conţine date referitoare la nume, prenume, anul naşterii etc.);
b) funcţii (metode) de prelucrare a datelor.

Obiectul este o instanţă a unei clase. În baza unei clase pot fi create mai multe obiecte cu aceeaşi structură. Următoarea situaţie prezintă un exemplu de clasă şi obiecte:

```
class dreptunghi{
double lung,lat;
public: void citire();//pentru citirea datelor
void afisare();//pentru afisare
double arie();
double perimetru();
void init(double a, double b){lung=a;lat=b;}
};
```

În clasa *dreptunghi* sunt declarate date pentru păstrarea informaţiei despre dreptunghi şi metode de prelucrare a datelor. Prin intermediul declaraţiilor de tipul: *dreptunghi a, b*; au fost create două obiecte, cu aceeaşi structură. Ele se diferenţiază doar prin datele fiecăruia dintre ele, de exemplu, instrucţiunea *a.init(10,5)* specifică că obiectul *a* este un dreptunghi cu lungimile laturilor egale cu 10, respectiv 5.

Un alt avantaj al POO îl reprezintă posibilitatea de *proiectare grafică* a modelului orientat pe obiecte. Drept consecință, se obține o prezentare mai clară a materialului teoretic, se oferă posibilitatea de proiectare şi modificare rapidă a structurii proiectului prin existența unei relaţii de asociativitate dintre obiectele din viaţa cotidiană şi cele care fac parte din proiect.

Un rol important în formare îl reprezintă *aplicarea cunoştinţelor teoretice în practică*. Aceasta se realizează prin rezolvarea a cât mai multe probleme în stilul POO. În aşa mod studentului îi sunt formate deprinderi practice şi, totodată, el este confruntat cu situaţii reale, care necesită a fi rezolvate. Utilizarea conceptelor POO permite de a elabora programe cu o dimensiune a codului mai redusă şi de o complexitate mai mică.

2. Odată cu apariţia limbajelor de POO evoluţia produselor software a cunoscut un salt important. Anume în baza acestei concepţii, sunt elaborate o bună parte din ele. Pentru o mai bună pregătire şi creare a unei culturi

informatice a viitorului cadru didactic considerăm necesar de a-i forma acestuia competențe de programare orientată pe obiecte (CPOO).

3. Conform planurilor de studii [68; 69; 70; 71] ale instituţiilor de învăţământ superior din Republica Moldova cursul de „*Programare orientată pe obiecte*" este studiat (în mod direct sau indirect) de către studenţii specialităţii "Informatica" (tab. 1.2).

Tabelul 1.2. Discipline ale ciclului I de studii din perspectiva studierii POO.

Instituţia	Disciplinele conform planului de învăţământ	Nr. de credite
USM	POO	4
UPSC	POO (Delphi)	8
	Programare Java	3
	Baze de date (SQL, Delphi)	4
USB	POO	5
	Programarea aplicaţiilor Visual Basic Programarea aplicaţiilor Visual C /Java	6
UST	Programare 2 (POO în Borland Delphi)	5
	Programare 4 (POO)	5

Prin studierea POO viitorului cadru didactic i se prezintă o nouă metodologie de rezolvare a problemelor. Conform acestei metodologii, informaţia este reprezentată sub formă de obiecte care pot să comunice între ele.

4. O importanţă majoră în realizarea procesului didactic îl constituie *resursele* utilizate. Din acest punct de vedere, profesorul de informatică are posibilitatea de a selecta diverse produse software educaţionale. În viziunea M. Vlada „software educaţional reprezintă orice produs software în orice format ce poate fi utilizat pe orice calculator şi care reprezintă un subiect, o temă, un experiment, o lecţie, un curs etc., fiind o alternativă sau unica soluţie faţă de metodele educaţionale tradiţionale" [60]. Un dezavantaj al acestora îl reprezintă costul lor, care în condiţiile sistemului de învăţământ din Republica Moldova sunt foarte mari. Acesta este unul dintre motivele,

care impune cadrul didactic să elaboreze de sine stătător materiale didactice necesare pentru lecţii. Aplicația MS Power Point este cea mai frecvent utilizată de către profesori la elaborarea materialelor didactice. Deşi utilizarea acestora aduce un plus de calitate în procesul didactic, sunt cazuri când este necesar de a prezenta elevului/studentului exemple *mai speciale*. În acest sens, prin intermediul mediilor de programare vizuală orientate pe obiecte, pot fi elaborate diverse platforme educaţionale, care mai pot fi elaborate şi pentru diferite compartimente (crearea tipurilor dinamice de date: inserarea/ excluderea unui element, metode de sortare etc.).

5. Pregătirea calitativă a cadrului didactic. Actualmente POO este predată studenţilor de la specialitatea ”Informatică” din colegii (tab. 1.3).

Tabelul 1.3.. Predarea POO la specialitatea ”Informatica” în colegii.

Instituţia	Discipline conform planului de învăţământ la specialitatea ”Informatica”	Nr. de ore
Colegiul Politehnic	POO	128
Colegiul de Informatică	POO (Delphi)	128
	Programare Java	128
	Bazele de date (SQL, Delphi)	128
Colegiul Financiar-Bancar	POO I	128
	POO II	256

Pentru a-şi desfăşura activitatea cadrul didactic necesită a fi pregătit. Deseori acesta urmează cursuri de calificare, acestea fiind însă costisitoare. Considerăm că formarea competenţei de POO viitorilor profesori de informatică în cadrul formării inițiale la facultate va veni în întâmpinarea acestora, printr-o calificare mai înaltă.

6. Aşa cum mediile de programare vizuale orientate pe obiecte (în cazul nostru - Borland Delphi) sunt populare, elevii claselor liceale ar putea studia modul de utilizare a acestora în cadrul orelor opţionale.

Prin studierea POO vor fi dezvoltate capacităţile intelectuale ale viitorului profesor de informatică. Cunoaşterea unei asemenea modalităţi de analiză, proiectare a produselor informatice prin metodele POO este vitală în dezvoltarea intelectuală a oricărui student de la specialitatea ”Informatica”. Interpretarea unor conținuturi ale informaticii, din punct de vedere al POO,

este mai simplă şi mai uşor de explicat, De exemplu, lucrul cu fişiere. Asupra fişierelor pot fi efectuate mai multe operaţii *caracteristice* tuturor fişierelor cum ar fi copiere, redenumire, mutare, ştergere etc. şi operaţii specifice cum ar fi formatarea unui document **.doc*, de tip imagine, sau execuţia fişierelor executabile. Dacă am face abstracţie de tipul acestora şi le-am interpreta drept obiecte cu proprietăţi: nume, data creării, dimensiune etc. şi metode precum copiere, formatare etc. abordarea în acest sens ar fi mai simplă. Astfel de exemple pot fi: formatarea textului, lucrul cu directoarele etc.

1.3. Tendinţe de dezvoltare a metodelor de predare-învăţare în cazul programării orientate pe obiecte

În evoluţia sa, omenirea a cunoscut mai multe salturi esenţiale în dezvoltare. O etapă esenţială a fost apariţia calculatorului. Se părea că odată cu apariţia acestuia toate problemele vor fi soluţionate. Însă calculatorul este doar o maşină care primeşte comenzi şi le execută. Astfel apare necesitatea de a găsi un mijloc de comunicare între om şi sistemul de calcul. Acest mijloc a fost programul. Pe baza programelor au fost elaborate alte programe, care la rândul lor „*creau programe*".

Pentru a facilita citirea de mai departe a conţinutului vom prezenta cele mai importante noţiuni din domeniul programării: programare, program, algoritm, limbaj de programare.

Programarea are ca scop realizarea de programe care să constituie soluţiile unor probleme concrete oferite cu ajutorul calculatorului.

Un *program* reprezintă un şir de instrucţiuni-maşină care sunt obţinute în rezultatul compilării unui algoritm descris într-un limbaj de programare (ca şi cod sursă).

Prin *algoritm* se înţelege o mulţime finită de operaţii (instrucţiuni) elementare care executate într-o ordine bine stabilită, pornind de la un set de date de intrare dintr-un domeniu de valori posibile, produce în timp finit un set de date de ieşire (rezultate).

Un *limbaj de programare* reprezintă un set bine definit de expresii şi reguli valide de formulare a instrucţiunilor pentru un sistem de calcul. Prin intermediul lui se oferă posibilitatea programatorului de a specifica în mod

exact şi amănunţit acţiunile pe care trebuie să le execute calculatorul, ordinea acestor acțiuni şi datele care vor fi prelucrate.

Limbajele de programare se clasifică după mai multe categorii. În timp acestea au cunoscut mai multe generaţii.

Prima generaţie a limbajelor de programare a apărut în perioada 1954-1958. Limbajele din această generaţie au fost concepute pentru a simplifica, în special, complexitatea formulelor matematice. Reprezentanți ai acestei generații sunt limbajele: FORTRAN I, ALGOL-58, Flowmatic şi IPL V.

A doua generaţie o constituie limbajele de programare elaborate în perioada anilor 1959-1961. Aceste limbaje se caracterizează prin concentrarea atenţiei asupra abstracţiunilor algoritmice. În perioadă nominalizată calculatoarele au devenit mai puternice, ele fiind utilizate în soluţionarea problemelor de ordin economic, de evidenţă a personalului etc. A doua generație poate fi caracterizată de limbajele: FORTRAN II, ALGOL-60, COBOL, Lisp.

Odată cu apariţia tranzistoarelor, costul unui calculator a scăzut considerabil, astfel producerea calculatoarelor crescând aproape exponenţial. Creşterea numărului de calculatoare, dar şi multiplicarea domeniilor lor de aplicaţie, a condus, în perioada anilor 1962-1971, la apariţia unei noi generaţii de limbaje de programare. Au apărut limbaje precum: PL/I, ALGOL-68, Pascal, Simula. Aceste limbaje propun un alt nivel de abstractizare a datelor, oferind programatorului posibilitatea de a crea noi tipuri de date.

Perioada anilor 1970-1979 se caracterizează prin apariţia mai multor limbaje de programare, care nu au făcut faţă cerinţelor pieţii. Au avut o răspândire mai redusă decât cele din generaţia a treia.

K. Schmucker prezintă evoluţia limbajelor de programare sub formă de arbore [112, p.477]. În baza fig. 1.6 observăm trei direcţii de dezvoltare a limbajelor de programare: limbaje care sunt bazate pe proceduri, limbaje care sunt bazate pe obiecte şi orientate pe obiecte.

G. Booch defineşte programarea orientată pe obiecte drept o „tehnologie concepută în baza principiilor de abstractizare, incapsulare, modularitate, ierarhizare, tipizare, paralelilism şi persistenţă. Ceea ce este important constă în faptul că aceste principii se regăsesc în modelul orientat pe obiecte.” [112, p.25]. Un limbaj de programare se consideră a fi orientat

pe obiecte dacă acesta suportă cel puţin primele patru principii (abstractizare, incapsulare, modularitate, ierarhizare). În caz contrar, dacă acesta susţine cel puţin un principiu, acesta este considerat un limbaj bazat pe obiecte. L. Cardelli şi P. Wegner menţionează faptul că „limbajele de programare se consideră a fi orientate pe obiecte doar atunci când satisfac următoarele condiţii: 1) oferă posibilitatea de a opera cu obiecte, care sunt abstracţii cu interfaţă comună, iar accesul la elementele acestora este protejat la anumite nivele de protecţie; 2) obiectele sunt instanţe ale diferitor clase; 3) clasele pot moşteni atributele altor clase” [115].

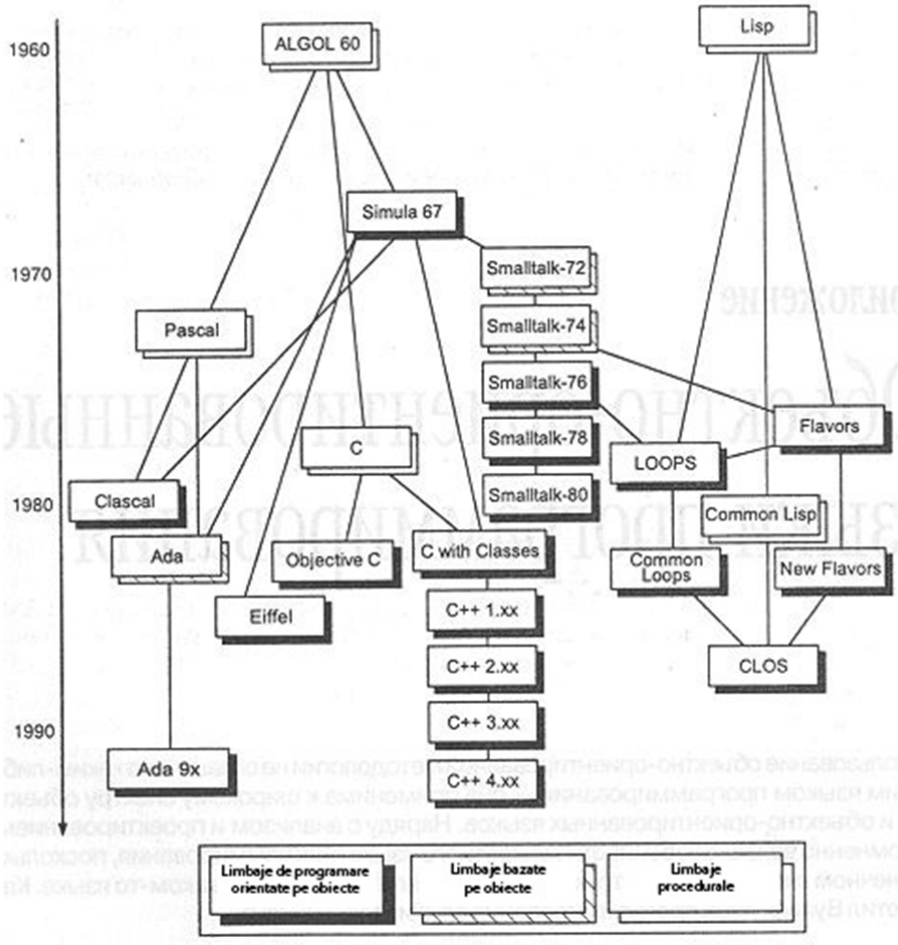


Fig. 1.6. Evoluţia limbajelor bazate pe obiecte şi orientate pe obiecte.

Concomitent cu dezvoltarea limbajelor de programare au apărut şi noi stiluri de organizare a codului sursă a unui program. În timp s-au evidenţiat următoarele stiluri:

1. *Programarea nestructurată* care este intuitivă şi nu presupune nici o organizare a codului. Acest stil este folosit doar de către programatorii începători. Toate datele şi instrucţiunile se află în programul principal într-o unitate de compilare (un fişier sursă, din care, prin compilare, va rezulta un fişier obiect);
2. *Programarea structurată* ce presupune organizarea codului unui program în secvenţe bine structurate la nivel logic şi sintactic. Variante ale acestui stil sunt:
    2.a. *Programarea procedurală* care reprezintă organizarea codului în proceduri şi funcţii, care sunt porţiuni identificabile de cod, cu o funcţionalitate bine precizată;
    2.b. *Programarea modulară* ce oferă posibilitatea de grupare a procedurilor cu funcţionalităţi asemănătoare (sau strâns corelate) în acelaşi fişier, numit unitate de compilare. Astfel programul este împărţit în mai multe fişiere, care ulterior pot fi compilate separat;
3. *Programarea orientată pe obiecte* POO "presupune unirea datelor cu subprogramele care prelucrează aceste date într-un tot întreg, numit obiect" [4, p.148]. Acest stil de scriere a codului, numit tehnologie, oferă posibilităţi de modelare a obiectelor, a proprietăţilor şi a relaţiilor dintre ele, dar şi posibilitatea de a descompune o problemă în componentele sale.

Modelul orientat pe obiecte este unul complex, format din mai multe etape, cum ar fi etape de analiză, de proiectare şi de programare. V. Arnaut defineşte aceste etape, după cum urmează:

A. *Analiza orientată pe obiecte* (AOO) „o metodologie în cadrul căreia cerinţele faţă de sistem sunt concepute din perspectiva claselor şi a obiectelor evidenţiate din domeniul problemei" [59, p.10];
B. *Designul orientat pe obiecte* (DOO) „o metodologie care reuneşte în sine procesul de decompoziţie şi metodele de reprezentare a modelului logic şi fizic, static şi dinamic al sistemului proiectat" [59, p.10];
C. *Programarea orientată pe obiecte* (POO) „o metodologie de programare, legată prin intermediul unui set de obiecte, fiecare din ele fiind un exemplar al unei anumite clase, iar clasele formează o ierarhie de moştenire" [59, p.10].

Odată cu apariţia modelului orientat pe obiecte a apărut necesitatea de a utiliza noi metode de predare-învăţare, aceasta deoarece termenul de obiect, cuprinde un spectru mai larg de caracteristici decât un tip de date dintr-un limbaj de programare. „Obiectul are stare, comportament şi identitate, structura şi comportamentul obiectului este determinată de clasa în care acesta este descris” [112, p.81]:

Starea obiectului este determinată de proprietăţile acestuia. Prin modificarea lor se schimbă starea obiectului.

Comportamentul – modul în care un obiect acţionează şi reacţionează. Comportamentul obiectului este exprimat în termeni de stare şi de transmitere a mesajelor. Ca structură comportamentul unui obiect este determinat de metodele clasei în baza căreia este creat obiectul.

Identitatea reprezintă o proprietate a obiectului, prin intermediul căreia obiectul poate fi distins de către alte obiecte.

Un grup de obiecte nu prezintă nici un interes, dacă acestea nu colaborează între ele. Un obiect se află într-o relaţie cu alt obiect, ele având posibilitatea de a comunica (a-şi modifica starea şi comportamentul). J. Rumbaugh menţionează că „relaţia dintre obiecte este o conexiune fizică sau conceptuală între ele” [160]. Obiectele se pot afla în relaţii de asociere, agregare, moştenire.

Asocierea presupune colaborarea între obiecte ce aparţin diferitor clase. Sunt trei tipuri de relaţii: unul la unu, unul la mulţi şi mulţi la mulţi. De exemplu: ţară-capitală, ţara are o singură capitală, prin urmare avem o relație ”unul la unu”. Relația ”profesor-disciplină” poate fi o relație de tipul ”unul la unu”, caracteristică pentru învățământul preuniversitar, unde profesorul, de regulă, predă o singură disciplină școlară. În cazul învățământului superior relația de mai sus este, mai degrabă, de tipul ”unul la mulți”, deoarece un profesor predă mai multe discipline.

Agregarea - relaţia între două obiecte în care unul dintre ele aparţine celuilalt obiect. De exemplu student-torent, studentul este parte componentă a unui torent.

Un obiect înglobează date şi operaţii fiind astfel o abstracţiune a unei entităţi din lumea reală. Prin intermediul clasei sunt descrise entităţile obiectului. Colaborarea dintre obiecte este determinată de clasele prin

intermediul cărora acestea sunt create. „Clasa – un set de obiecte cu o structură şi un comportament comun” [112, p. 102].

Moştenirea - mecanismul prin care clasa unui set de obiecte preia structura şi comportamentul acestora în cadrul ei. De exemplu, persoana–angajat; orice angajat este şi persoana, prin urmare structura clasei persoana se regăseşte în structura clasei angajat.

În timp s-au impus mai multe metode de predare-învăţare a POO. Apariţia acestor metode se datorează specificului viitorului specialist în domeniul tehnologiilor informaţionale. Dacă pentru un inginer este important de a cunoaşte structura unui program orientat pe obiecte, atunci pentru un programator este important de a deţine competenţe privind modul de dezvoltare a programului. Astfel, pentru identificarea metodelor de predare învăţare a POO din perspectiva viitorului profesor de informatică a fost analizată literatura de specialitate privind POO [112], [59], [61], [73], [85] etc. şi a planurilor de învăţământ al universităţilor din ţară cât şi de peste hotare (România, Rusia).

În urma cercetărilor au fost identificate patru modele (fig. 1.7) în baza cărora este studiată tehnologia POO.

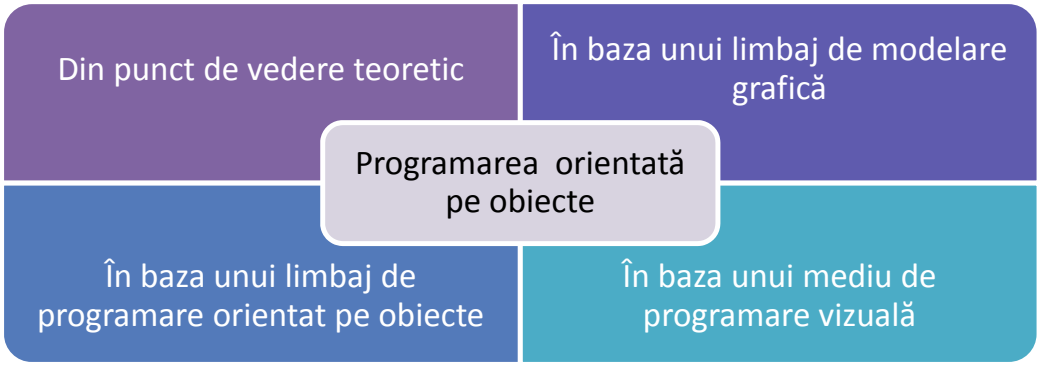


Fig. 1.7. Modele de predare-învăţare a programării orientate pe obiecte.

Indiferent de metoda selectată (mai puţin - prima), este important ca în procesul de studiu, accentul să fie pus pe studierea tehnologiei POO şi nu pe resursele (limbaje, medii de programare etc.) utilizate. Creatorul limbajului de programare C++, B. Stroustrup afirma: „trebuie să se facă distincţia dintre învăţarea de a programa şi învăţarea unui limbaj de programare” [109]. De asemenea, el subliniază că este important să se acorde atenţie la învăţarea metodelor de programare, iar pentru aceasta au fost elaborate limbaje de programare.

Prezentăm modelele identificate în fig. 1.7, din perspectiva APC:

1. Modelul **teoretic** – care presupune studierea principiilor la nivel de noţiuni (definiţii) şi concepte, iar exemplele sunt bazate pe situaţii din lumea reală. Prin intermediul acestei metode poate fi format doar un concept despre programarea orientată pe obiecte, fără a intra în detalii. În mod etapizat această metodă poate fi organizată în felul următor (fig. 1.8):

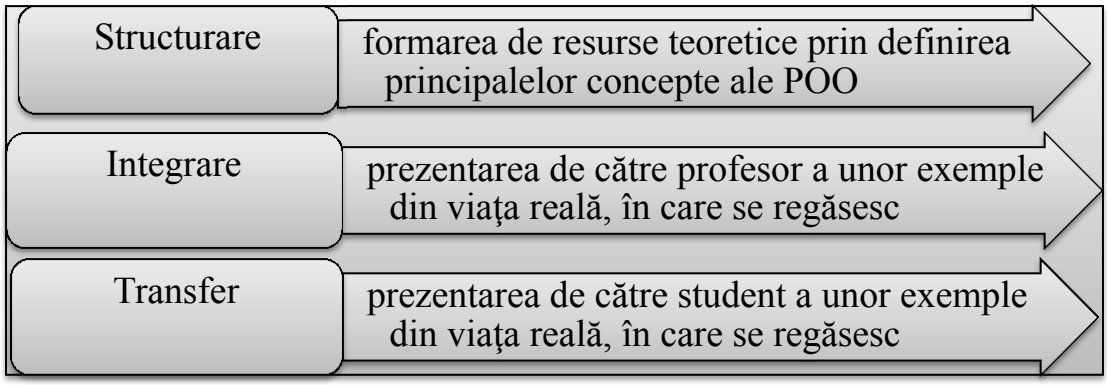


Fig. 1.8. Modelul teoretic de studiere a POO.

Conform modelului teoretic, profesorul este persoana care se expune asupra corectitudinii exemplului prezentat. Din punctul nostru de vedere, considerăm că acesta este un mare dezavantaj. Modelul dat nu poate fi aplicat în cercetarea noastră, datorită modului simplist de tratare a situaţiilor.

2. Formarea de competenţe POO **în baza diferitor metalimbaje**. Dintre metalimbajele cunoscute, în POO s-a impus limbajul UML (Unified Modeling Language). „UML este un limbaj de modelare grafică, care ajută la descrierea şi design-ul sistemelor software, în particular, al sistemelor software construite folosind stilul orientat-obiect” [89, p.27]. Acest limbaj a fost conceput pentru descrierea, specificarea și documentarea funcțiilor unei aplicații în timpul etapei de proiectare. Utilizând UML se face abstracţie de limbajul de programare, accentul fiind pus mai mult pe structura sistemului. UML este fundamental legat de metodologia OO (Object-Oriented) folosită pentru dezvoltarea de produse software. Diagramele UML pot fi utilizate la faza de analiză şi proiectare a unui produs program orientat pe obiecte, prin intermediul cărora se rezolvă două sarcini principale:

a. Identificarea claselor şi obiectelor domeniului cercetat;
b. Creare relaţiilor de interacţiune dintre obiecte conform cerinţelor problemei.

La etapa de proiectare sunt abordate întrebări precum:

I. Ce clase vom avea şi care vor fi relaţiile dintre ele ?
II. În baza căror mecanisme vor fi create relaţiile necesare ?
III. La ce etapă va fi elaborată fiecare clasă ?
IV. Cum trebuie organizată activitatea procesorului, pentru a efectua cât mai multe operaţii ?

Pentru tratarea fiecărei situaţii vor fi utilizate diagrame precum:

1. Diagrame de clase (class diagram) - prin intermediul cărora pot fi create clase şi definite relaţii între ele;
2. Diagrame de obiecte (object diagram) – prin intermediul cărora sunt reprezentate obiectele din sistem şi relaţiile dintre ele;
3. Diagrama componentelor (Component Diagram) – descrie structura fizică a codului în termenii componentelor de cod şi relaţiile dintre acestea;
4. Diagrame de interacţiune (Interaction Diagram) – prin intermediul lor pot fi descrise interacţiunile în timp dintre obiecte, relaţiile şi mesajele care circulă între acestea.

În mod etapizat această metodă se reprezintă în felul următor (fig. 1.9):

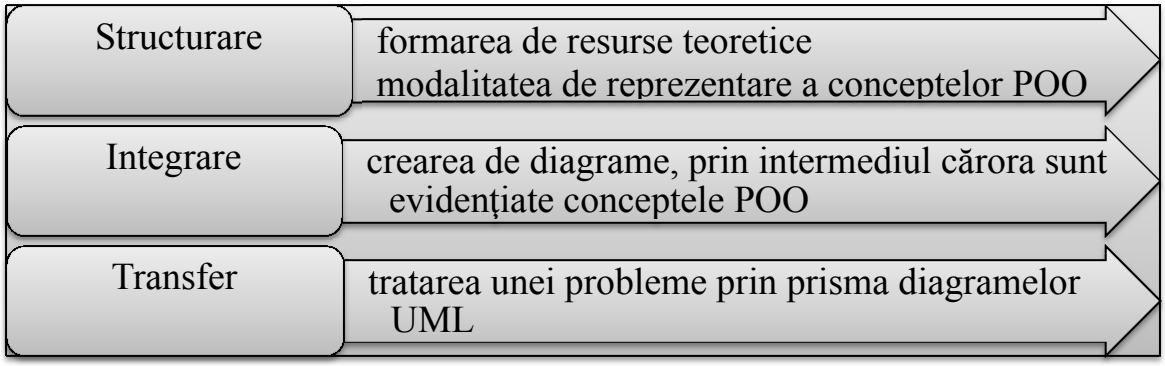


Fig. 1.9. Formarea de competenţe POO în baza unui limbaj de modelare grafică.

Spre deosebire de modelul teoretic, modelul descris oferă mai multe avantaje. În acelaşi timp, în cadrul modelului nu sunt elaborate programe, în care să fie utilizate clasele şi obiectele create. Se recomandă a utiliza acest model pentru pregătirea tehnicienilor din domeniul tehnologiilor informaționale.

3. Formarea de competenţe POO **în baza unui limbaj de programare orientat pe obiecte** (fig. 1.10) - se bazează pe prezentarea conceptelor POO în baza unui limbaj de programare; cele mai frecvent utilizate sunt limbajele

Object Pascal şi C++. De altfel acest model este utilizat şi în predarea cursului de POO la specialităţile informatice ale mai multor instituţii de învăţământ superior din Republica Moldova.

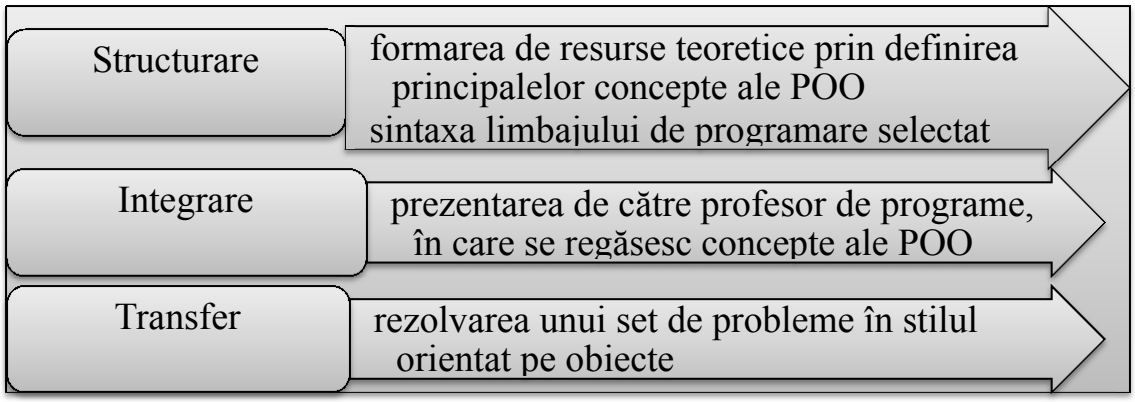


Fig. 1.10. Formarea de competenţe POO în baza unui limbaj de programare

Chiar dacă limbajele de programare oferă un şir de avantaje, cum ar fi: crearea claselor şi relaţiilor dintre ele, definirea metodelor pentru fiecare clasă, rezolvarea unei probleme cu utilizarea de obiecte etc., la etapa actuală formarea de competenţe în baza unui limbaj de programare nu este suficientă, datorită platformelor de dezvoltare existente. Avantajul modelului se caracterizează prin faptul că, dacă sunt selectate adecvat programele rezolvate şi propuse spre rezolvare, la final acestea pot fi incluse într-un program complex, prin intermediul căruia poate fi prezentată o situaţie semnificativă, în care se vor regăsi majoritatea conceptelor studiate.

4. Formarea de competenţe POO (fig. 1.11) **în baza unui mediu de programare vizuală**. În prezent sunt mai multe medii de programare vizuală (tab. 1.4), construite în baza unui limbaj de programare orientate pe obiecte.

Tabelul 1.4. Corelaţia dintre limbaje de programare şi mediile de programare.

Limbaj de programare	*Medii de programare*
Object Pascal	Borland Delphi
C++	C++ Builder, Visual C++
C#, C++, Basic	Platforma Visual Studio

Mediile de programare vizuală se bucură de o popularitate imensă. Din perspectiva formării viitorului profesor de informatică, este esenţial de a forma competenţe, ce ar permite cadrului didactic să elaboreze aplicaţii

Windows orientate pe obiecte. Dacă prin intermediul limbajul de programare accentul este pus, în mare măsură, pe crearea claselor şi a relaţiilor dintre acestea, atunci prin intermediul unui mediu de programare vizuală se oferă posibilitatea de a opera cu clase deja create, instanţele acestor clase fiind componente. Avantajul acestei metode constă în faptul că fiecare mediu de programare vizuală conţine o gamă variată de biblioteci, care, la rândul lor, sunt create din clase. În consecință, pentru prezentarea relaţiilor dintre obiecte pot fi utilizate instanţele claselor existente.

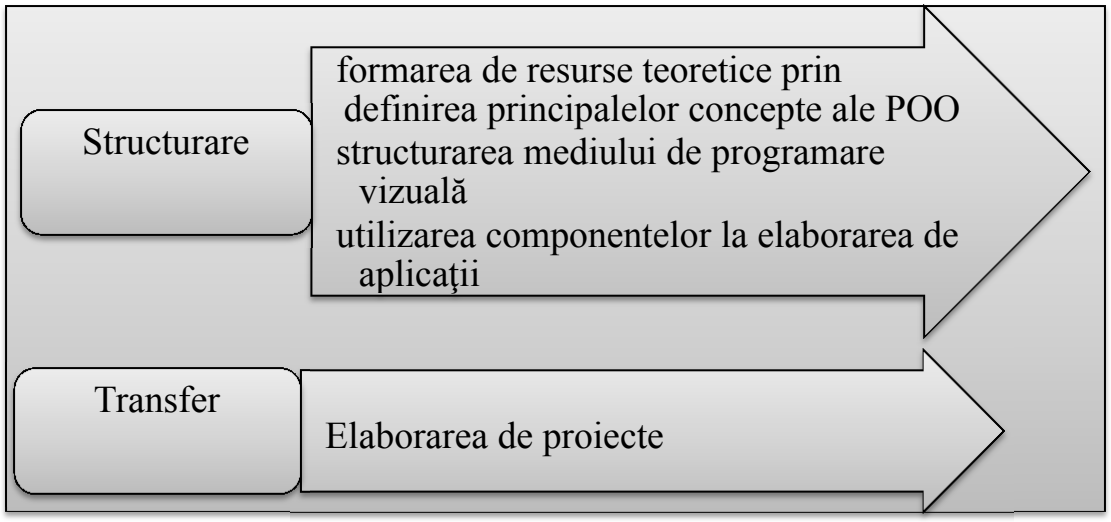


Fig. 1.11. Formarea de competenţe POO în baza unui mediu de programare vizuală

Utilizarea acestui model duce frecvent la studierea mediului de programare vizuală şi a modalităţii de proiectare a interfeţei grafice. În acest caz, în procesul de studiu, accentul nu este pus pe crearea de clase şi a relaţiilor dintre ele, ci pe utilizarea obiectelor existente. Considerăm că viitorul profesor de informatică trebuie să deţină competenţe privind modalitatea de creare a claselor, a relaţiilor dintre ele, cât şi modalităților de rezolvare a unei probleme semnificative, utilizând obiecte.

2. BAZELE TEORETICO-METODICE DE FORMARE ŞI DEZVOLTARE A COMPETENŢEI DE PROGRAMARE ORIENTATĂ PE OBIECTE

2.1. Modelul de formare şi dezvoltare a competenţei de programare orientată pe obiecte

Pentru prima dată noţiunea de model a fost utilizată de către matematicianul E. Beltrami în 1868, referindu-se la modelul Euclidian pentru geometria neeuclidiană. Termenul provine din latinescul „*modus*" ceea ce înseamnă *mijloc*.

Noţiunea de model se referă la un mod de cunoaştere a realităţii care constă în reprezentarea fenomenului studiat cu ajutorul unui sistem construit artificial. În viziunea lui А. И. Богатырев „modelul reprezintă o modalitate artificială de reprezentare a obiectelor sub formă de diagrame, structuri fizice, fenomene, formule prin intermediul cărora sunt scoase în evidenţă anumite proprietăţi şi relaţii dintre obiecte" [95]. Datorită diverselor domenii de utilizare a modelelor (matematică, pedagogie, tehnică etc.), în mod convenţional, modelele se clasifică în: modele *fizice* (similare cu originalul); modele *matematice* (natura lor fizică este diferită de prototip, dar este posibilă descrierea matematică a comportamentului); modele *logice* (construit din caractere speciale, simboluri şi diagrame). Între aceste tipuri de modele nu există limite rigide.

Conform А. Н. Дахин „modelele pedagogice se referă, în principal, la cel de al doilea şi a treilea tip" [87]. Autorul nominalizat consideră că „*modelarea pedagogică* reprezintă o metodă de cercetare orientată într-o anumită direcţie, ce reflectă caracteristicile fenomenelor cercetate ca formă şi ca conţinut" [88, p. 17]. Ea se realizează prin parcurgerea a patru etape: "(1) Analiza domeniului cercetat; (2) Identificarea şi stabilirea unor relaţii dintre elementele care stau la baza modelului, formularea criteriilor în baza cărora va fi elaborat modelul; (3) În baza elementelor identificate se determină setul minim de componente care ar asigura funcţionalitatea modelului; (4) Dezvoltarea dinamică a modelului: 4.a) formularea şi definirea problemei; 4.b) stabilirea regulilor de funcţionare a sistemului, inclusiv parametrii optimi necesari pentru descrierea comportamentului obiectului; 4.c) analiza

posibilității de dezvoltare a modelului; 4.d) stabilirea modului în care va reacţiona modelul la acţiunea unor factori externi; 4.e) descrierea şi analiza condiţiilor de incertitudine a obiectului modelat" [87].

În viziunea noastră, modelarea pedagogică este o activitate creativă şi se realizează conform schemei din fig. 2.1.

Formarea de competenţe presupune asimilarea de resurse prin aplicarea diferitor metode de predare-învăţare, având drept finalitate rezolvarea unui set de situaţii specifice. În cercetarea noastră am optat pentru:

1. selectarea conţinuturilor necesare pentru formarea CPOO) la viitorii profesori de informatică;
2. elaborarea unui model de formare şi dezvoltare a CPOO la viitorii profesori de informatică.

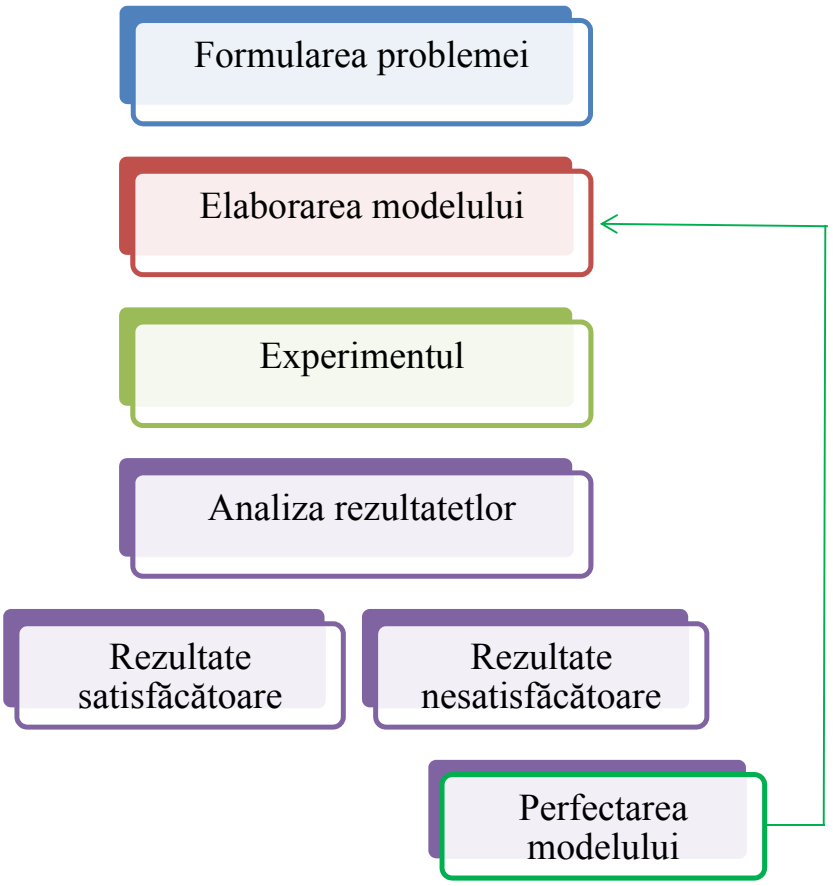


Fig. 2.1. Etapele modelării pedagogice.

Asupra tehnologiei de programare orientată pe obiecte s-au expus mai mulţi cercetători precum: G. Booch [112], J. Rumbaugh [89], И. Н. Аржанов [74], Н. А. Мещерякова [100] etc. POO include în sine mai multe elemente, în mare măsură acestea ţin de conceptele tehnologiei, de modul de

implementare şi capacitatea de utilizare a lor. În viziunea noastră modelul structural al CPOO (fig. 2.2) va conţine:

Resurse - totalitatea noţiunilor, principiilor şi mecanismelor caracteristice tehnologiei POO;

Situaţii – o mulţime de probleme, în rezolvarea cărora vor fi apelate resursele cumulate;

Acţiuni - măsurile ce necesită a fi întreprinse de către student pentru soluţionarea unei situaţii concrete în baza resurselor cumulate.

În baza analizei termenului de competenţă (1.1) au fost evidenţiate trei direcţii de formare a competenţei:

a. pornind de la *resursele* care necesită a fi apelate pentru demonstrarea competenţei;
b. pornind de la *acţiunile* ce necesită a fi întreprinse pentru demonstrarea competenţei;
c. pornind de la *situaţiile* ce necesită a fi rezolvate pentru demonstrarea ei.

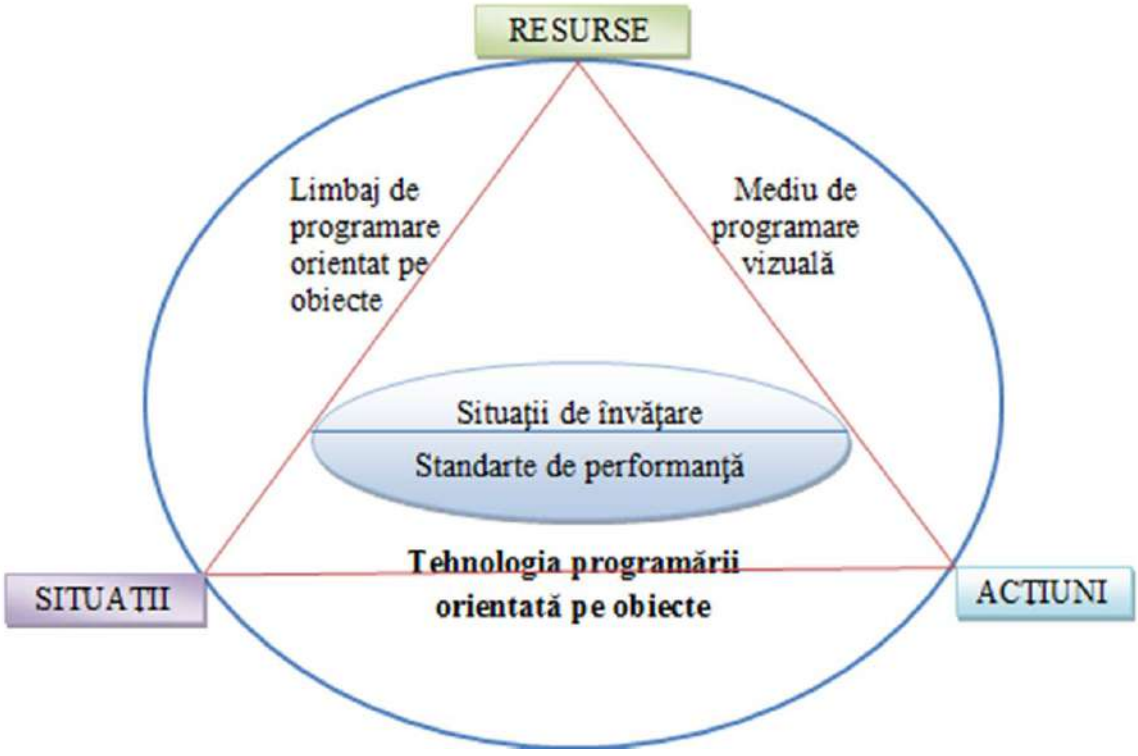


Fig. 2.2. Modelul structural de formare a CPOO.

Cu toate că aceste trei elemente sunt indispensabile (din punct de vedere al competenţei), modelul de formare şi dezvoltare a CPOO urmează a fi elaborat în bază de *situaţii*. Este important să oferim un răspuns referitor la conţinuturile incluse în procesul de formare a CPOO şi a modului de

proiectare al lor. Din punct de vedere al modului de proiectare, competenţa, poate fi reprezentată drept o mulţime, formată dintr-o reuniune de situaţii (fig. 2.3):

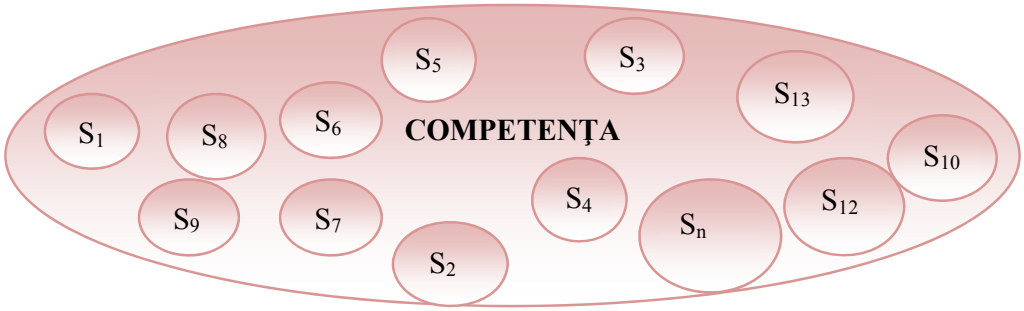


Fig. 2.3. Reprezentarea competenţei sub formă de situaţii specifice

unde S_1, S_2, …, S_n – reprezintă situaţiile specifice POO.

După cum observăm în fig. 2.3, competenţa reprezintă o mulţime de situaţii independente. Formarea ei, după un astfel de model, va permite studentului să fie capabil a rezolva o gamă largă de probleme, însă nu vom putea preciza cu exactitate dacă acesta va putea acţiona competent, într-o anumită situaţie. În [9, p.95] autorii menţionează „noţiunea de *a acţiona competent* se sprijină pe: înţelegerea de către persoană a situaţiei; perceperea de către persoană a scopurilor acţiunilor sale în situaţie; ideea pe care o are persoana referitor la efectul tratării situaţiei; posibilitatea persoanei de a intra în situaţie cu tot bagajul său de cunoştinţe, capacităţi, atitudini; posibilitatea persoanei de a utiliza o pluritate de resurse, de a adapta resursele pe care le cunoaşte sau de a construi resurse noi; posibilitatea persoanei de a reflecta asupra acţiunilor sale, de a le valida şi de a le conceptualiza; posibilitatea persoanei de a adapta tot ce ea a construit în situaţia dată la alte familii de situaţii din aceiaşi familie sau din alte familii de situaţii”.

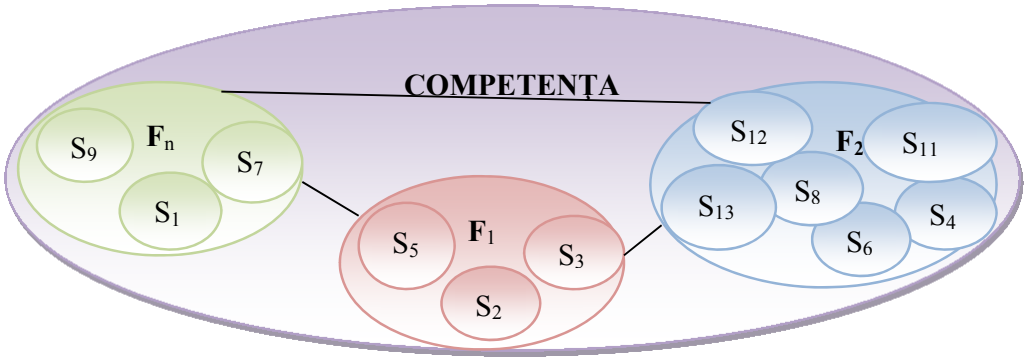


Fig. 2.1.4. Reprezentarea competenţei sub formă de situaţii specifice

unde F_1, F_2, …, F_n reprezintă familii de situaţii specifice POO, iar S_1, S_2, …, S_n reprezintă situaţii specifice ale familiei F_i.

O *familie de situaţii* include în structura sa o serie de probleme (situaţii) pentru un anumit compartiment (unitate de învăţare). Competenţa de POO va fi definită printr-o reuniune de familii de situaţii (fig. 2.4):

În viziunea noastră, este important ca familiile de situaţii să *comunice* între ele. Aceasta va asigura continuitate în procesul de studiu şi va permite profesorului să facă trimitere la ele în diverse cazuri.

Cadrul situaţional		**Tratarea competentă**	
		A acţiona	
Familii de situaţii	*Exemple de situaţii*	*Categorii de acţiuni*	*Exemple de acţiuni*
Familia de situaţii X	Situaţia *a*	Categoria *A*	Acţiunea 1 …… Acţiunea n
	Situaţia *b*	Categoria *B*	Acţiunea 1 …… Acţiunea n
	Situaţia *c* …… Situaţia *n*	Categoria *C* …… Categoria *N*	Acţiunea 1 …… Acţiunea n

Resurse
Resursa 1 …… Resursa n

Fig. 2.5. Matricea acţiunii competente (adaptat după [134])

Din punct de vedere al conţinuturilor este necesar de a descrie elementele competenţei. Cercetătorul canadian Ph. Jonnaert a elaborat un instrument numit **matricea acţiunii competente** (fig. 2.5), prin intermediul căruia se propune fixarea elementelor de descriere a competenţei. Matricea este divizată pe secţiuni [134, p.18-19], şi anume:

A. *cadrul situaţional* este destinat pentru precizarea domeniului de acţiune;

B. tratarea competentă - precizează sensul acţiunii "a acţiona competent";

C. un ansamblu de resurse - servesc drept suport pentru acţiunile descrise.

Matricea permite de a descrie situaţiile specifice pentru formarea şi dezvoltarea CPOO. În cazul nostru vom completa matricea astfel:

Tabelul 2.1. Familiile de situaţii şi categoriile de acţiuni caracteristice CPOO.

Cadrul situaţional		**Tratarea competentă**	
		A acţiona	
Familii de situaţii	*Exemple de situaţii*	*Categorii de acţiuni*	*Exemple de acţiuni*
1. Implementarea conceptelor de *abstractizare* şi *incapsulare*	Crearea unui tip de date care să reunească în structura sa date (variabile) şi funcţii de prelucrare a datelor. Elaborarea de programe cu utilizarea obiectelor create.	Proiectarea structurii unei clase	- Definirea datelor; - Definirea metodelor; - Specificarea nivelului de protecţie a membrilor clasei; - Definirea constructorilor; - Definirea destructorului; - Declararea variabilei de tipul obiect; - Prelucrarea obiectului prin intermediul metodelor.
2. Implementarea conceptului de *moştenire*	Crearea unor ierarhii de clase aflate în relaţie de moştenire simplă sau multiplă. Determinarea structurii claselor aflate în relaţia de moştenire	Stabilirea relaţiilor dintre obiecte într-o ierarhie de clase	- Incapsularea clasei de bază; - Incapsularea clasei derivate; - Abstractizarea obiectelor din ierarhia creată. - Modalitatea de apelare a constructorului în relaţia de moştenire.

3. Implementarea conceptului de *polimorfism*	Definirea polimorfismului dinamic în relaţia de moştenire. Definirea polimorfismului static prin supraîncărcare de operatori.	Crearea unei ierarhii de clase, în care se va regăsi conceptul de polimorfism	- Stabilirea metodelor pentru care poate fi elaborat polimorfismul; - Declararea funcţiilor virtuale; - Redefinirea metodelor; - Definirea datelor în mod corespunzător cu tipul de dată creat; - Supraîncărcarea de operatori cu respectarea proprietăţilor tipului; - Utilizarea în practică a obiectului şi operatorilor supraîncărcaţi.
4. Implementarea conceptului de *agregare*	Crearea şi prelucrarea unei ierarhii de clase aflate în relaţie de agregare.	Definirea metodelor caracteristice ierarhiei	- Incapsularea datelor; - Crearea unei ierarhii de obiecte; - Prelucrarea ierarhiei de obiecte; - Distrugerea ierarhiei create şi eliberarea memoriei.
5. Elaborarea de probleme utilizând *şabloane*	Prelucrarea datelor de tipuri diferite, dar cu algoritmi similari.	Elaborarea de algoritm şablon pentru prelucrarea obiectelor	- Crearea unei funcţii de tip template; - Proiectarea unei clase template; - Utilizarea unui algoritm şablon pentru prelucrarea diferitor tipuri de date.
7. Elaborarea de aplicaţii Windows orientate pe obiecte	Elaborarea de aplicaţii Windows orientată pe obiecte pentru:	Gestionarea de componente	- Identificarea componentelor; - Aranjarea lor pe suprafaţa formei; - Modificarea

	efectuarea calculelor cu numere naturale, reale, complexe; prelucrarea de obiecte aflate în relaţie de moştenire la nivel de componente.		proprietăţilor; - Stabili relaţiile dintre diferite componente; - Prelucrarea evenimentelor.
8. Elaborarea de aplicaţii Windows pentru gestiunea unei baze de date	Elaborarea de aplicaţii Windows orientată pe obiecte pentru gestiunea înregistrărilor unei baze de date.	Modalităţi de intrare/ ieşire a înregistrăril or unei BD.	- selectarea şi editarea componentelor pentru utilizarea tabelelor şi a interogărilor; - selectarea şi editarea componentelor pentru gestiunea înregistrărilor; - selectarea şi editarea componentelor pentru crearea de rapoarte.

Conform instrumentului elaborat de către Ph. Jonnaert pentru soluţionarea competentă a situaţiilor şi acţiunilor proiectate este necesară prezenţa resurselor. Fiecărei familii de situaţii îi sunt caracteristice mai multe resurse (tabelul 2.2).

Tabelul 2.2. Resursele necesare pentru soluţionarea competentă a situaţiilor specifice CPOO.

NR.	**Resursa**
Resursele necesare pentru soluţionarea competentă a situaţiei: **Implementarea conceptelor de *abstractizare* şi *incapsulare***	
1.	Descrierea conceptului *obiect*;
2.	Descrierea conceptului *dată*;
3.	Descrierea conceptului *metodă*;
4.	Descrierea conceptului *constructor*;

5.	Descrierea conceptului *destructor*;
6.	Descrierea conceptului *modificator de acces*;
7.	Descrierea conceptului *clasă*;
8.	Descrierea conceptului *instanţă*;
9.	Descrierea sintaxei unui limbaj de programare orientat pe obiecte;
10.	Capacitatea de a proiecta o clasă;
11.	Capacitatea de a defini domeniile de valori ale unui obiect;
12.	Capacitatea de a defini setul de operatori caracteristici unui obiect;
13.	Capacitatea de a modifica structura obiectului prin intermediul metodelor;
14.	Capacitatea de a proteja datele şi metodele unui obiect;
15.	Capacitatea de a utiliza toate resursele necesare ale unui limbaj de programare pentru implementarea în practică a incapsulării şi abstractizării.
Resursele necesare pentru soluţionarea competentă a situaţiei: **Implementarea conceptului de *moştenire***	
1.	Descrierea conceptului *moştenire simplă*;
2.	Descrierea conceptului *moştenire multiplă*;
3.	Descrierea conceptului *clasă de bază*;
4.	Descrierea conceptului *clasă derivată*;
5.	Descrierea conceptului *moştenire virtuală*;
6.	Descrierea modalităţii de definire a relaţiei moştenire simplă;
7.	Descrierea modalităţii de definire a relaţiei moştenire multiplă;
8.	Descrierea ordinii de apelare a constructorului în relaţia de moştenire;
9.	Descrierea ordinii de apelare a destructorului în relaţia de moştenire;
10.	Descrierea modalităţii de moştenire a datelor şi metodelor;
11.	Capacitatea de a proiecta o ierarhie de clase, bazate pe relaţia de moştenire;
12.	Capacitatea de a proiecta clasa de bază;
13.	Capacitatea de a proiecta clasa derivată;

14.	Capacitatea de a utiliza datele şi metodele clasei de bază în clasa derivată;
15.	Capacitatea de a crea o ierarhie de clase bazate pe relaţia de moştenire.
Resursele necesare pentru soluţionarea competentă a situaţiei: **Implementarea conceptului de *polimorfism***	
1.	Descrierea conceptului *polimorfism, tipuri de polimorfism*;
2.	Descrierea conceptului *supraîncărcare*;
3.	Descrierea conceptului *funcţie membru, funcţie friend*;
4.	Descrierea modalităţii de supraîncărcare a unui operator;
5.	Descrierea conceptului *funcţie virtuală, funcţie virtuală pură*;
6.	Descrierea conceptului *legare târzie*;
7.	Descrierea conceptului *redefinirea metodelor*;
8.	Descrierea modalităţii de definire a polimorfismului dinamic;
9.	Capacitatea de a redefini o metodă în relaţia de moştenire;
10.	Capacitatea de a crea dinamic obiecte într-o relaţie de moştenire;
11.	Capacitatea de a face deosibirea dintre o funcţie virtuală şi una care nu este virtuală;
12.	Capacitatea de a utiliza metode virtuale;
13.	Capacitatea de a supraîncărca operatorii specifici obiectului utilizat;
14.	Capacitatea de a supraîncărca metode;
15.	Capacitatea de a utiliza toate resursele necesare pentru a crea un tip de dată.
Resursele necesare pentru soluţionarea competentă a situaţiei: **Implementarea conceptului de *agregare***	
1.	Descrierea conceptului *ierarhizare*;
2.	Descrierea conceptului *agregare*;
3.	Descrierea modalităţii de definire a relaţiei de agregare dintre două şi mai multe obiecte;
4.	Capacitatea de a incapsula obiecte în structura altor obiecte;
5.	Capacitatea de a proiecta o ierarhie de clase bazate pe relaţia de agregare;

6.	Capacitatea de a proiecta structura claselor din ierarhie;
7.	Capacitatea de a opera cu obiectele din ierarhie;
8.	Capacitatea de a utiliza toate resursele necesare pentru a defini relaţia de agregare.
Resursele necesare pentru soluţionarea competentă a situaţiei: **Elaborarea de probleme utilizând *şabloane***	
1.	Descrierea conceptului *şablon*;
2.	Descrierea conceptului *funcţie template*;
3.	Descrierea conceptului *clasă template*;
4.	Descrierea modalităţii de definire a unei funcţii template;
5.	Descrierea modalităţii de proiectare a unei clase template;
6.	Capacitatea de a utiliza o funcţie template;
7.	Capacitatea de a utiliza o clasă template;
8.	Capacitatea de a utiliza toate resursele necesare pentru scrierea unui algoritm şablon.
Resursele necesare pentru soluţionarea competentă a situaţiei: **Elaborarea de aplicaţii Windows orientate pe obiecte**	
1.	Descrierea noţiunii de *componentă*;
2.	Descrierea noţiunii de *proprietate*, *metodă*, *eveniment*;
3.	Descrierea funcţionalităţii componentelor studiate;
4.	Descrierea modalităţii de crearea a unei aplicaţii vizuale orientată pe obiecte;
5.	Descrierea proprietăţilor esenţiale ale unei componente;
6.	Descrierea evenimentelor esenţiale ale unei componente;
7.	Capacitatea de a modifica proprietăţile unei componente;
8.	Capacitatea de a apela metodele unei componente;
9.	Capacitatea de a prelucra evenimentele unei componente;
10.	Capacitatea de a gestiona proiecte POO;
11.	Capacitatea de a elabora aplicaţii ce vor conţine mai multe componente;
12.	Capacitatea de a proiecta o aplicaţie grafică POO.

Resursele necesare pentru soluţionarea competentă a situaţiei: **Elaborarea de aplicaţii Windows pentru gestiunea unei baze de date**	
1.	Descrierea structurii componentelor pentru lucru cu tabele şi interogări;
2.	Descrierea metodelor pentru tabele şi interogări;
3.	Descrierea metodelor pentru componentele destinate lucrului cu tabele şi interogări;
4.	Capacitatea de a realiza conexiunea dintre o componentă şi o sursă (tabel, interogare);
5.	Capacitatea de a afişa înregistrările dintr-o sursă;
6.	Capacitatea de a selecta componente necesare prelucrării datelor;
7.	Capacitatea de a crea un raport;
8.	Capacitatea de a elabora formulare pentru gestiunea datelor;

În acest context vom defini competenţa de programare orientată pe obiecte după cum urmează:

Definiţie *CPOO reprezintă un sistem de cunoştinţe, abilităţi, deprinderi şi atitudini, numite resurse, bine structurate şi temeinic însuşite, prin intermediul cărora individul va putea acţiona, în baza conceptelor POO, pentru soluţionarea/tratarea anumitor situaţii specifice*.

Urmând aceste viziuni asupra termenului de competenţă, în cadrul cercetării a fost elaborat modelul de formare/dezvoltare a CPOO (fig. 2.6), care se deosebeşte de alte modele existente prin:

a. *Conţinuturile selectate*. În cadrul modelului se regăsesc elemente de conţinut care ţi de modalităţi de proiectare a unei clase şi posibilitatea de a gestiona cu obiectele claselor create.

b. Parcurgerea a cinci etape: *familiarizare, structurare, integrare, transfer şi extindere*, în cadrul unei unităţi de învăţare.

c. *Structurarea conţinuturilor* în şase unităţi de învăţare, care permit a forma studentului noi competenţe şi a dezvolta competenţele deja formate.

d. *Strategiile didactice* utilizate se axează pe dezvoltarea unei gândiri orientată pe obiecte a studentului în baza aplicării *decompoziţiei pe obiecte*, efectuării a unui *studiu de caz*, a *instruirii în bază de proiecte* etc.

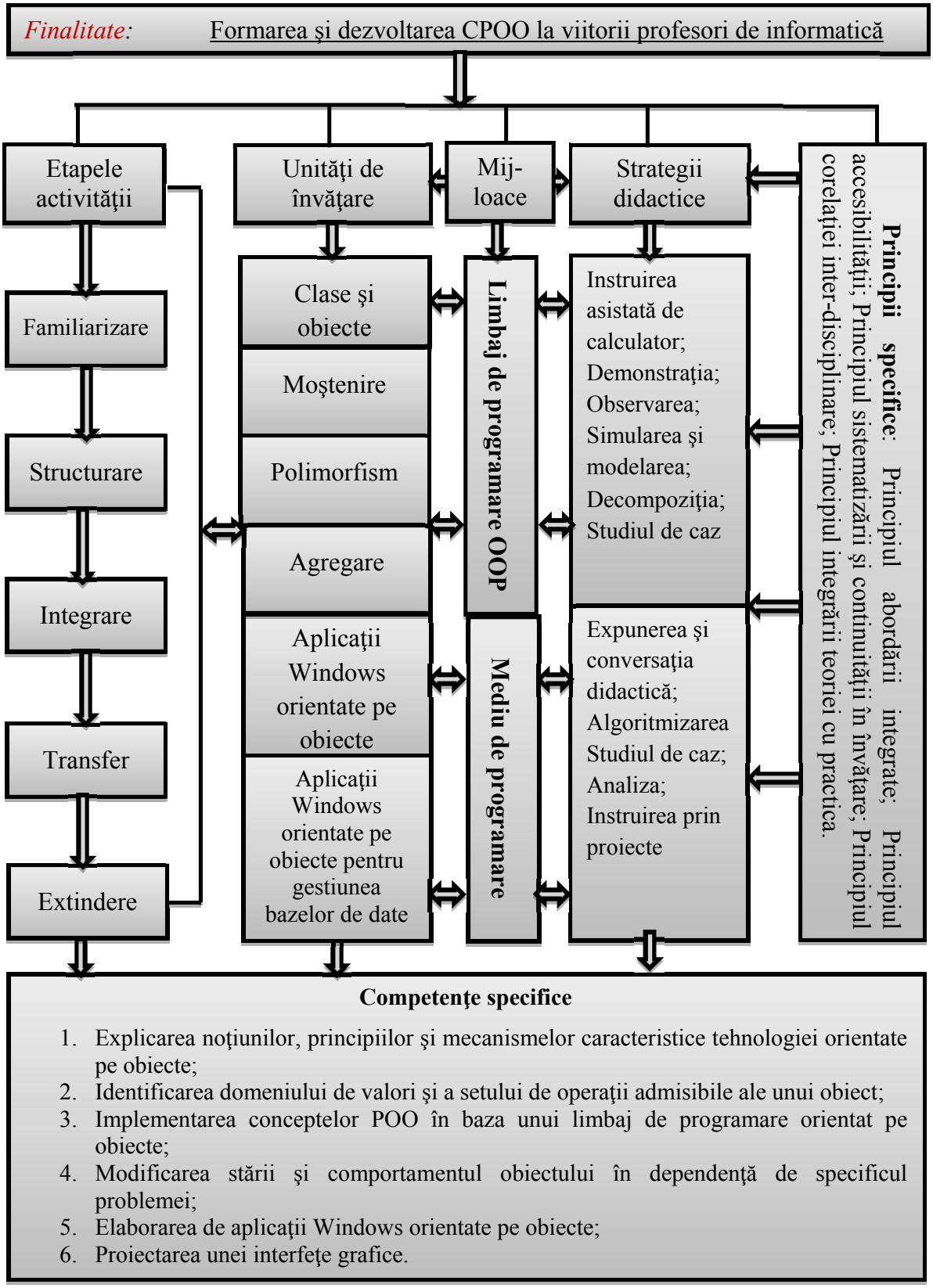


Fig. 2.1.6. Modelul de formare şi dezvoltare a CPOO

Modelul pedagogic de formare şi dezvoltare a competenţei de programare orientată pe obiecte este elaborat în baza mai multor principii specifice, dintre care cinci sunt esenţiale cercetării noastre (fig. 2.7).

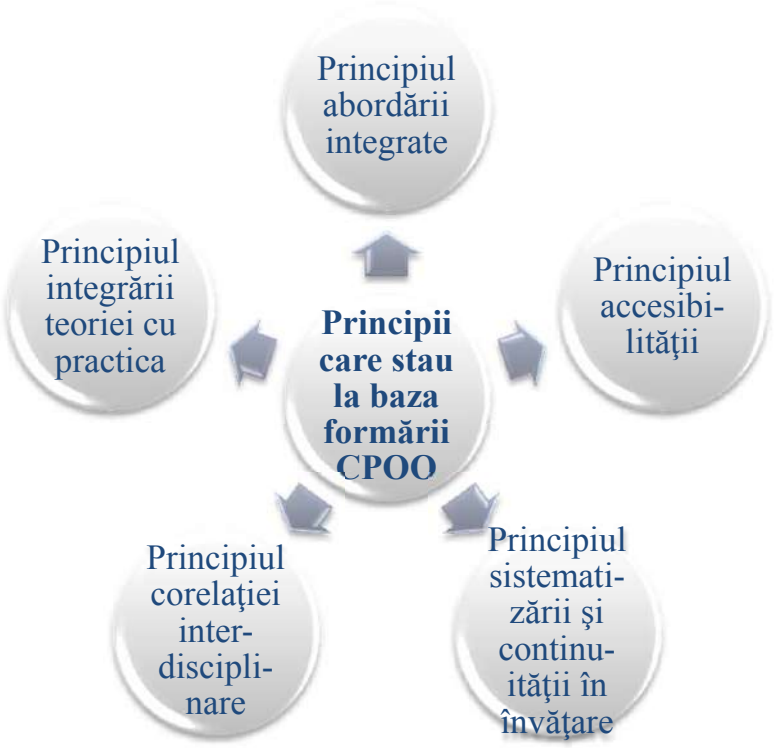
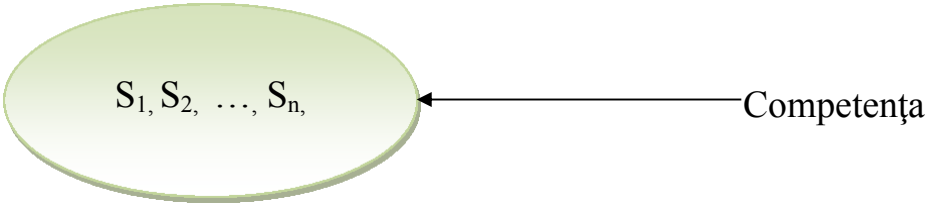


Fig.2.7. Principii de formare a CPOO.

Principiul abordării integrate. Conform acestui principiu conţinutul cursului va fi structurat într-un model integrat, format dintr-o mulţime de situaţii specifice programării orientate pe obiecte, bazate pe principalele principii ale POO, cât şi a relaţiilor dintre obiecte. Soluţionarea fiecărei situaţii va duce în mod implicit la atingerea scopului final. Conform acestui principiu structurarea conţinuturilor se va efectua în modul următor:



unde S_1, S_1, …, S_n reprezintă ansamblul de situaţii specifice.

Principiul accesibilităţii. La structurarea conţinuturilor se va ţine cont de capacităţile studenţilor, în special, de nivelul lor de pregătire. Accesibilitatea depinde de capacităţile studentului şi de complexitatea

sarcinilor. În general, tehnologia POO este predestinată pentru soluţionarea problemelor cu o complexitate ridicată, din această cauză este necesar de a oferi studentului sarcini accesibile, dar care să prezinte interes din perspectiva POO. R. B. Iucu menţionează câteva reguli referitoare la accesibilitate [35, p.167]:

1. Trebuie să se ţină cont de posibilităţile şi de disponibilităţile reale ale celui ce învaţă;
2. Dificultatea sarcinii de învăţare să se situeze cu puţin deasupra posibilităţilor de moment ale educatului, pentru a-l determina să facă efortul de depăşire şi în sens de dezvoltare;
3. Dinamismul sarcinii de învăţare necesită a fi menţinut în scopul obţinerii unei continuităţi a dezvoltării;
4. Nu trebuie eliminate dificultăţile de învăţare, ci, dimpotrivă, trebuie exploatate în favoarea procesului de instruire şi în favoarea educatului.

Utilizând noţiunea de *zonă a proximei dezvoltări*, propusă de psihologul rus Lev Vâgotsky [82], se poate spune că dificultatea sarcinii trebuie să se afle în zona proximei dezvoltări a studentului.

Principiul sistematizării şi continuităţii în învăţare. Între cunoştinţele deja formate şi cele ce urmează a fi formate, trebuie să existe o relaţie de supraordonare şi subordonare. După cum menţionează R. B. Iucu, "relaţiile de supraordonare şi de subordonare între diferitele conţinuturi predate trebuie să se obiectiveze în ierarhii la nivel de compartimente" [35, p.169]. Anume prin intermediul acestui principiu majoritatea competenţelor formate participă la formarea şi dezvoltarea de noi competenţe. De exemplu, în limbajul C++ polimorfismul static poate fi reprezentat prin supraîncărcarea operatorilor. Respectând principiul accesibilităţii, se supraîncarcă operatorii caracteristici pentru numere complexe. Utilizând numerele complexe, se vor forma capacităţi de operare cu conceptul de agregare, prin crearea unui tip de date „matrice formată din numere complexe". Acest exemplu este unul relativ complicat, dar datorită principiului sistematizării şi continuităţii în învăţare, el este elaborat pe etape, fiind accesibil studenţilor.

Principiul corelaţiei inter-disciplinare. Acest principiu presupune abordarea unui demers didactic interferat cu alte discipline de specialitate. Stilul de programare orientat pe obiecte este asemănător stilului de gândire uman. În acest sens nu este corect (şi aproape imposibil) de a ignora corelaţia

dintre POO şi alte discipline, inclusiv a disciplinelor de specialitate. De exemplu, poate să existe o corelaţie dintre POO şi algebră, prin formarea unor tipuri de date precum: grupuri, inele etc. Corelaţia poate fi şi între alte discipline care contribuie la formarea viitorului profesor de informatică. Prin intermediul mediilor de programare vizuală pot fi prelucrate baze de date, care sunt create cu ajutorul diferitor platforme. Acestea conţin o gamă variată de instrumente pentru gestionarea lor.

Principiul integrării teoriei cu practica. Reprezintă valorificarea cunoştinţelor prin soluţionarea diferitor sarcini practice. Strategia didactică aplicată în acest caz presupune:

1. Crearea unor situaţii în care studenţii se confruntă cu probleme practice ce nu pot fi soluţionate în absenţa cunoştinţelor teoretice;
2. Studierea cunoştinţelor teoretice;
3. Reîntoarcerea la problemele practice şi soluţionarea lor pe baza noilor cunoştinţe;
4. Identificarea de noi probleme practice, care pot fi soluţionate cu ajutorul teoriei însuşite.

Aplicarea acestui principiu din perspectiva POO presupune:

a. Analiza proiectelor deja elaborate;
b. Dezvoltarea proiectelor elaborate;
c. Elaborarea propriile proiecte în baza competenţelor formate şi a proiectelor analizate.

În procesul aplicării practice a cunoştinţelor are loc o îmbogăţire a experienţei de cunoaştere prin consolidarea deprinderilor de muncă. Din punct de vedere metodologic, acest principiu susţine ideea transferului de cunoştinţe pentru diferite situaţii.

Finalitatea modelului elaborat, se caracterizează prin formarea la studenţi a şase competenţe specifice, după cum urmează:

CSPoo_1. Explicarea noţiunilor, principiilor şi mecanismelor caracteristice tehnologiei orientate pe obiecte;

CSPoo_2. Identificarea domeniului de valori şi a setului de operaţii admisibile ale unui obiect;

CSPoo_3. Implementarea conceptelor POO în baza unui limbaj de programare orientat pe obiecte;

CSPoo_4. Modificarea stării şi comportamentul obiectului în dependenţă de specificul problemei;

CSPoo_5. Elaborarea de aplicaţii Windows orientate pe obiecte;

CSPoo_6. Proiectarea unei interfeţe grafice.

În mare măsură, acestea se află într-o corelaţie cu competenţele specifice disciplinei ”Informatica” studiată în liceu, conform căreia elevului îi vor fi formate 11 competenţe specifice:

CS1. Formarea unei viziuni ştiinţifice asupra componentei informatice în societatea contemporană.

CS2. Cunoaşterea proceselor, principiilor şi metodelor de codificare şi decodificare a informaţiei în scopul realizării comunicării interumane şi uman – sistem informatic.

CS3. Identificarea structurii generale a sistemelor digitale, a principiilor de funcţionare a sistemelor de transmitere, stocare şi de prelucrare a informaţiei.

CS4. Elaborarea modelelor informatice a obiectelor, sistemelor şi proceselor frecvent întâlnite în activitatea cotidiană.

CS5. Aplicarea metodelor de algoritmizare, de formalizare, de analiză, de sinteză şi de programare pentru soluţionarea problemelor legate de prelucrarea automatizată a informaţiei.

CS6. Translarea algoritmilor frecvent utilizaţi într-un limbaj de programare de nivel înalt.

CS7. Colectarea, păstrarea şi prelucrarea informaţiei cu ajutorul aplicaţiilor software specializate.

CS8. Crearea şi elaborarea documentelor Web.

CS9. Efectuarea experimentelor virtuale, rezolvarea problemelor de activitate cotidiană şi elaborarea de modele ale fenomenelor studiate, folosind aplicaţii, laboratoare şi medii digitale educaţionale; interpretarea rezultatelor obţinute.

CS10. Folosirea competenţelor informatice pentru căutarea şi selectarea informaţiilor în interes de autoinstruire şi orientare profesională.

CS11. Respectarea dreptului de autor asupra resurselor digitale, a normelor de etică şi securitate informaţională. Protejarea de infracţiunile informatice.

Prin formarea competenţelor specifice, POO contribuie la dezvoltarea mai multor competenţe specifice disciplinei ”Informatica” studiată în liceu. Corelaţia dintre acestea este indicată în fig. 2.8.

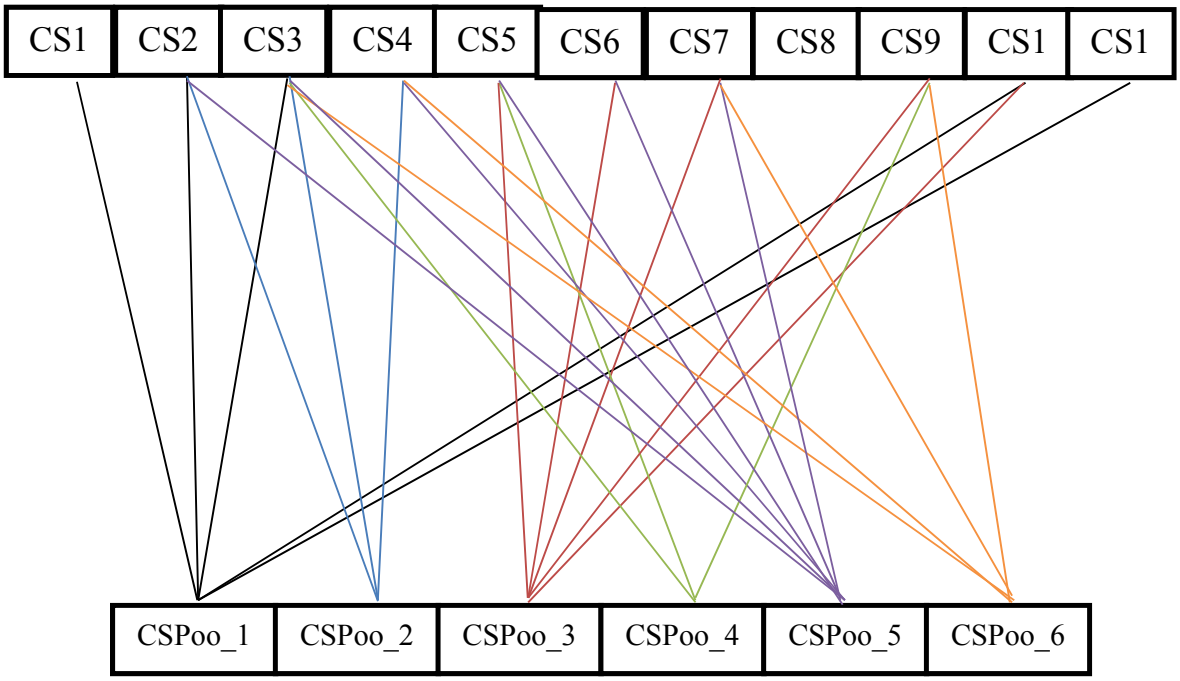


Fig.2.8. Corelaţia dintre competenţele specifice disciplinei ”Informatica” şi competenţele specifice POO.

2.2. Metodologia utilizării modelului elaborat

Termenul metodologie este preluat din limba greacă: „*metodos*” (cale, mod de dobândire a cunoştinţelor etc.) şi *logos* (ştiinţă). Are mai multe interpretări, printre care cea didactică: „sumă a metodelor de predare-învăţare-evaluare a anumitor obiecte, inclusiv a psihopedagogiei şi ca forme de educaţie a generaţiilor în creştere” [49, p.44] şi interpretate pentru un anumit domeniu (disciplină).

Din punct de vedere al programării G. Booch defineşte „*metodologia* - un set de metode utilizate în ciclul de viaţă al unui produs program, unite de o abordare filosofică comună”, iar „*metoda* reprezintă un proces secvenţial de crearea unor modele care descriu, din diverse aspecte, mijloacele sistemului dezvoltat” [112].

O analiză amplă a noțiunii de metodologie este efectuată de către academicianul A. M. Новиков. Cercetătorul rus definește metodologia drept „o modalitate de organizare a activităţilor” [101, p.20]. Autorul propune „schema metodologiei” bazată pe trei elemente: caracteristicile activităţii, componentele activităţii şi divizarea în timp a activităţilor. [101, p.24-25], care poate fi reprezentată după cum urmează (fig. 2.9):

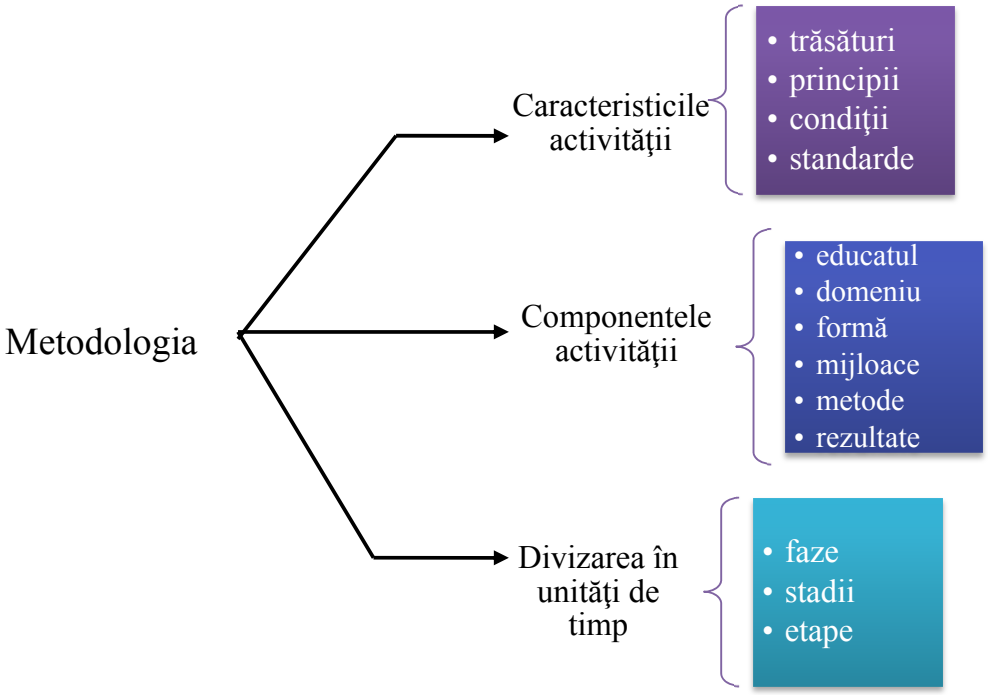


Fig. 2.9. Schema metodologiei după A. M. Новиков.

Aşadar, componente precum: conţinut, metode, forme şi mijloace de instruire sunt indispensabile oricărei metodologii de instruire. И. В Роберт. în [103] a extins numărul componentelor metodologiei, prin includerea a două elemente: finalităţile şi factorii ce influenţează asupra metodologiei (fig. 2.10). Modernizarea sistemului metodic este influenţat în mod direct de discrepanţele dintre finalităţile programate şi rezultatele obţinute.

Pentru formarea şi dezvoltarea competenţei de programare orientată pe obiecte viitorilor profesori de informatică vor fi identificate un şir de competenţe, asimilarea cărora va duce la atingere scopului final.

Competenţele necesită a fi formulate astfel încât să corespundă nivelului de pregătire a viitorului profesor de informatică. La identificarea lor s-a ţinut cont de aspectul didactic, în sensul elaborării, ajustării, utilizării materialului didactic; aspectul metodologic al POO (clase, obiecte, relaţii

dintre ele etc.). În mare măsură acestea ţin de tehnologia POO în limbajul de programare C++ şi mediul de programare vizuală C++ Builder.

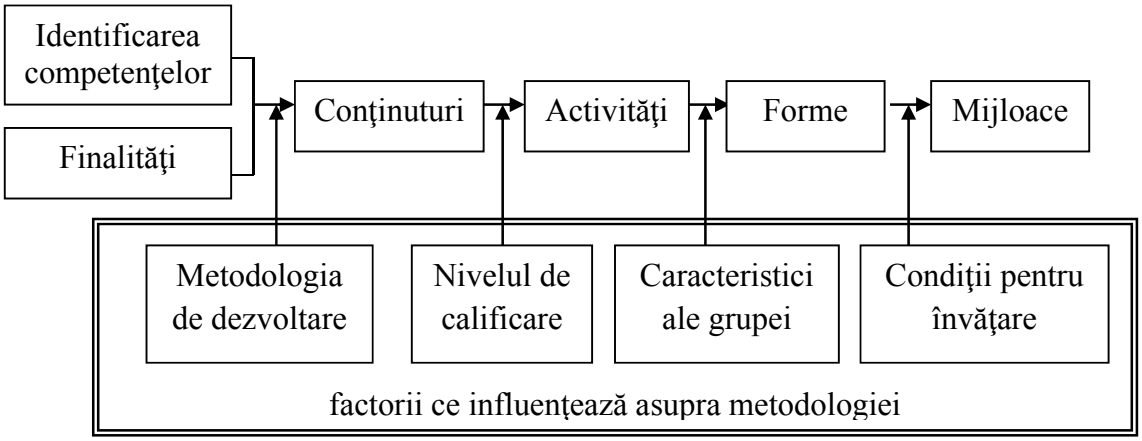


Fig. 2.10. Componentele sistemului metodologic.

Primul pas în elaborarea metodologiei, îl reprezintă identificarea finalităţilor şi formularea competenţelor (vezi p. Xxx). În baza lor sunt identificate elemetele de conţinut (tab. 2.3), care urmează a fi asimilat de către student prin aplicarea mai multor strategii.

Tabelul 2.3. Structurarea conţinuturilor în formarea şi dezvoltarea CPOO.

Modulul 1	**Formarea CPOO în baza unui limbaj de programare**
Unitatea 1	Clase şi obiecte
Unitatea 2	Moştenire
Unitatea 3	Polimorfism
Unitatea 4	Agregare
Modulul 2	**Formarea CPOO în baza unui mediu de programare vizuală**
Unitatea 1	Aplicaţii Windows orientate pe obiecte
Unitatea 2	Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date

Conţinuturile urează a fi structurate în unităţi de învăţare. Unitatea de învăţare reprezintă „*o diviziune logică a unui conţinut care urmează a fi însuşit, împreună cu ansamblul de deprinderi, capacităţi care este corelat acelui conţinut*” [9, p.112].

În mod etapizat unitatea va fi structurată în cinci etape după cum urmează:

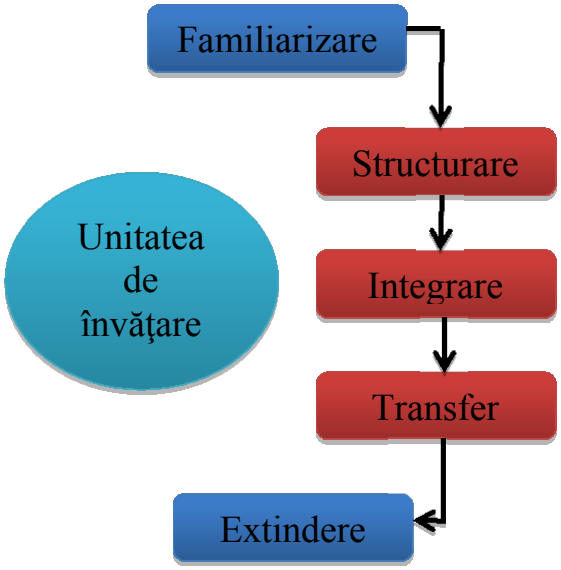


Fig. 2.11. Modalitate de structurare a unei unităţi de învăţare.

1. **Familiarizare** – etapa în care se motivează studentul pentru învăţare. După cum se menţionează în [5, p.112] „procesul de învăţare constă, în principiu, în nivelul adecvat de motivare, adică a acelui sistem motivaţional în măsură să se declanşeze şi să susţină posibilităţi interpretative şi acţionare rămase altfel latente, neutilizate”. În cadrul acestei etape profesorul:

- *prezintă rezultate care necesită a fi obţinute la final*;
- *stabileşte nivelul de cunoaştere a unor noţiuni*;
- *oferă pretexte-problemă, create conflicte cognitive, se recurge la situaţii-problemă.*

2. **Structurare** – la baza unei competenţe se regăsesc o serie de resurse (cunoştinţe, capacităţi ş.a.). Formarea lor, se realizează prin stocarea în memorie a informaţiilor: definiţii, noţiuni, concepte etc. J. R. Anderson menţiona: „cel ce învaţă trebuie să primească, mai întâi, o sumă de informaţii şi instrucţiuni care vor constitui sursa pentru achiziţia procedurilor ori a strategiilor de rezolvare a problemelor” [162, p.63]. În cadrul acestei etape:

- *Sunt formate cunoştinţele teoretice*;
- *Se identifică metode de lucru care să dezvolte rezultate teoretice.*

3. **Integrare** – se manifestă prin rezolvarea unor situaţii practice, adică resursele teoretice sunt utilizate în practică. „Resursele devin resurse numai dacă ele sunt mobilizate pentru a ameliora situaţia, a rezolva o problemă, a

soluţiona o chestiune sau a se adapta la mediu" [9, p. 92]. În cadrul acestei etape:

- *Sunt propuse activităţi pentru aplicarea cunoştinţelor teoretice*;
- *Sunt oferite probleme, în rezolvarea cărora sunt aplicate resurse teoretice şi capacităţi practice recent dobândite*;
- *Se identifică metode de lucru care să dezvolte rezultate teoretico-practice.*

4. **Transfer** – etapa când resursele devin abilităţi, deprinderi, ceea ce oferă posibilitatea studentului de a obţine anumite rezultate. La această etapă:

- *Sunt propuse activităţi pentru aprofundarea subiectului, prin probleme cu un grad sporit de aprofundare*;
- *Sunt prezentate/analizate/rezolvate situaţi, inclusiv din viaţa cotidiană, caracteristice familiei de situaţii.*

Primele patru etape duc la formarea de resurse interne, care necesită a fi utilizate pentru soluţionarea unei situaţii semnificative. O problemă complexă reprezintă „o situaţie, constituită din mai multe situaţii-problemă cu un spectru mai larg de aplicabilitate a resurselor. Situaţia-problemă reuneşte situaţiile de instruire, caracterizate prin faptul că în calea activităţilor practice sau teoretice de cunoaştere a elevilor apare un obstacol, o dificultate" [3, p.71].

5. **Extindere** – etapa în care este conştientizată competenţa şi utilizată spre aplicarea acesteia împreună cu alte competenţe dobândite. În cadrul etapei respective:

- *Sunt trase concluzii asupra problemei specifice unităţii de învăţare*;
- *Sunt prezentate/analizate/rezolvate situaţii, soluţionarea cărora se realizează prin apelare la resurse dobândite în diferite familii de situaţii*;
- *Sunt reunite mai multe familii de situaţii într-o familie mai vastă.*

O astfel de organizare a procesului de formare şi dezvoltare a CPOO permite să formăm la studenţi:

a) *Micro-competenţe*. Situaţii pentru rezolvarea cărora studentul va utiliza până la cinci resurse;

b) *Competenţe*. Situaţii care, la rândul lor, includ mai multe „*situaţii mai mărunte*”. Pentru tratarea acestui tip de situaţii vor fi utilizate între cinci şi zece resurse;
c) *Macro-competenţe*. Situaţii semnificative pentru tratarea cărora studentul va utiliza mai multe de zece resurse.

Odată stabilite, conţinuturile necesită a fi transmise de către profesor şi însuşite de către student, ei aflându-se într-o interacţiune continuă. Transmiterea informaţiei reprezintă o etapă importantă în formarea de competenţe. Pentru aceasta este necesară îndeplinirea a două condiţii:

- Buna organizare a activităţilor. Pentru buna organizare a procesului didactic ambii participanţi urmează să-şi proiecteze activităţile. De modul cum sunt organizate acestea depinde, în mare măsură, nivelul de formare a competenţelor. А. М. Новиков menţionează că la organizarea activităţilor este necesar de a ţine cont de: „1) crearea condiţiilor necesare pentru buna colaborare dintre părţi (student, profesor); 2) un set de procese care duc la îmbunătăţirea relaţiilor dintre părţi; 3) nivelul de implicare a părţilor, acţionând în baza unor reguli şi acţiuni prestabilite” [99, p. 22].
- Selectarea adecvată a metodelor de instruire. Nivelul de reuşită al studenţilor va fi mai ridicat, dacă în procesul de formare şi dezvoltare a CPOO vor fi utilizate o serie de metode specifice de instruire. Vom descrie succint unele metode.

Simularea şi modelarea. Simularea este utilizată pentru prezentarea la faza iniţială a unor concepte, oferind posibilitatea de ghidare a activităţii studentului în bază de situaţii practice. Datorită specificului POO, la etapa iniţială nu pot fi prezentate exemple de programe datorită mai multor factori printre care: complexitatea programelor, sintaxa limbajului etc. În baza simulării pot fi prezentate exemple din viaţa reală de utilizare a unor concepte POO.

Modelarea este „o metodă de cercetare sau învăţare a obiectelor, fenomenelor (legi, principii, norme), care constă în folosirea unui model construit pe baza proprietăţilor esenţiale ale originalului. Modelul (care facilitează învăţarea) este un mod de materializare a generalului, a ansamblului de la care se pleacă în descoperirea elementelor particulare; el reproduce simplificat trăsăturile şi caracteristicile obiectelor şi fenomenelor,

dificil de perceput şi cercetat, în mod direct" [83, p. 35]. Prin intermediul acestei metode se pot reda, prin analogie, diverse situaţii, raţionamente, care pot să reprezinte relaţiile dintre obiecte, fenomene, procese etc. Metoda modelării didactice este cu succes utilizată în domeniul tehnologiei informaţionale. În 1997 G. Booch, I. Jacobson, şi J. Rumbaugh au lansat un limbaj de modelare grafică UML (Unified Modeling Language). După M. Fowler „UML reprezintă o familie de notaţii grafice, la baza cărora se află un singur meta-model. Acesta este predestinat pentru proiectarea şi elaborarea de produse software, în special a celor orientate pe obiecte" [107, p.27]. Graţie utilizării acestei metode, în procesul de instruire se obţin o serie de avantaje: creşterea motivaţiei de învăţare a studentului, timpul redus pentru aplicarea ei, exemple din viaţa reală de utilizarea a conceptelor POO etc.

Problematizarea mai poate fi denumită şi predare prin rezolvare de probleme sau predare productivă de probleme. Conform acestei metode instruitul este pus în faţa unor dificultăţi create în mod deliberat, şi prin depăşirea lor învaţă ceva nou. „Punctul forte" al metodei îl constituie *situaţia-problemă*. În acest sens este necesar de a formula corect situaţia. La crearea situaţiei de tip problemă se va ţine cont de următoarele caracteristici:

A. Situaţia trebuie să prezinte o dificultate pentru instruit, iar pentru a găsi soluţia, este necesar un efort de gândire;
B. Situaţia trebuie să prezinte interes pentru instruit, astfel încât acesta să acţioneze spre a rezolva problema;
C. Situaţia trebuie să orienteze activitatea instruitului spre a rezolva problema şi de a-l cointeresa pe acesta de a dobândi noi cunoştinţe;
D. Rezolvarea situaţiei nu va fi posibilă fără apelarea resurselor recent dobândite.

Prin intermediul situaţiei create, instruitul este cointeresat de a studia, analiza şi a participa la rezolvarea problemei. Aplicarea acestei metode presupune parcurgerea a patru etape:

1. *Formularea problemei* prin descrierea situaţiei-problemă, explicarea, după necesitate, a diferitor puncte cheie, care ar permite instruitului să perceapă problema;
2. *Studierea problemei.* La această etapă se lucrează în mod independent, sunt reactualizate anumite resurse;

3. *Determinarea soluţiei*. În cadrul acestei etape sunt pregătite resursele necesare, se descoperă mijloacele care duc la rezolvarea problemei şi este analizat modul de aplicare a acestora în determinarea soluţiei;
4. *Obţinerea rezultatului final:* se analizează rezultatul obţinut şi se formulează anumite concluzii.

Algoritmizarea reprezintă o metodă de predare-învăţare bazată pe utilizarea şi valorificarea algoritmilor în procesul de instruire. Din punct de vedere didactic, algoritmizarea presupune stabilirea unui traseu în baza căruia sunt structurate unităţile de conţinut ale unei unităţi de învăţare. Algoritmul de instruire se reprezintă sub forma unui grup de scheme, unui set de operaţii, iar parcurgerea lor într-o ordine bine stabilită duce la rezolvarea unui set de probleme caracteristice unei familii de situaţii. În viziunea lui C. Cucoş „algoritmii se caracterizează prin faptul că se prezintă ca o succesiune aproximativ fixă de operaţii, iar această suită este prestabilită de către profesor sau este presupusă de logica intrinsecă a discursului disciplinei respective" [28, p.298]. În rezultatul aplicării acestei metode studentului i se oferă posibilitatea de a elabora treptat propriile scheme, aplicabile în diferite circumstanţe didactice.

Instruirea asistată de calculator este o metodă didactică care valorifică principiile de modelare şi analiză cibernetică. După cum menţionează C. Masalagiu „absolut toate noţiunile, conceptele, exerciţiile, problemele, evaluările, testările, prezentările legate de o anumită temă în cadrul unei lecţii (de informatică) sunt îndepliniri, dirijări, verificări cu ajutorul calculatorului" [38, p.136]. Metoda este aplicabilă în cadrul orelor de seminar şi laborator. Prin intermediul calculatorului se pune la dispoziţia studentului un set de probleme, care necesită a fi analizate, completate sau elaborate. Utilizarea metodei oferă posibilitatea de „organizare a informaţiei conform cerinţelor programei adaptabile la capacităţile fiecărui student; stimularea cognitivă a studentului prin secvenţe didactice şi întrebări ce vizează depistarea unor lacune, probleme, situaţii-problemă; rezolvarea sarcinilor didactice prezentate anterior prin reactivarea sau obţinerea informaţiilor necesare de la resursele informatice apelate prin intermediul calculatorului; realizarea unor sinteze recapitulative după parcurgerea unor teme, module de studiu, lecţii; asigurarea unor exerciţii suplimentare de stimulare a creativităţii studentului" [1, p.11].

Metoda **decompoziţiei** presupune a descompune o situaţie (problemă) mai complexă în situaţii mai simple, iar rezolvarea fiecărei situaţii în parte duce implicit la rezolvarea problemei în întregime. Această metodă este aplicată cu succes în predarea POO. Conform DEX „decompoziţia este un procedeu de determinare prin analiză sau prin calcul a caracteristicilor unui obiect". În programare termenul de decompoziţie este întâlnit în formele:

- ✓ Decompoziţia *algoritmică;*
- ✓ Decompoziţia *pe obiecte*. Este un procedeu de analiză în rezultatul căruia o situaţie (problemă) este tratată prin prisma mai multor obiecte, care se află sau nu, în relaţie cu alte obiecte. În rezultat sunt create mai multe obiecte care împreună oferă soluţia problemei, iar prin intermediul relaţiilor dintre obiecte (moştenire, agregare) se obţin o serie de avantaje caracteristice POO. Decompoziţia pe obiecte oferă un şir de avantaje faţă de decompoziţia algoritmică. Conform G. Booch „ea reduce dimensiunea sistemului software-ului, prin reutilizarea mecanismelor comune" [112, p.16]. Se recomandă a efectua decompoziţia pe obiecte doar după ce studentul va fi capabil să definească conceptele POO. Punerea în aplicare a decompoziţiei va permite studenţilor să se concentreze pe analiza şi crearea claselor, cât şi pe anumite relaţii dintre ele. La aplicarea decompoziţiei se recomandă a urma cerinţele de definire a unei clase:

- structura obiectului trebuie să fie clară;
- obiectul nu trebuie să includă în structura sa (la nivel conceptual), mai multe obiecte;
- obiectul trebuie să fie „complet", adică să conţină suficiente date şi metode pentru prelucrarea acestuia.

Decompoziţia va fi efectuată până când obiectele create vor avea în definirea lor o structură clară, în conformitate cu sarcina dată, şi, totodată, va fi creată o relaţie bine definită dintre celelalte obiecte. Prin decompoziţie, studenţii învăţă a analiza şi a crea obiecte. În consecinţă ei vor deveni capabili să justifice alegerea claselor, cât şi a relaţiilor dintre acestea. Crearea unui obiect începe prin determinarea tipului şi a entităţilor (datelor şi metodelor). Iniţial nu ar trebui să fie stabilite anumite entităţi, ele vor fi stabilite în procesul de analiză a problemei. La stabilirea entităţilor unui obiect se va ţine cont de următoarele cerinţe:

1. Obiectul trebuie să aibă minimum caracteristicile unui tip de date predefinit: a) domeniu de valori; b) set de operaţii admisibile.
2. Obiectul va satisface o serie de cerinţe: a) va consta dintr-o serie de câmpuri, care corespund proprietăţilor datelor descrise, ce permit descrierea domeniului de valori; b) va consta dintr-o serie de operaţii, care corespund operaţiilor admise asupra datelor; c) accesul la câmpuri va fi realizat doar prin intermediul unor operaţii, care vor forma interfaţa tipului.

Prin intermediul acestei metode se face deosebire dintre modul de gândire algoritmic (deja format la studenţi) şi modul de gândire orientat pe obiecte. Conform acestuia problema (situaţia) este divizată în probleme mai mici, organizate sub formă de obiecte, care colaborează între ele.

Metoda **studiul de caz** valorifică o situaţie reală care se analizează şi se rezolvă. „Ea reprezintă o modalitate de apropiere a procesului de învăţare de modelul vieţii, al practicii, având o mare valoare euristică şi aplicativă” [33, p. 96]. În opinia lui S. Cristea studiul de caz „reprezintă o metodă didactică care elaborează acţiunea didactică prin intermediul unor situaţii reale, angajate ca premise inductive şi deductive, necesare pentru realizarea unor concluzii cu valoare de reguli, principii, legităţi” [15, p. 353]. I. Cerghit consideră că studiul de caz „mijloceşte o confruntare directă cu o situaţie din viaţa reală, autentică. Studiul de caz lărgeşte câmpul cunoaşterii, căci un caz invocat poate servi drept suport al cunoaşterii inductive, care trece de la premise particulare la dezvăluirea generalului, la formarea unor concluzii generalizatoare (noţiuni, reguli, principii, legităţi etc.), dar şi invers, ca bază a unei cunoaşteri deductive, de trecere de la general la particular, de concretizare a unei idei, a unor generalizări, de aplicare (de transfer) a cunoştinţelor sau deprinderilor însuşite la situaţii sau probleme noi” [10, p. 233]. Aşa cum problemele rezolvate în stilul orientat pe obiecte au un grad sporit de dificultate, sunt cazuri când este necesar de a prezenta studentului probleme deja rezolvate. Avantajul metodei, constă în faptul că fiecare student îşi va aduce aportul la analiza şi rezolvarea problemei. În utilizarea acestei metode se conturează câteva etape:

1. Selectarea şi prezentarea cazului;
2. Organizarea echipelor de lucru;
3. Prelucrarea şi conceptualizarea;

4. Structurarea finală a studiului.

Instruirea prin proiecte reprezintă „o modalitate de instruire sau autoinstruire graţie căreia elevii, dar mai ales studenţii, efectuează o cercetare orientată spre obiective practice şi finalizată într-un produs ce poate fi un obiect, un aparat, o instalaţie, o culegere tematică, un album, o lucrare ştiinţifică etc.” [55, p. 191]. Pentru prima dată conceptul de proiect este utilizat în 1702 în Paris, unde Academia Regală de Arhitectură, fondată în 1671, anunţa un concurs în cadrul căruia trebuia să fie prezentate planuri de construcţie care au fost numite proiecte. Prin intermediul acestui concurs s-a dorit ca să se evidenţieze creativitatea şi cooperarea între studenţi. „Prin elaborarea proiectului studenţii îşi dezvoltă creativitatea şi, în măsura posibilităţilor, se obţin soluţii originale” [86, p.18]. În prima jumătate a sec. XIX instruirea în bază de proiecte este utilizată şi în alte ţări din Europa, iar din 1879 este utilizată şi în SUA. Astfel elevii „nu doar elaborau proiecte, dar le şi aplicau în practică, construiau motoare, mobilă etc.” [136, p. 91]. Proiectele pot fi clasificate în mai multe categorii, W. H. Kilpatrick în [135] scoate în evidenţă patru tipuri de proiecte:

- Producer’s Project – proiecte care sunt destinate pentru a crea ceva;
- Consumer’s Project– proiecte care pot fi implementate în diferite situaţii;
- Problem Project– proiecte care sunt destinate pentru soluţionarea anumitor probleme practice;
- Learning Project– proiecte care sunt destinate pentru acumularea de cunoştinţe.

În dependenţă de domeniul utilizat, noţiunea de proiect este definită corespunzător. În cercetare proiectul reprezintă un grup de mai multe fişiere care împreună alcătuiesc o aplicaţie. Fiecare fişier reprezintă, o „resursă” care necesită setări speciale pentru a putea fi elaborată aplicaţia finală.

Prin intermediul proiectului se oferă posibilitatea de a evalua o gamă mai vastă de cunoştinţe şi deprinderi ale studentului, chiar mai mult prin elaborarea acestuia studentul îşi manifestă competenţele formate pentru un anumit compartiment. Elaborarea unui proiect presupune parcurgerea mai multor etape. Diferiţi cercetători evidenţiază de la trei până la opt etape în realizarea proiectului. În [86] sunt prezentate diferite modele de elaborare a proiectelor (tabelul 2.4).

Tabelul 2.4. Etapele de elaborare a unui proiect.

în varianta cercetătorilor din S:U.A.	*în varianta cercetătorilor din Europa*
(a) evidenţierea problemei, formularea sarcinii; (b) discutarea variantelor cercetării, selectarea procedeelor; (c) autoinstruirea şi actualizarea cunoştinţelor; (d) repartizarea obligaţiilor; (e) cercetarea: rezolvarea unor sarcini; (f) generalizarea rezultatelor; (g) analiza succeselor şi a greşelilor; (h) corectarea sau trecerea la un alt proiect.	(a) pregătirea; (b) planificarea; (c) cercetarea; (d) rezultatele şi/sau concluziile; (e) prezentarea sau raportul; (f) aprecierea rezultatelor şi a procesului.

În varianta noastră instruirea prin proiecte se realizează prin parcurgerea a patru etape:

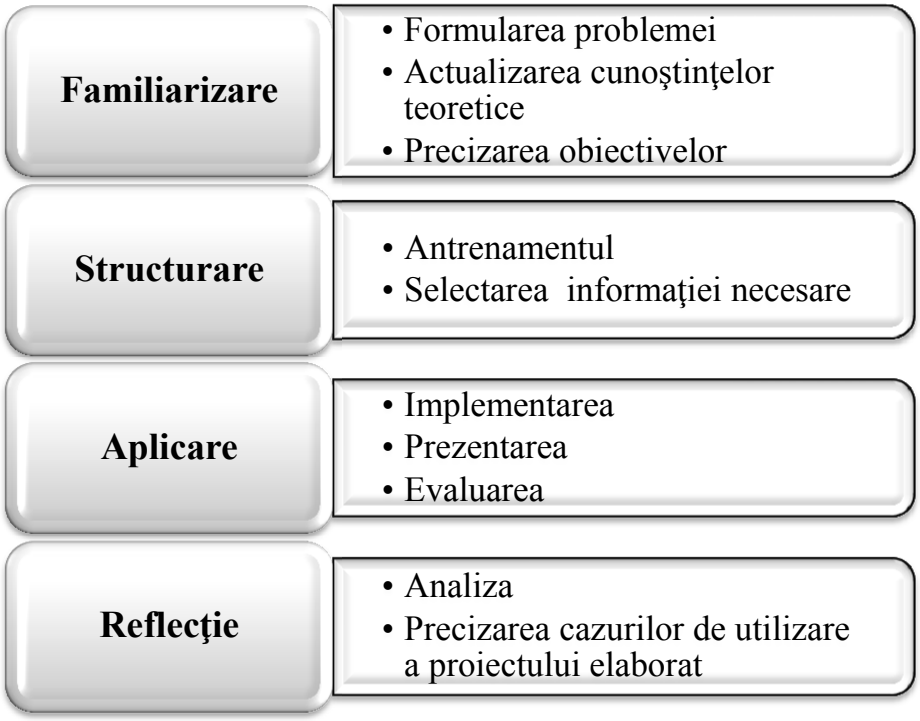


Fig. 2.12. Etapele de realizare a unui proiect.

1. *Familiarizare*. Această etapă se caracterizează prin actualizarea cunoştinţelor necesare şi prezentarea cerinţelor faţă de proiect.

2. *Structurare*. În cadrul acestei etape sunt prezentate exemple de proiecte, împreună cu studenţii sunt elaborate proiecte similare, astfel încât studentul să fie capabil a selecta informaţii necesare pentru implementarea propriului proiect.
3. *Aplicare*. Aceasta este etapa în care studentul elaborează proiectul, îl prezintă şi este evaluat pentru munca depusă.
4. *Reflecţie*. Este o etapă în care sunt analizate proiectele elaborate, sunt precizate cazurile în care urmează a fi utilizat proiectul. Este forte important ca orice proiect elaborat, să fie utilizat şi după evaluarea acestuia.

Metoda proiectul, ca formă de instruire, este o soluţie la noile cerinţe din domeniul tehnologiilor informaţionale cât şi a evoluţiei societăţii contemporane. Elaborarea proiectului reprezintă un pas important în pregătirea viitorului specialist. După realizarea acestuia, studentul obţine o viziune mai clară despre viitoarea lui specialitate.

În baza metodelor de instruire profesorul va colabora cu studentul în vederea formării şi dezvoltării de competenţe (tab. 2.5).

Pentru asigurarea interacţiunii dintre componentele sistemului metodologic se impune o formă de organizare adecvată a acestuia, prin intermediul căreia urmează să se asigure unitatea, corelarea şi articularea armonioasa a activităţii de predare-învăţare.

Tabelul 2.5. Corelaţia funcţiilor student-profesor.

Profesorul	*Studentul*
Transmiterea de informaţii	Acumularea de resurse
Prezentarea de situaţii simple de integrare a resurselor	Formarea de micro-competenţe
Prezentarea situaţiilor de complexitate medie	Formarea de competenţe
Prezentarea situaţiilor de complexitate ridicată	Formarea de macro-competenţe
Studiu de caz asupra unei probleme generale	Dezvoltarea competenţelor

În cazul nostru, procesul de instruire va fi organizat în formă de acumulare de cunoştinţe teoretice, integrare a lor în exemple practice şi

adaptarea lor pentru noi situaţii prin rezolvarea de probleme. Contactul dintre teorie şi practică contribuie în mod substanţial la formarea de competenţe, prin rezolvarea de probleme studentul îşi demonstrează competenţa. Pentru rezolvarea lor, studentul va fi pus în situaţia de a rezolva un set de probleme din diferite compartimente cu nivele diferite de complexitate. Fiecărei unităţi de învăţare îi sunt caracteristice un set de probleme. În cadrul orelor de laborator studentului îi vor fi propuse mai multe tipuri de probe, după cum urmează:

A. Rezolvarea unor probleme cu un nivel scăzut de dificultate. Rezolvarea lor va asigura formarea de *micro-competenţe*. Prin intermediul lor sunt evidenţiate cele mai mici detalii ale unei situaţii; de altfel, la rezolvarea problemelor mai dificile aceste momente nu sunt luate în consideraţie.
B. Probleme de complexitate medie. În acest caz, studentul va apela la o gama mai variată de cunoştinţe şi capacităţi. Prin rezolvarea lor, studentul îşi va demonstra *competenţele* formate în cadrul unei unităţi de învăţare.
C. Manifestarea competenţei prin capacitatea de a selecta şi utiliza resurse necesare pentru soluţionarea situaţiilor semnificative. Prin rezolvarea lor, studentul va demonstra că acţionează competent în familii de situaţii corespunzătoare. Rezolvarea acestor tipuri de probleme asigură formarea de *macro-competenţe*.

Considerăm important ca şi macro-competenţele să fie utilizate în comun, pentru soluţionarea anumitor situaţii. Datorită complexităţii problemelor, studentului i se va propune spre analiză anumite proiecte şi, totodată, probe pentru completarea lor. În acest sens, studentul va elabora, analiza şi dezvolta proiecte.

Procesul de formare şi dezvoltare a CPOO va fi organizat sub formă de:

- prelegeri – lecţii teoretice;
- seminare – lecţii practice;
- lecţii de laborator – lecţii cu un nivel sporit de aplicabilitate din punct de vedere al conceptelor POO.

Ca formă individuală de studiu, în formarea de competenţe, studentul va apela la materialul teoretic, va avea la dispoziţie un set de mijloace (limbaje de programare şi mediul de programare vizuală) pentru aplicarea cunoştinţelor în practică.

Buna organizare a procesului didactic este condiţionată, frecvent, de utilizarea unor mijloace la care profesorul şi studentul recurg în scopul, înţelegerii, fixării, consolidării cunoştinţelor şi capacităţilor. Prin mijloace înțelegem un ansamblu de instrumente care contribuie la realizarea finalităţilor propuse. Ele pot fi: informativ-demonstrative, de exersare şi formare, audiovizuale, de raţionalizare a timpului didactic, cu utilizarea tehnicii de calcul. Din perspectiva POO calculatorul reprezintă un mijloc indispensabil în formarea de competenţe. În calitate de mijloace urmează a fi utilizat un limbaj de programare orientat pe obiecte şi un mediu de programare vizuală. Mijloacele de instruire oferă un şir de contribuţii practice, „favorizează procesul de predare, consolidând şi susţinând mesajul educaţional, fiind suport pentru concretizare ori pentru demonstraţie; susţin şi eficientizează învăţarea chiar şi procesele de formare a noţiunilor a priceperilor şi deprinderilor prin intermediul psihologiei foarte diverse; optimizează relaţia şi comunicarea educaţională prin variabilitatea formelor internaţionale pe care le poate dezvolta, obligând cadrul didactic la o reconsiderare a stilului relaţional promovat în raporturile cu grupul de elevi; perfecţionează sistemul de evaluare prin oferta de tehnicitate care ar reduce efectul de subiectivitate al testelor docimologice tradiţionale; favorizează procesul de orientare şcolară şi profesionala prin prezentările publicitare indirecte ale diverselor sectoare economico-profesionale” [66, p. 2].

Prin intermediul evaluării se oferă posibilitatea de verificare a nivelului de formare a CPOO. Aceasta se realizează la finele fiecărei unitate de învăţare. Canadianul J. Tardif propune nouă principii care stau la baza sistemului de evaluare în bază de competenţe: „1. Monitorizarea procesului de studiu; 2. Evaluarea competenţelor şi nu a resurselor; 3. Determinarea resurselor mobilizabile şi combinabile; 4. Determinarea resurselor mobilizate şi combinate; 5. Delimitarea situaţiilor de manifestare a competenţelor; 6. Documentarea riguroasă a nivelului de dezvoltare a competenţelor pentru fiecare student; 7. Dezvoltarea autonomiei studentului în formarea de competenţe; 8. Utilizarea mai multor criterii la fiecare evaluare; 9. Utilizarea de criterii diferenţiate în aprecierea studentului” [168].

În acest context, studentul va fi supus mai multor forme de evaluare (fig. 2.13).

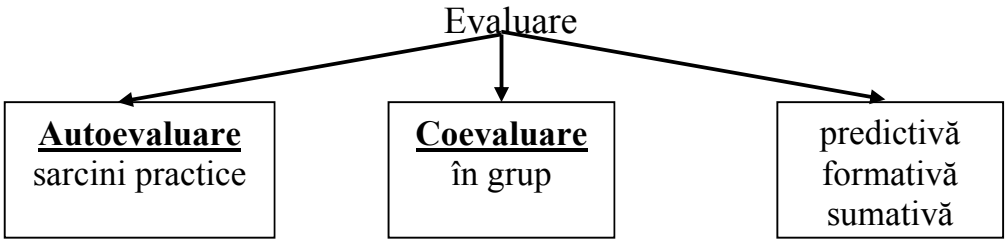


Fig. 2.13. Modalităţi de evaluare a studenţilor.

Aceasta permite cadrului didactic să se pronunţe asupra autenticităţii rezultatelor obţinute de către student.

În viziunea lui V. P. Bespalco [76, p.71] achiziţiile studentului se pot situa la patru nivele: „nivelul I (recunoaştere), studentul poate să recunoască cele studiate; nivelul II (reproducere), studentul poate să reproducă de sine stătător cele studiate; nivelul III (aplicare), studentul poate să aplice în practică cele însuşite în timpul învăţării, selecta informaţie nouă, pentru a realiza sarcinile puse în faţă; nivelul IV (creaţie), studentul este în stare să activeze în mod creator, caută independent informaţia necesară pentru a rezolva probleme apărute în condiţii ne-standarde. Din perspectiva formării şi dezvoltării CPOO, fiecare nivel va reflecta un anumit grad de competenţă la studenţi. Astfel, **Nivelul I** reflectă gradul de posedare a cunoştinţelor, prin definirea principalelor concepte ale POO; **Nivelul II** presupune mobilizarea de resurse pentru rezolvarea unor situaţii simple. Sunt evaluate *micro-competenţele* formate; **Nivelul III** presupune rezolvarea unor situaţii-problemă prin intermediul cărora studentul va da dovadă de *competenţă*; **Nivelul IV** presupune tratarea competentă a unor situaţii semnificative în baza cărora studentului îi vor fi evaluate *macro-competenţele* formate.

În cadrul cercetării s-a măsurat nivelul de formare a CPOO la studenţi prin prisma celor patru nivele descrise mai sus.

2.2.1. Formarea şi dezvoltarea competenţei de programare orientată pe obiect în baza unui limbaj de programare

Din punct de vedere didactic, un limbaj de programare, reprezintă un mijloc de instruire prin intermediul căruia studentului îi sunt formate competenţe de translare a algoritmilor şi de aplicare a metodelor de algoritmizare, de formalizare, de analiză, de sinteză şi de programare pentru soluţionarea problemelor legate de prelucrarea automatizată a informaţiei. Din perspectiva formării şi dezvoltării CPOO, limbajul de programare

selectat ca mijloc de instruire, trebuie să fie cunoscut de către studenţi şi, în acelaşi timp, să conţină cât mai multe concepte ale POO. Conform planurilor de învăţământ al universităţilor din Republica Moldova, la specialitatea ”Informatica” în mare măsură sunt studiate două limbaje de programare: Pascal şi C. Din această cauză la selectarea limbajului de programare, în calitate de mijloc de instruire, se va ţine cont de: (a) sintaxa lui, să fie cunoscută de către studenţi; (b) orientarea limbajului pe obiecte. Amintim că un limbaj de programare este orientat pe obiecte, dacă conţine conceptele de abstractizare, incapsulare, modularitate, ierarhizare.

Astfel au fost identificate două potenţiale limbaje: Object Pascal şi C++, care să corespundă cerinţelor menţionate (tab. 2.6).

Tabelul 2.6. Studiu comparativ privind conceptele POO.

Principii	*Particularităţi*	*Object Pascal*	*C++*
Abstractizare	Instanţiere date	Da	Da
	Instanţiere metode	Da	Da
	Variabile de tip clasă	Nu	Da
	Metode de tip clasă	Nu	Da
Incapsulare	Date	public	public, protected, private
	Metode	public	public, protected, private
Modularitate	Tipuri de module	unit	fişiere
Ierarhizare	Moştenire	simplă	Simplă, multiplă
	Şabloane	Nu	Da
	Metaclase	Nu	Nu
Tipizare	Puternic tipizat	Da	Da
	Polimorfism	Da	Da
Paralelism	Multitasking	Nu	Indirect prin clase
Persistenţa	Obiecte persistente	Nu	Nu

Observăm că limbajul de programare C++ include în structura sa mai multe principii ale POO. Considerăm că în condiţiile sistemului de învăţământ din Republica Moldova, el reprezintă cel mai adecvat mijloc pentru formarea CPOO. Utilizarea acestuia oferă un şir de avantaje: este

cunoscut de către studenţi; este orientat pe obiecte; include în structura sa mai multe concepte ale POO etc.

Pentru a forma studenţilor CPOO în baza unui limbaj de programare, procesul de studiu urmează a fi structurat în patru unităţi de învăţare. Fiecare unitate se caracterizează prin tratarea competentă a mai multor familii de situaţii după cum urmează:

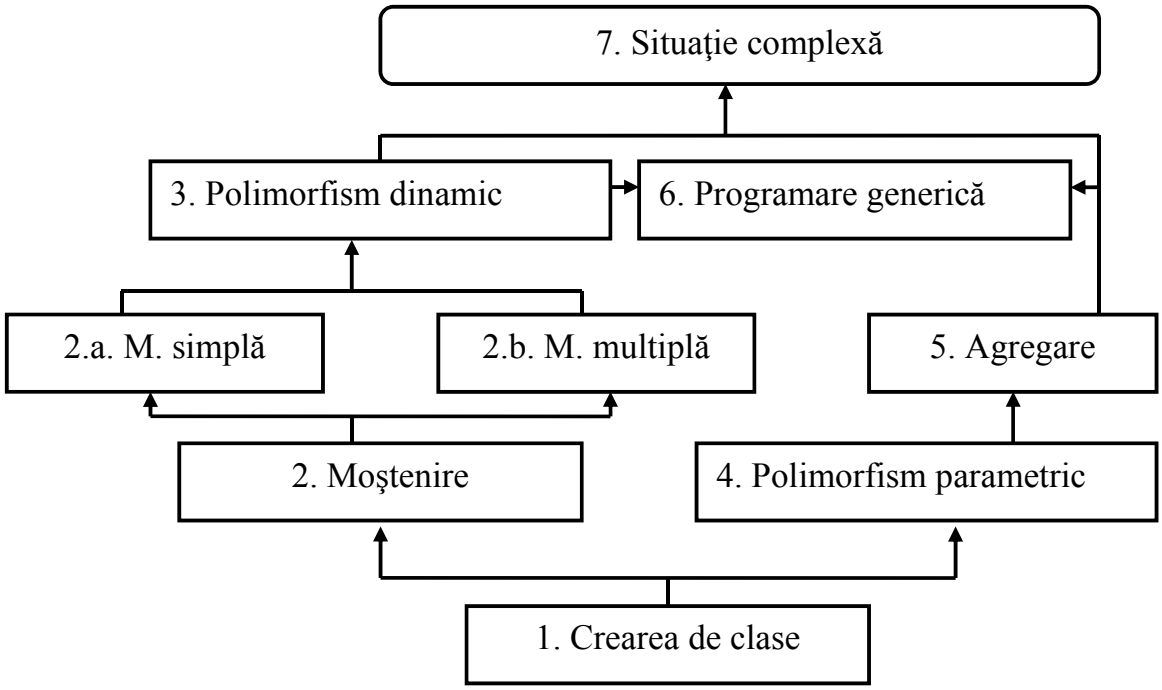


Fig. 2.14. Relaţiile dintre familiile de situaţii.

Aceste familii de situaţii urmează a fi tratate prin prisma unui algoritm (fig. 2.15), cu etape precum:

1. Formarea de resurse;
2. Integrarea resurselor caracteristice unei situaţii;
3. Integrarea situaţiilor caracteristice unei familii de situaţii;
4. Integrarea familiilor de situaţii pentru tratarea unei situaţii complexe.

Prin situaţie complexă se are în vedere o problemă generală în care să se regăsească majoritatea competenţelor formate pe parcursul procesului de studiu caracteristic unui modul.

Ele urmează a fi grupate în patru unităţi de învăţare. Fiecare unitate, în mod etapizat, va fi structurată în cinci etape (fig. 2.11).

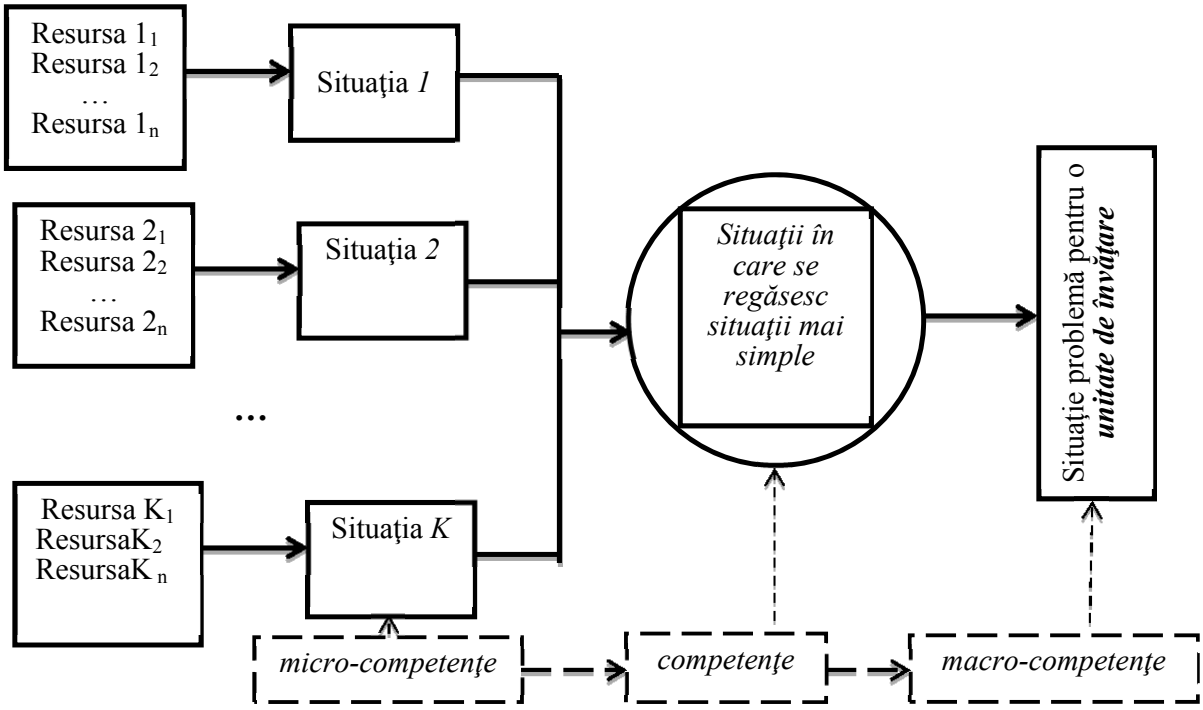


Fig. 2.15. Algoritmul de formare a competenţelor.

Prezentăm modalitatea de aplicare a metodologiei de utilizare a modelului elaborat prin prisma unităţii de învăţare *moştenire*.

Metodologia de aplicare a modelului

Finalitatea unei unităţi de învăţare se caracterizează prin formarea şi dezvoltarea (cu excepţia primei unităţi de învăţare) unui şir de competenţe. Datorită modului de organizare a familiilor de situaţii (fig. 2.14), la această etapă (unitatea de învăţare *moştenire*) studenţii urmează să dobîndească un şir de competenţe caracteristice primei familii de situaţii. Astfel, în cadrul unităţii de învăţare *moştenire*, urmează a fi formate competenţe (Tab. 2.7) şi dezvoltate: *micro-competenţe*: 4, 6, 7, 8, 9, 16, 17, 18; *competenţe*: 1, 2, 3 ,4, 5, 9; *macro-competenţe*: 1, caracteristice unităţii de învăţare: *Clase şi obiecte*.

Etapa de **familiarizare** ţine de motivare studentului pentru învăţare şi formarea unei concepţii despre moştenire. Unul dintre dezavantajele utilizării limbajului de programare, în calitate de mijloc de instruire, ţine de sintaxa acestuia. În baza lui nu pot fi prezentate exemple ale POO, la această etapă, deoarece acesta nu cunoaşte sintaxa limbajului (ulterior urmează a fi studiată). Eliminarea acestui dezavantaj se realizează prin aplicarea unor metode adecvate de instruire, în acest caz va fi aplicată metoda de *simulare şi modelare*. În baza simulării, pot fi prezentate unele avantaje ale utilizării

moştenirii: utilizarea moştenirii reduce în mod considerabil dimensiunea codului unui program; codul programului poate fi reutilizat.

Tabelul 2.7. Competenţe formate pe parcursul unităţii de învăţare**:** *moştenire.*

Unitatea de învăţare	moştenire
micro-competenţe	
1) *Descrierea tipurilor de moştenire*; 2) *Explicarea conceptului de ierarhizare, moştenire;* 3) *Modalitatea de definire a unei relaţii de moştenire simplă*; 4) *Descrierea caracteristicilor unei clase de bază*; 5) *Descrierea caracteristicilor unei clase derivate*; 6) *Modalitatea de protejare a membrilor clasei în relaţia de moştenire*; 7) *Descrierea funcţionalităţii constructorului şi destructorului în relaţia de moştenire*; 8) *Gestiunea constructorilor cu parametri în relaţia de moştenire*; 9) *Argumentarea necesităţii funcţiilor virtuale*; 10) *Caracteristici ale moştenirii multiple*; 11) *Argumentarea necesităţii utilizării cuvântului **virtual** în relaţia de moştenire*; 12) *Modalităţi de redefinire a metodelor*; 13) *Argumentarea necesităţii creării claselor abstracte în relaţia de moştenire*.	
competenţe	
1) *Proiectarea clasei de bază cu o structură redusă;* 2) *Proiectarea clasei derivate cu o structură redusă;* 3) *Crearea unei relaţii de moştenire simplă;* 4) *Crearea unei relaţii de moştenire multiplă;* 5) *Descompunerea unei situaţii de moştenire în clase;* 6) *Definire domeniu de valori fiecărei clase într-o relaţie de moştenire;* 7) *Iniţializare datelor obiectului clasei derivate la nivel de constructor;* 8) *Elaborarea de programe de complexitate medie;* 9) *Implementarea ierarhizării în baza relaţiei de moştenire.*	

macro-competenţe
1) *Proiectarea unei situaţii complexe în clase aflate în relaţia de moştenire;* 2) *Elaborarea programelor cu rezolvarea situaţiilor complexe*.

Pentru simulare, studenţilor li se va propune spre analiză următoarea situaţie: Fie că sunt create trei obiecte în baza a trei clase diferite, aflate în relaţie de moştenire: clasa B moşteneşte clasa A, iar clasa C moşteneşte clasa B. Obiectele sunt create în mod corespunzător: *a* este instanţa clasei A, *b* - instanţa B, *c* - instanţa C. Obiectul *c* este format din 50 de entităţi (date şi metode), obiectul *b* din 20 entităţi, iar obiectul *a* din 10 entităţi. Stabiliţi numărul de entităţi care necesită a fi descris în fiecare dintre clase. Situaţia dată se propune în baza schemei din fig. 2.16.

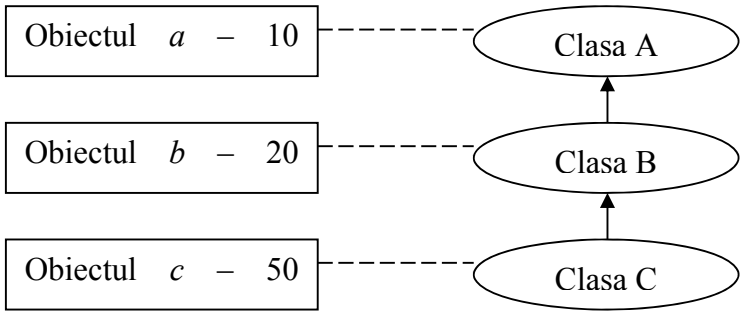


Fig. 2.16. Situaţie de simulare.

Datorită relaţiei de moştenire în clasa A vor fi definite 10 entităţi, în clasa B definite 10 entităţi (10 le preia de la clasa A), în clasa C vor fi definite 30 de entităţi (20 le preia de la clasa B). În rezultatul moştenirii acestor clase codul programului se reduce corespunzător pentru definirea a doar 50 de entităţi, fără moştenire era necesar de a defini 80 de entităţi.

În baza simulării nu se vorbeşte despre componentele claselor. Aceasta poate fi efectuată prin modelare (fig. 2.17). În acest caz, în predarea conceptului de *moştenire*, accentul este pus pe clase şi relaţiile dintre ele. Din punctul nostru de vedere este important ca, mai întâi, să fie înţeles conceptul de moştenire, după care se va indica şi modul de implementare a conceptului prin intermediul unui limbaj de programare orientat pe obiecte. Situaţii similare urmează a fi prezentate studenţilor şi în cazul relaţiei de moştenire multiplă.

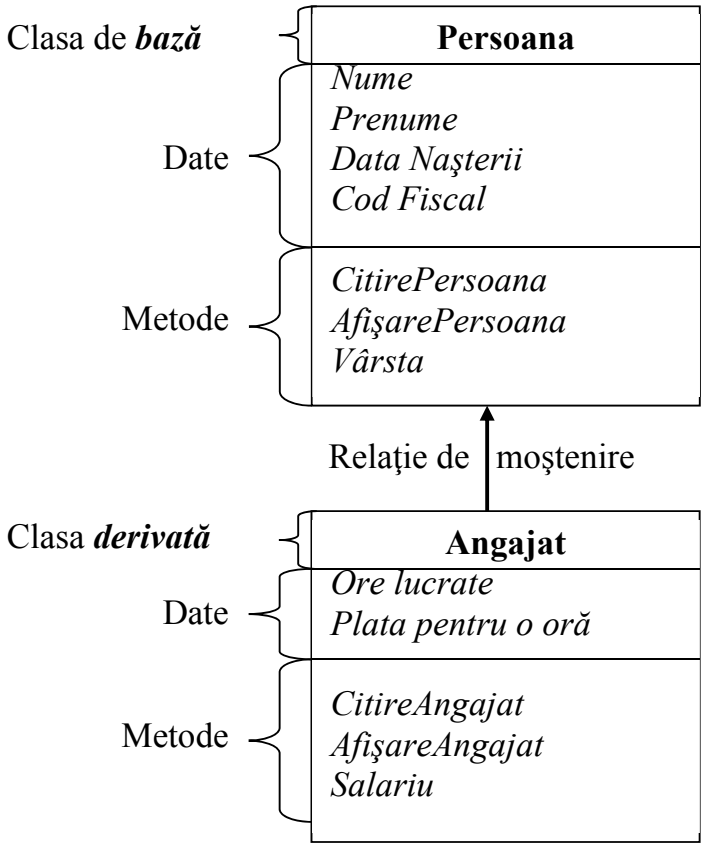


Fig. 2.17. Reprezentarea relaţiei *persoana-angajat* în baza diagramelor UML.

Unitatea de învăţare *moştenire* urmează a fi divizată în două subunităţi:

a) modalitatea de realizare a moştenirii simple în baza limbajului C++;
b) modalitatea de realizare a moştenirii multiple în baza limbajului C++.

O subunitate de învăţare va fi tratată prin prisma a trei etape: Structurare, Integrare, Transfer (Fig. 2.18).

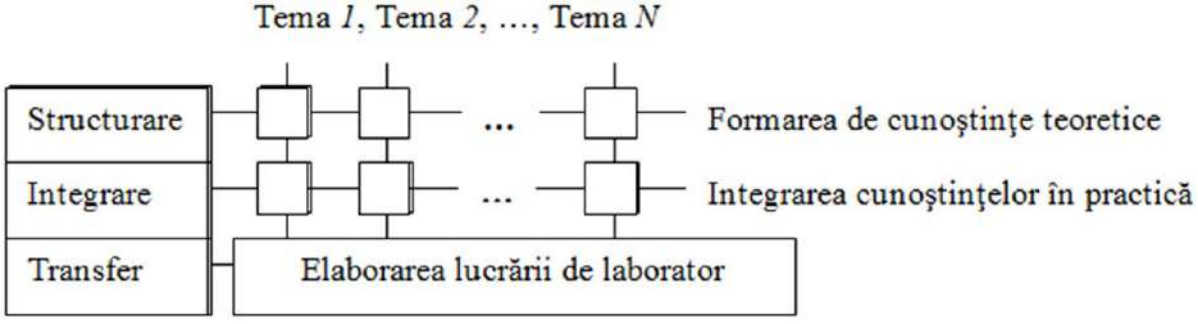


Fig. 2.18. Structurarea unei subunităţi de învăţare.

Fiecare unitate de conţinut (temă) va predată sub aspect teoretico-practic, iar transferul de cunoştinţe se va realiza la orele de laborator.

Moştenirea simplă

Subunitatea *moştenire simplă* se recomandă a fi structurată în trei unităţi de conţinut, după cum urmează:

I) crearea relaţiei de moştenire simplă dintre două şi mai multe clase.

Structurare. Sunt formate cunoştinţe referitoare la sintaxa limbajului C++ în vederea creării relaţiei de moştenire. „Studentul reţine 10% din ceea ce citeşte, 20% din ceea ce aude, 30% din ceea ce vede, 50% din ceea ce vede şi face, 80% din ceea ce spune şi 90%din ceea ce face şi spune în acelaşi timp” [63], în acest sens se recomandă ca relaţia de moştenire să fie prezentată în baza diagramelor de sintaxă (fig. 2.19).

Putem vorbi despre competenţă doar în cazul când cunoştinţele dobândite sunt utilizate pentru a face ceva. Or, „pentru ca instruitul să însuşească cunoştinţe şi să demonstreze priceperi şi deprinderi, el trebuie să se raporteze, într-un anumit mod la ele, adică să-şi formeze anumite atitudini” [47, p.8].

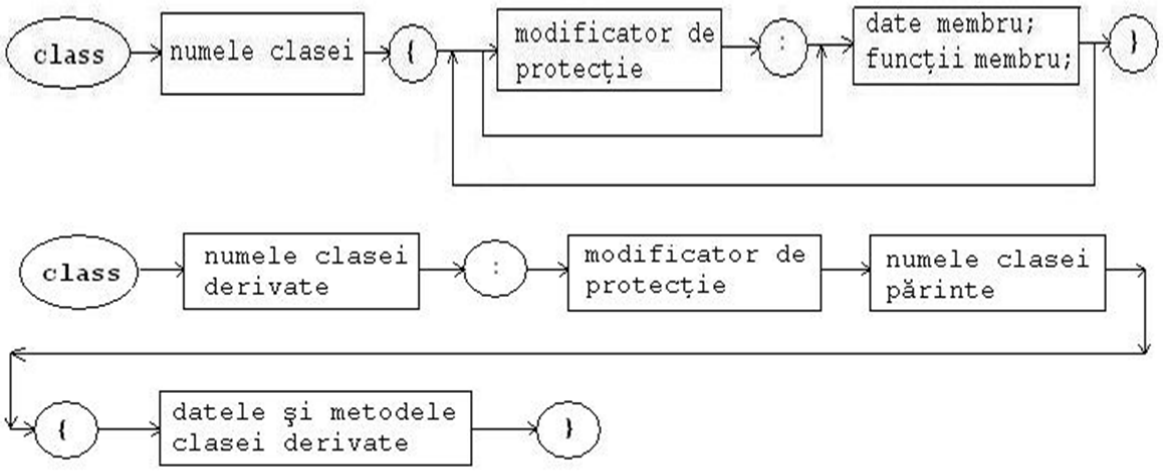


Fig. 2.19. Diagrama de sintaxă a relaţiei de moştenire simplă.

La etapa de **Integrare** vor fi prezentate exemple de creare a unor relaţii de moştenire dintre două sau mai multe clase. Se propune ca exemplele prezentate să cuprindă situaţii precum:

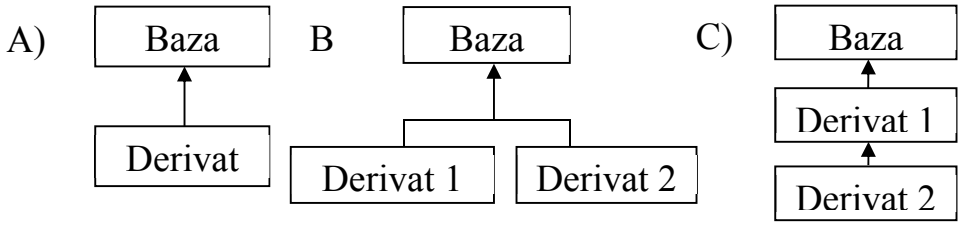


Fig. 2.20. Situaţii de prezentare a relaţiei de moştenire simplă.

S-a observat, că prin integrarea cunoştinţelor teoretice prin prisma acestor trei situaţii studentul ajunge la nivelul de competenţă, fiind capabil să soluţioneze şi situaţii mai complexe. Conform principiului de sistematizare şi continuitate în învăţare aceste situaţii urmează a fi utilizate şi pentru alte unităţi de conţinut.

II) Rolul modificatorilor de acces în relaţia de moştenire.

Structurare. Reprezintă etapa în care sunt formate cunoştinţe despre modul de protejare a datelor şi metodelor claselor aflate în relaţia de moştenire.

Integrare. Se va efectua o analiză asupra celor trei tipuri de situaţii. Pentru primul caz va fi prezentat exemplul:

```
class Baza{
 public:
 int a;
 void f1(){};
 protected:
 int b;
 void f2(){};
 private:
 int c;
void f3(){};
};
class Derivat: modificator de acces Baza{
 public:
 int d;
 void f4(){};
 protected:
 int e;
 void f5(){};
 private:
 int f;
 void f6(){};
};
```

unde în calitate de *modificator de acces* va fi în mod consecutiv vor fi modificatorii: *public*, *protected* şi *private*. În acest mod, vor fi analizate trei cazuri. Studentului i se va propune spre completare următoarele două tabele:

Tabelul 2.8. Nivelul de accesibilitate a membrilor clasei moştenite.

Date şi metode moştenite la nivel *private*	Date şi metode moştenite la nivel *protected*	Date şi metode moştenite la nivel *public*

În baza exemplului vor fi completate fiecare din coloane cu datele şi metodele corespunzătoare. Este important că studentul să poată defini relaţii de moştenire, dar şi mai important este capacitatea de a utiliza obiectele care pot fi create în urma acestei relaţii. În acest sens se consideră declaraţia:

Derivat ob;

pentru a evidenţia datele şi metodele accesibile şi inaccesibile se propune spre completare tabelul 2.9:

Tabelul 2.9. Accesibilitatea datelor şi metodelor obiectului creat în relaţia de moştenire simplă.

Date şi metode obiectului care sunt accesibile	Date şi metode obiectului care sunt inaccesibile

III) Constructorul şi destructorul în relaţia de moştenire

Structurare. Se recomandă a realiza o recapitulare asupra noţiunilor de constructor şi destructor, după care să fie prezentată metoda de lucru a lor în cazul moştenirii.

Integrare. Pentru utilizarea eficientă a timpului cunoştinţele despre constructor şi destructor vor fi integrate prin prisma situaţiilor din fig. 2.20, astfel nu va fi necesar de a construi noi relaţii dintre clase şi definite caracteristicile acestora.

Transfer. Se realizează în cadrul orelor de laborator, cât şi a lucrului independent efectuat de către student. Vom considera că transferul este realizat dacă studentul îndeplineşte corect lucrarea de laborator. La această etapă se propun pentru rezolvare probleme de tipul:

„Se consideră clasa *triad*, date: trei numere reale; metode: *constructorul* cu/fără parametri, *citire* şi *afisare*. Creaţi clasa *triunghi*, derivata clasei *triad*. Pentru clasa *triunghi* vor fi implementate metode pentru determinarea suprafeţei, a perimetrului şi o metodă prin intermediul căreia se va verifica dacă datele introduse pot fi lungimile laturilor unui triunghi. De la tastatură se citesc datele despre *n* triunghiuri. Elaboraţi un program prin intermediul căruia, la ecran se vor afişa datele despre toate triunghiurile, triunghiul cu suprafaţă maximă şi triunghiul cu perimetru maxim" [27, p. 14]. Tot în [27, p. 14 -17] sunt propuse spre rezolvare probleme referitoare la familia de situaţii *moştenire simplă*.

Orice metodologie de instruire se finalizează cu evaluare, în baza căreia este verificat nivelul de formare a competenţei studentului. Pentru evaluare, studentul va rezolva situaţii cu caracter teoretic, de tip exerciţiu, de tip problemă (lucrare de laborator). Situaţiile cu caracter teoretic oferă posibilitatea verificării noţiunilor şi conceptelor studiate.

Exemplu de situaţie cu caracter teoretic:

1. La ce nivel de protecţie necesită a fi protejate datele clasei pentru ca acestea să fie accesibile din exteriorul clasei ?

A. public B. private C. protected

2. Definiţi noţiunea de moştenire simplă.

3. Care este numărul minim de clase care trebuie să participe într-o relaţie de moştenire pentru a putea realiza o moştenire simplă ?

Exemplu de situaţii tip exerciţiu. Se consideră programul:

```
#include<iostream.h>
class baza{
 int i,j;
 protected:
 int arataij(){cout<<i<<j;}
 int puneij(int a,int b){i=a;j=b;}
};
class derivat:public baza{
 int k; public:
 int pune(int a, int b, int c){ k=a;puneij(b,c);}
 int arata(){arataij();cout<<k;}
};
```

```
main(){
 derivat ob;
 ob.pune(8,5,6);
 ob.arata();
}
```

1. Precizaţi volumul de memorie ocupat de variabila de tip obiect ob.
2. Precizaţi vizibilitatea fiecărei metode a obiectului *ob* în interiorul clasei derivate/în afara acesteia.
3. Ce se va afişa în urma execuţiei programului ?

Moştenirea multiplă

Conceptul de moştenire multiplă nu este susţinut de mai multe limbaje de programare orientate pe obiecte cum ar fi: Object Pascal, Java etc. Dacă în calitate de mijloc de instruire nu va fi utilizat limbajul de programare C++ (cu excepţia celor care susţin moştenire multiplă), atunci nu va fi posibil de prezentat acest concept. Subunitatea *moştenire multiplă* se recomandă a fi structurată în două unităţi de conţinut, după cum urmează:

I) crearea relaţiei de moştenire multiplă dintre trei şi mai multe clase.

Ca şi în cazul moştenirii simple la etapa de **structurare** relaţia de moştenire multiplă va fi prezentat prin prisma diagramelor de sintaxa (modul de definire al acestora este similar fig. 2.19), iar la etapa de **integrare** datorită numărului mare de clase care participă la crearea unei relaţii de moştenire multiplă vor fi prezentate situaţii după cum urmează:

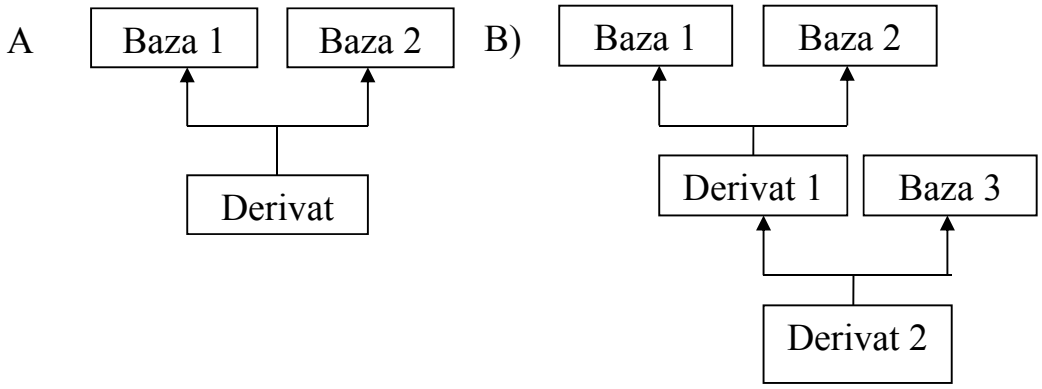


Fig. 2.21. Situaţii de prezentare a relaţiei de moştenire multiplă.

II) Ambiguităţi în relaţia de moştenire multiplă.

În situaţii reale, nu există doar cazuri de moştenire multiplă, ele vin ca o continuare a relaţiei de moştenire simplă, care duc la apariţia unor situaţii

ambigue. Prin situaţie ambiguă se are în vedere o relaţie de moştenire de tipul celei prezentate în fig. 2.22.

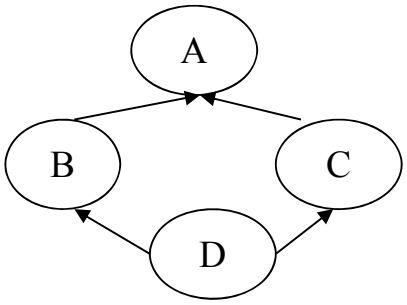


Fig. 2.22. Situaţie ambiguă în relaţia de moştenire.

Un obiect din clasa D va conţine membrii clasei A de două ori, o dată prin clasa B şi o dată prin clasa C. În această situaţie, accesul la un membru al unui obiect de tip D moştenit din clasa A (de exemplu, D ob; ob.x = 2) este interzis (este semnalat ca eroare la compilare).

Aşadar, în cadrul etapei de **Structurare** este prezentată situaţia ambiguă şi modalitatea de tratare a acesteia prin intermediul moştenirii virtuale. La etapa de **Integrare** este prezentat un exemplu de program în care se confruntă cu o asemenea situaţie. În [4, p. 185-187] este prezentat un asemenea exemplu.

Transferul de cunoştinţe se realizează în baza rezolvării unor probleme de tipul [27, p.21-25]. Model de problemă:

În calitate de clase de bază se consideră clasele *pătrat* şi *triunghi echilateral*. Aceste clase derivă clasa *figura*, formată dintr-un pătrat şi un triunghi echilateral. Creaţi clasa *figura*, derivata claselor *pătrat* şi *triunghi echilateral*, dacă lungimea laturii triunghiului echilateral este egală cu *a+0.3*a*, unde *a* este lungimea laturii pătratului. Pentru clasa *figura* vor fi implementate metodele de determinare a suprafeţei şi a perimetrului.

Pentru evaluare vor fi formulate situaţii după cum urmează:

Exemplu de situaţie cu caracter teoretic:

1. Care este numărul minim de clase pentru a crea o ierarhie de tipul moştenire multiplă ?

A. 0 B. 1 C. 2 D. 3

2. Definiţi noţiunea de moştenire multiplă.

3. Descrieţi ordinea execuţiei constructorilor într-o relaţie de moştenire multiplă ?

Exemplu de situaţii tip exerciţiu. Se consideră programul:

```
#include<iostream.h>
#include<conio.h>
class baza1{
 protected:  int x;  public:
 int aratax() {cout<<x;}
};
class baza2{
 protected:  int y;
 public:
 int aratay() {cout<<y;}
};
class derivat:public baza1,public baza2{
 public:
 int pune(int i,int j){x=i;y=j;}
};
main(){
 derivat ob; ob.pune(10,20);
 ob.aratax();
 ob.aratay();
}
```

1. Precizaţi volumul de memorie ocupat de variabila de tip obiect *ob*.
2. Precizaţi vizibilitatea fiecărei metode a obiectului ob în interiorul clasei derivate/în afara acesteia.
3. Ce se va afişa în urma execuţiei programului ?

La această etapă studentului îi sunt formate *micro-competenţe* şi *competenţe*. Pentru formarea de *macro-competenţe* este necesară tratarea unei situaţii care să reprezinte o problemă complexă. Aceasta urmează a fi soluţionată prin aplicarea metodei *decompoziţie*.

Exemplu. În cadrul unei instituţii există mai multe categorii de personal: secretară, administrator, director, cumulat (angajat temporar) şi consultant. Reprezentaţi categoriile de angajaţi prin obiecte. Stabiliţi entităţile fiecărui tip de obiect.

Studenţilor li se propune spre analiză această situaţie. Pentru buna organizare a procesului didactic, se propune a forma grupuri de câte 3-5

studenţi. Fiecare grup va prezenta soluţia găsită. Astfel profesorul va avea posibilitatea să evalueze nivelul de formare a competenţei pentru fiecare grup.

Pentru rezolvarea problemei ei trebuie:

a) să descompună problema în mai multe obiecte;
b) să descrie structura fiecărei clase;
c) să stabilească relaţiile dintre acestea;
d) să se asigure că descompunerea efectuată este una fiabilă, oferă posibilitatea de a adăuga noi tipuri de personal.

Aplicând metoda decompoziţiei pe obiecte se va ajunge la concluzia că este necesar de a crea un obiect care să conţine atributele comune fiecărui tip de obiect din această ierarhie. După stabilirea structurii acestuia vor fi create obiecte noi, pe baza celor existente. Se propune ca decompoziţia să fie realizată conform figurii 2.23.

Prin intermediul dreptunghiurilor este descris numele clasei, iar prin săgeţi se indică relaţia de moştenire (săgeata are o extremitate la clasa derivată şi este îndreptată spre clasa de bază). Studenţilor li se propune a elabora un program în C++ prin intermediul căruia să se efectueze operaţii cu obiectele din ierarhie.

În mod similar urmează a fi tratate fiecare dintre cele trei unităţi de învăţare. Pentru integrarea familiilor de situaţii, va fi efectuat un studiu de caz asupra unei probleme în care se vor regăsi majoritatea familiilor de situaţii deja analizate.

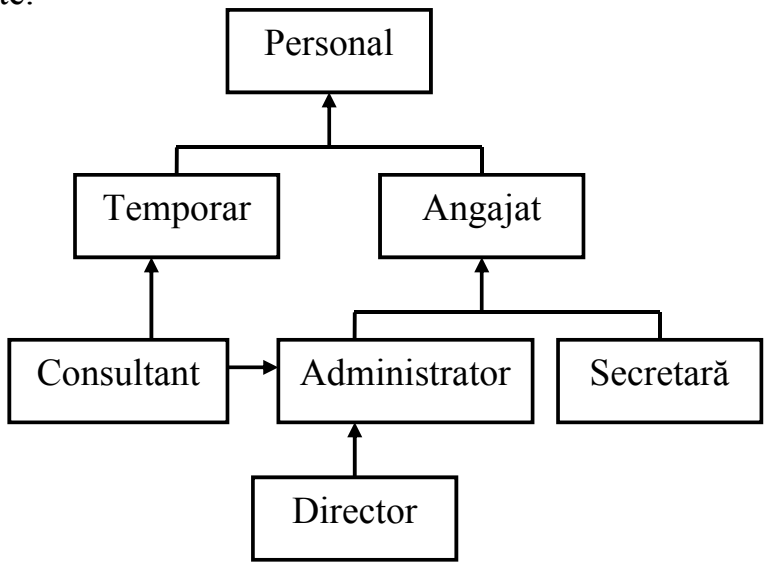


Fig. 2.23. Descompunerea problemei în obiecte.

***Studiu de caz**: Evidenţa personalului unei instituţii.*

***Etapa 1**. Selectarea şi prezentarea cazului*. De elaborat un program prin intermediul căruia se va efectua evidenţa personalului dintr-o instituţie. În instituţia dată sunt următoarele tipuri de personal :

Angajat, despre care se cunosc datele: nume, prenume, codul fiscal, numărul de ore lucrate, plata pentru o oră. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea salariului, determinat după formula:

*salariu=ore_lucrate*plate_ora*.

Angajat de bază, despre care se cunosc datele: nume, prenume, codul fiscal, numărul de ore lucrate, plata pentru o oră, gradul (0,1,2,3), anul angajării. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea salariului după formula:

*salariu=ore_lucrate*plate_ora*.

Angajatul va beneficia şi de un adaos la salariu. Salariaţii cu gradul 0 vor avea un adaos de 50%, gradul 1 – 40%, gradul 2 – 30%. În dependenţă de vechimea în muncă, adausul va fi de 35% pentru cei cu o vechime în muncă ≥ 10 ani, 25% pentru cei cu o vechime în muncă ≥ 3 ani, şi 10% ceilalţi.

Student, despre care se cunosc datele: nume, prenume, codul fiscal, grupa, nota medie. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea mărimii bursei. Studentul nu va avea bursă daca va avea o medie mai mica ca 7.5, va primi 300 lei dacă media e mai mica ca 8.5, 400 lei dacă media e mai mică ca 9.5 şi 500 în caz contrar.

Student Angajat, despre care se cunosc datele: nume, prenume, codul fiscal, grupa, nota medie, numărul de ore lucrate, plata pentru o oră. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea mărimii bursei, şi a salariului. Metodele *bursa* şi *salariu* vor fi determinate conform formulelor din clasele precedente.

Asupra personalului vor fi efectuate operaţii de adăugare a unei persoane, excludere, afişare a tuturor persoanelor din instituţie, cât şi determinarea tuturor banilor ce trebuie achitaţi pentru tot personalul. La ecran va fi afişat un meniu, prin intermediul căruia vor fi prezentate informaţia despre personal.

***Etapa 2**. Organizarea echipelor de lucru*. Pentru rezolvarea acestei probleme va fi necesar de a crea şapte clase. Studenţii se împart în şapte

grupe. Grupurilor formate li se propune spre analiză câte o clasă. Ei analizează structura acestora pentru a o explica colegilor.

Etapa 3. *Prelucrarea şi conceptualizarea*. Fiecare grupă va explica structura fiecărei clase. Profesorul va monitoriza lucrul studenţilor, pentru însuşirea modului de rezolvare a problemei. Programul propus pentru analiză este prezentat în fig. 2.24.

Etapa 4. *Structurarea finală a studiului.* Programul este prezentat studenţilor şi executat pentru interpretarea rezultatelor. Studenţii prezintă avantajele rezolvării acestei probleme prin stilul orientat pe obiecte, comparativ cu alte stiluri cunoscute.

La realizarea acestui studiu sunt incluse majoritatea familiilor de situaţii din fig. 2.14 cu excepţia programării generice. În acelaşi mod sunt dezvoltate şi macro-competenţele caracteristice fiecărei unităţi de învăţare.

```
#define an 2013
class Persoana{
 public:
 char nume[10], prenume[10];
 char idnp[13];
 virtual void citire();
 virtual void afisare();
 virtual double salariu(){
  return 0;}
 virtual double bursa(){
  return 0;}
};
class Angajat :
virtual public Persoana{
 public:
 int ore;
 double pl_ora;
 void citire();
 void afisare();
 double salariu();
};
class Ang_baza :
public Angajat{
 public:
 int an_ang,grad;
 void citire();
 void afisare();
 double salariu();
};
class Student :
virtual public Persoana{
 public:
 char grupa[8];
 double media;
 void citire();
 void afisare();
 double bursa();
};
class Stud_Ang :
public Student,
public Angajat{
 public:
```

```
 void citire();
 void afisare();
};
class celula{
 public:
 Persoana *p;
 celula *next;
 celula(){next=NULL;}
 void cit();
 void afis();
 double consum();
};
```

```
class lista{
 celula *prim;
 public:
 lista(){prim=NULL;}
 void creare();
 void afisare();
 void inserare();
 void exclude();
 double bani();
 ~lista();
 };
```

Fig. 2.24. Modalitatea de structurare a claselor.

2.2.2. Formarea competenţei de programare orientată pe obiect în baza unui mediu de programare vizuală

Un mediu de programare vizuală oferă posibilitatea de creare rapidă a unor aplicaţii uşor de manevrat, destinate sistemelor de operare din familia Windows. Autorul S. Tudor scoate în evidenţă trei caracteristici ale unei aplicaţii Windows [58, p. 9-10]:

1. Programele sub Windows prezintă o interfaţă grafică de excepţie;
2. Programele sub Windows sunt concepute să răspundă anumitor evenimente;
3. Programele sub Windows utilizează biblioteci speciale de subprograme.

În prezent există mai multe medii de programare vizuală precum Borland Delphi, Borland C++ Builder, Visual Basic etc. Drept mijloc pentru aplicarea modelului elaborat a fost selectat mediul de programare Borland C++ Builder, deoarece el este orientat pe obiecte şi este creat în baza limbajului de programare C++. Conform А. Я. Архангельский prin intermediul a C++ Builder pot fi: „

a) create aplicaţii Windows pentru diverse domenii (multimedia, grafică, baze de date etc.);

b) create interfeţe grafice fără cunoştinţe aprofundate;

c) create produse program pentru gestiunea bazelor de date locale şi în reţea;

d) create aplicaţii utilizate de alte programe precum Microsoft Office în special Word, Excel etc.;

e) create aplicaţii care pot fi compilate atât sub Windows, cât şi sub Linux;

f) create aplicaţii pentru lucru în Internet şi intranet;

g) de a utiliza biblioteci .dll, componente gen ActiveX etc." [74, p.35].

În viziunea noastră elaborarea unei aplicaţii în C++ Builder presupune parcurgerea de către student a câtorva etape:

1) determinarea componentelor necesare pentru elaborarea aplicaţiei;
2) dezvoltarea interfeţei grafice, prin plasarea pe suprafaţa formei a componentelor evidenţiate în 1;
3) prelucrarea evenimentelor în conformitate cu datele problemei;
4) eliminarea eventualelor erori logice şi de sintaxă;
5) salvarea/compilarea aplicaţiei şi crearea fişierului executabil.

Având în vedere posibilităţile mediului C++ Builder, considerăm că acesta este cel mai indicat mijloc de utilizare a modelului de formare şi dezvoltare a CPOO la viitorii profesori de informatică.

Pentru a forma studenţilor competenţa de POO în baza unui mediu de programare vizuală, procesul de studiu urmează a fi structurat în două unităţi de învăţare. Fiecare unitate de învăţare va fi structurată astfel încât să permită:

a) analiza structurii mai multor componente „o componentă este definită ca fiind o clasă ce derivă direct sau indirect din clasa *TComponent*." [45, p.17];
b) a realiza proiecte cu utilizarea mai multor componente. „Un proiect este un grup de mai multe fişiere care împreună alcătuiesc aplicaţia C++ Builder. Fiecare fişier reprezintă, pentru aplicaţia din care face parte, o "*resursă*" care necesită setări speciale pentru a putea fi legata aplicaţia finală (DLL sau EXE" [45, p.15]. Prin proiect avem în vedere o aplicaţie Windows orientată pe obiecte;
c) a elabora/dezvolta proiecte;
d) a elabora proiecte cu crearea propriilor obiecte.

În mod schematic, procesul de formare a competenţelor poate fi reprezentat în felul următor (fig. 2.25):

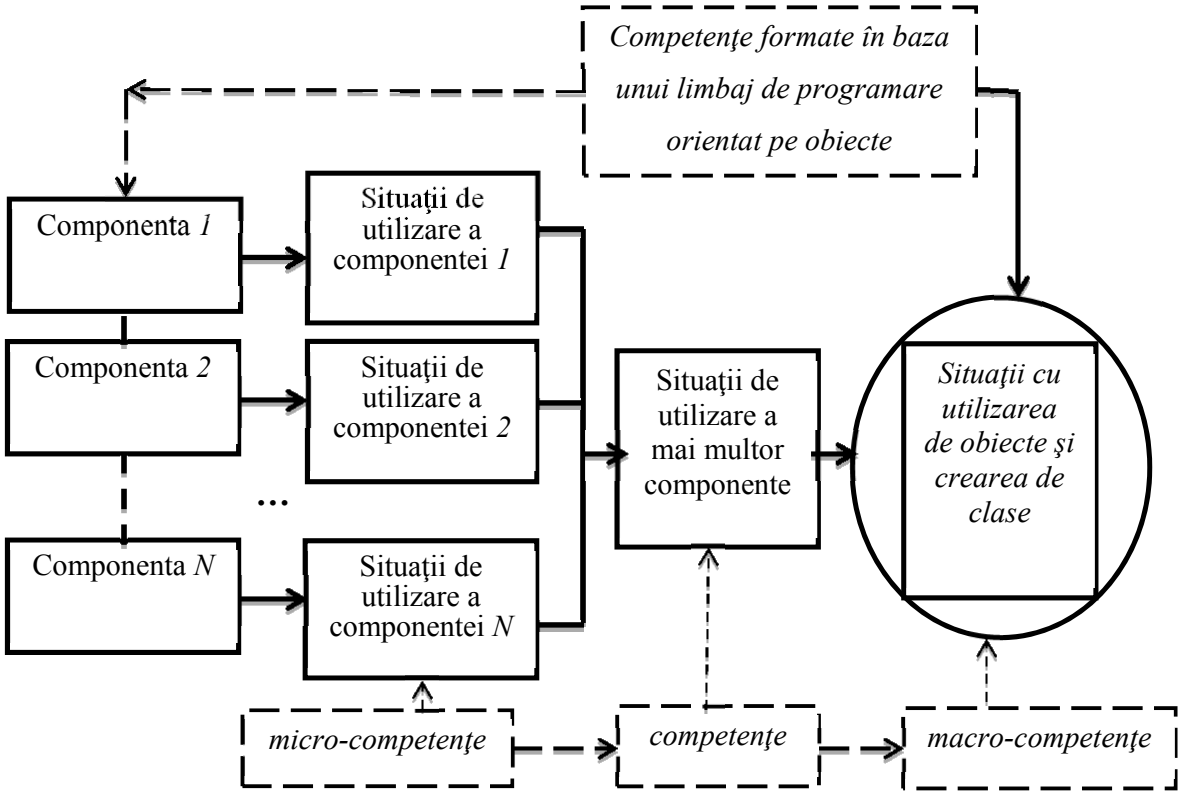


Fig. 2.25. Algoritmul de formare a competenţelor în cadrul unei unităţi de învăţare.

Aplicaţii Windows orientate pe obiecte

În conformitate cu numărul de ore disponibil pentru această unitate de învăţare, ea va fi divizată în două subunităţi, care vor avea ca finalitate câte un proiect, respectiv, aplicaţia *Calculator* şi aplicaţia *Evidenţa personalului*. Pe parcurs vor fi formate un şir de competenţe (tab. 2.10):

Tabelul 2.10. Competenţe formate pe parcursul unităţii de învăţare: *Aplicaţii Windows orientate pe obiecte.*

Unitatea de învăţare	moştenire
micro-competenţe	
Descrierea structurii unei componente; *Descrierea principalelor proprietăţi ale instanţelor clasei: TForm, TButton, TEdit, TLabel, AnsiString, TMemo, TMainMenu, TPopUpMenu, TOpenDialog, TSaveDialog*;	

Descrierea principalelor metode ale instanţelor clasei utilizate în procesul de studiu; *Descrierea principalelor evenimente ale instanţelor clasei utilizate în procesul de studiu*; *Modalitatea de modificare a proprietăţilor unei componente*; *Modalitatea de prelucrare a evenimentelor unei componente*; *Utilizarea ferestrelor pentru afişare în scopul extragerii unei informaţii;* *Utilizarea ferestrelor pentru intrare în scopul citirii unei informaţii;* *Descrierea algoritmului de creare/salvare/executare a unui proiect.*
competenţe
Descrierea paşilor ce necesită a fi efectuaţi pentru crearea unui proiect; *Identificarea componentelor necesare pentru elaborarea unui proiect*; *Identificarea proprietăţilor/metodelor/evenimentelor ale instanţei unei clase ce necesită a fi modificate/apelate/prelucrate*; *Modificarea comportamentului unei componente conform datelor problemei*; *Realizarea operaţiilor de intrare/ieşire a informaţiei la nivel de componente*; *Crearea dinamică a componentelor*; *Elaborarea de proiecte cu un nivel de complexitate medie*; *Dezvoltarea proiectelor existente.*
macro-competenţe
Elaborarea de proiecte cu un nivel sporit de complexitate; *Crearea şi utilizarea obiectelor în cadrul unui proiect.*

O subunitate de învăţare va fi tratată prin prisma instruirii în bază de proiecte. Conform modelului propus în 2.2 elaborarea proiectului se va realiza în mod etapizat (fig. 2.11).

Subunitatea calculator

1) **Familiarizare**. Studentului îi sunt formulate cerinţele faţă de proiect: Elaborarea unei aplicaţii Windows orientate pe obiecte, care va simula funcţionalitatea unui calculator de buzunar. În baza unei analize sunt identificate clasele, instanţele cărora vor participa la elaborarea proiectului: *TForm*, *TButton*, *TEdit*, *TLabel*, *AnsiString*. Studentul este conştient de faptul că doar prin studierea fiecărei clase, va fi capabil să-şi elaboreze propriul proiect.

2) **Structurare**. La această etapă studentul nu are formate deprinderi de lucru cu componente. Din această cauză se va prezenta modalitatea de utilizare a componentelor prin prisma instanţei unei clase. Se recomandă a utiliza instanţa clasei *TForm*, datorită structurii acesteia, care include o serie de proprietăţi, metode şi evenimente caracteristice mai multor componente. Instanţa clasei *TForm* reprezintă o fereastră Windows (numită formă) pe baza ei sunt prezentate noţiunile de proprietate (culoarea formei, dimensiunile ei, bara de titlu etc.), metodă (afişare fereastră, minimalizare, închidere etc.), eveniment (apăsarea unei taste, efectuarea unui click, deplasarea mausului etc.).

3) **Integrare**. Este etapa în care este demonstrată eficienţa utilizării instanţei *TForm*, prin prisma mai multor exemple: efectuarea unui click pe formă produce o schimbare a unei proprietăţi a ei (de exemplu, culoarea) etc.

4). Etapa de **transfer** se va realiza în două direcţii:

a. elaborarea de mici proiecte cu utilizarea clasei în studiu;
b. aplicare - realizarea şi dezvoltarea proiectului.

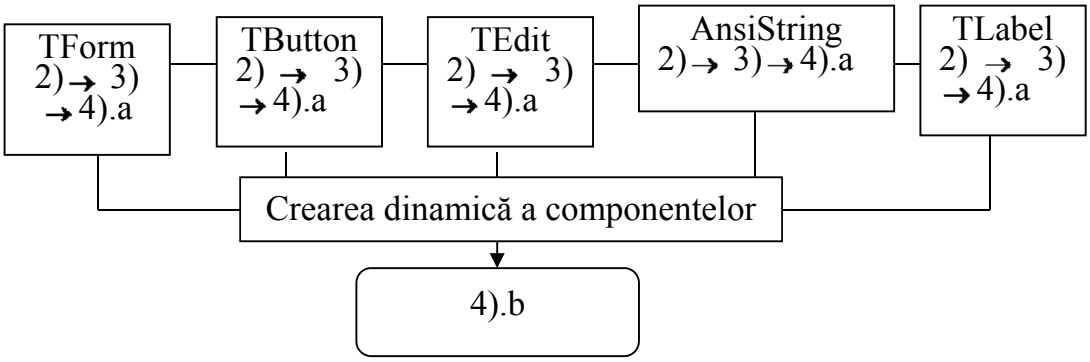


Fig. 2.26. Schema de studiu al claselor subunităţii calculator

Astfel lucrarea de laborator va fi formată dintr-o serie de mici proiecte, care vin să scoată în evidenţă anumite proprietăţi ale unor componente şi a unui proiect cu un grad mai ridicat de complexitate, în dependenţă de capacităţile fiecărui student.

Fiecare dintre clasele identificate în 1) vor parcurge traseul 2)→ 3)→ 4).a conform schemei din fig. 2.26.

Pentru etapa 4).a se propune rezolvarea problemelor din [27, p. 50-57].

Din punct de vedere al POO, un rol important în formarea şi dezvoltarea competenţelor îl constituie crearea dinamică a componentelor.

Pentru crearea lor este necesar de a declara pointeri la clase. Prin intermediul instrucţiunilor:

```
TWinControl *t[3];
t[0]=new TForm(this);
t[1]=new TButton(this);
t[1]->Parent=t[0];
t[2]=new TEdit(this);
t[2]->Parent=t[0];
t[0]->Show();
```

sunt create dinamic trei componente, iar componentele de tipul *TButton* şi *TEdit* sunt plasate pe suprafaţa formei create. În baza acestui exemplu se aminteşte de instanţă, constructor, moştenire etc. Crearea dinamică a componentelor se realizează conform algoritmului: „1. Se declară un pointer de tipul clasei; 2. Se alocă memorie pentru pointerul declarat; 3. Sunt setate valorile iniţiale pentru obiectul creat; 4. Este indicat părintele obiectului; 5. Utilizarea obiectului în cadrul aplicaţiei; 6. Distrugerea obiectului" [4, p. 104-105].

La etapa de aplicare 4).b împreună cu studenţii se realizează o aplicaţie care simulează funcţionarea unui calculator de buzunar. Aplicaţia va efectua calcule cu numere reale. Forma aplicaţiei este prezentată în fig. 2.27.

O versiune a acestei aplicaţii (fig. 2.27) este dată în [4, p. 34-38]. După elaborarea ei, fiecare student va avea creat un proiect de complexitate medie.

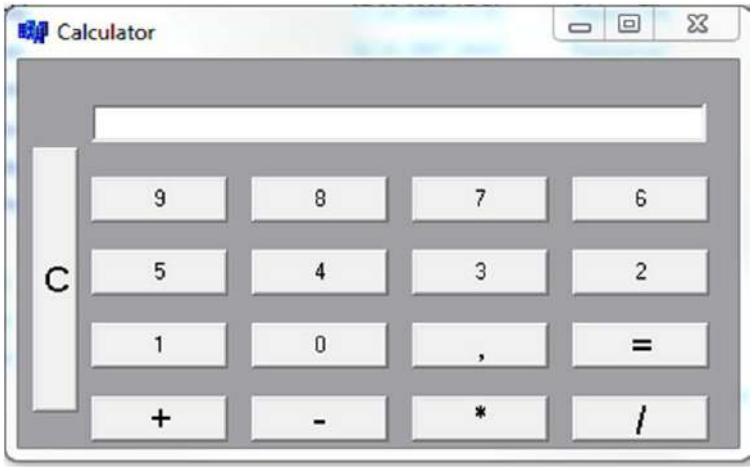


Fig. 2.27. Forma aplicaţiei *calculator.*

Pasul următor constă în dezvoltarea acestui proiect cu adăugare de noi funcţionalităţi. Se propune ca profesorul să ofere studentului posibilitatea ca acesta să dea dovadă de creativitate, posibilitatea de a se manifesta, adică

acesta îşi va propune singur sarcina. D. Isăchioaia [34, p. 1] defineşte creativitatea ca fiind „aptitudinea/ capacitatea de a produce ceva nou şi de valoare, procesul prin care se realizează produsul şi orice rezolvare de probleme noi”. Prin apelarea de resurse necesare studentul dă dovadă de creativitate, ceea ce vorbeşte despre faptul că acesta are deja formate competenţe.

Pentru buna organizare a procesului didactic profesorul va conlucra cu studenţii astfel, încât gradul de dificultatea a sarcinilor propuse să nu difere în mod substanţial.

Extindere. Este etapă necesară pentru a utiliza resurse dintr-o familie mai vastă de situaţii. Caracteristic acestei etape este faptul că pentru realizarea aplicaţiei studentul va fi impus să creeze noi tipuri de date, care nu sunt în mediul de programare utilizat. Aşadar, pentru rezolvarea problemei, studentul este pus în situaţia de a crea obiecte noi cu metode specifice problemei.

De exemplu, după elaborarea aplicaţiei *calculator*, va fi propusă elaborarea unei aplicaţii cu o dificultate mai mare: *Elaboraţi o aplicaţie prin intermediul căreia se va putea efectua operaţii aritmetice cu numere de tip* ***complex***.

Pentru rezolvarea problemei se propune crearea unei clase **complex** (varianta pentru C++ Builder 6.0), cu următoarea structură:

```
class complex{
 public:
 double x,y;//complex=x+yi
 complex(){x=y=0.0;}
 complex(double a,double b){x=a;y=b;}
 complex(AnsiString s);
 friend complex operator+(complex,complex);
 friend complex operator-(complex,complex);
 friend complex operator*(complex,complex);
 friend complex operator/(complex,complex);
 complex operator=(complex b){x=b.x;y=b.y;return *this;}
 AnsiString ToStr();
 friend complex ToComplex(AnsiString);
};
```

Prin intermediul constructorului se oferă posibilitatea de a:

a) iniţializa datele clasei cu 0, adică partea reală şi partea imaginară vor fi egale cu 0;
b) forma un număr complex prin transmiterea părţii reale şi imaginare a numărului complex la nivel de parametri formali;
c) forma un număr complex în baza unui şir de caractere, prin extragerea părţii reale şi imaginare. Va fi utilizat la transmiterea datelor din cutia de editare.

Pentru efectuarea calculelor matematice vor fi supraîncărcaţi operatori aritmetici: +, -, *, / şi operatorul de atribuire. Ultimele două metode sunt necesare pentru a realiza conversia din şirul de caractere în număr de tip complex şi reciproc. După implementarea clasei complex, rezolvarea problemei devine simplă. Este necesar doare de a utiliza obiectele create şi de a le prelucra prin intermediul componentelor. Forma aplicației este prezentată în fig. 2.28.

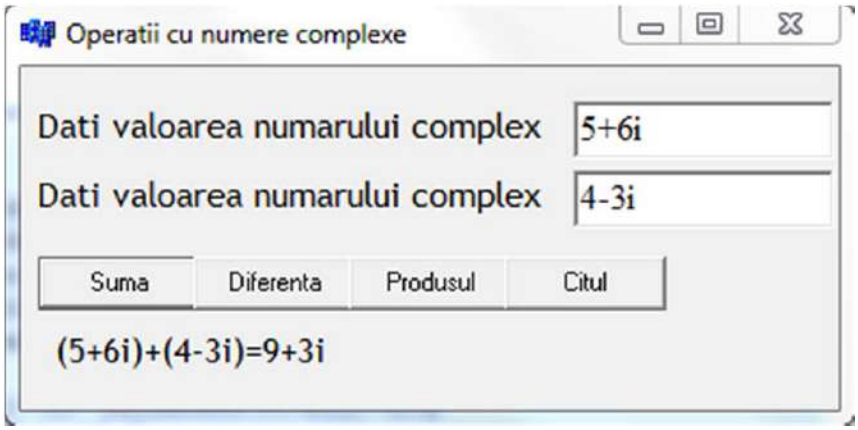


Fig. 2.28. Forma aplicaţiei calculator cu numere complexe.

În acest caz, pentru prelucrarea evenimentului *Button1Click*, vom scrie secvenţa:

```
complex a(Edit1->Text);complex b(Edit2->Text);
complex c=a+b;AnsiString rez;
rez="("+a.ToStr()+")"+"+"+"("+b.ToStr()+")"+"="+c.ToStr();
Label3->Caption=rez;
```

prin intermediul căruia se determină suma a două numere complexe. Rezolvarea integrală a problemei este prezentată în [27, p.43-45].

Subunitatea evidenţa personalului

1) **Familiarizare**. Elaborarea unei aplicaţii prin intermediul căreia se vor efectua operaţii de gestiune a datelor despre studenţi. Pentru realizarea proiectului se vor utiliza instanţele claselor: *AnsiString*, *TLabel*, *TForm*, *TButton*, *TEdit*, *TRadioGroup*, *TGroupBox*, ferestrele *InputBox* şi *MessageBox*.

2) **Structurare**. Formarea de cunoştinţe teoretice referitoare la instanţele claselor utilizate în procesul de studiu. Analiza principalelor proprietăţi, metode şi evenimente ale componentelor.

3) **Integrare**. Prezentarea cazurilor de utilizarea a instanţelor claselor utilizate în procesul de studiu. Realizarea unui studiu de caz asupra domeniului de aplicabilitate a componentelor studiate.

4) **Transfer**. Similar subunităţii *calculator*, vom distinge două etape: a) elaborarea proiectelor cu evidenţierea caracteristicilor fiecărei instanţe a claselor utilizate în procesul de studiu; b) implicarea studenţilor în elaborarea proiectului, completarea acestuia în cadrul orelor de laborator cu noi funcţionalităţi, cum ar fi operaţii de sortare, filtrare, căutare etc.

5) **Extindere**. La această etapă proiectul este completat cu noi tipuri de date. Dacă până la această etapă accentul a fost pus pe modalitatea de gestiune a unui singur tip de date (creat de către utilizator), atunci în cadrul acestei etape vor fi adăugate noi tipuri de date, care să se afle în relaţie de moştenire cu tipul creat. Studenților li se propune spre realizare următoarea sarcină:

Elaboraţi un program prin intermediul căruia se va efectua evidenţa personalului dintr-o instituţie. În instituţia dată sunt următoarele tipuri de personal:

Angajat, despre care se cunosc datele: nume, prenume, codul fiscal, numărul de ore lucrate, plata pentru o oră. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea salariului, conform formulei: *salariu=ore_lucrate*plata_ora*.

Student, despre care se cunosc datele: nume, prenume, codul fiscal, grupa, nota medie.În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor la ecran, determinarea mărimii bursei. Bursa studentului o vom determina astfel:

$$Bursa = \begin{cases} 0\ \ lei, dacă\ \ media < 7.5 \\ 300\ \ lei, dacă\ \ 7.5 \le media < 8.5 \\ 400\ \ lei, dacă\ \ 8.5 \le media < 9.5 \\ 500\ \ lei, dacă\ \ media \ge 9.5 \end{cases}$$

Student Angajat, despre care se cunosc datele: nume, prenume, codul fiscal, grupa, nota medie, numărul de ore lucrate, plata pentru o oră. În calitate de metode vor fi: *citirea* datelor de la tastatură, afişarea datelor, determinarea mărimii bursei şi a salariului. Metodele *bursa* şi *salariu* vor efectua calculele conform formulelor din clasele precedente.

Asupra personalului vor fi efectuate următoarele operaţii: de adăugare a unei persoane, excludere, afişare, sortare, filtrare, căutare a persoanelor din instituţie, cât şi determinarea bugetului lunar al personalului instituţiei.

Cu toate că sarcina propusă are un grad sporit de dificultate, în cazul nostru, codul programului pentru rezolvarea ei este redus, deoarece o variantă similară a acestei probleme a fost rezolvată în baza limbajului de programare C++. Pentru rezolvarea sarcinii studentul îşi va demonstra competenţele formate până la această etapă. Prezentăm structura claselor (tab. 2.11) cu precizarea competenţelor specifice unităţii de învăţare în care au fost formate.

Tabelul 2.11. Structura claselor proeictului elaborat.

Structura claselor	*Precizarea unităţii de învăţare în care au fost formate competenţe*

class Persoana{
protected:
AnsiString nume,prenume,idnp,tip;
public:
Persoana(){tip="Persoana";}
virtual AnsiString afisare();
AnsiString cod(){**return** idnp;}
virtual void citire();
virtual double salariu(){**return** 0;}
virtual double bursa(){**return** 0;}
};
	Unitatea de învăţare 1, 2, 3, 5

class Angajat:**virtual public** Persoana{
protected:
AnsiString functia;
int ore; **double** pl_ora;
public: Angajat(){tip="Angajat";}
void citire();
AnsiString afisare();
double salariu();
};
	Unitatea de învăţare 2, 3, 5

class Student:
virtual public Persoana{
protected:
AnsiString grupa;
double media;
public:
Student(){tip="Student";}
void citire();
AnsiString afisare();
double bursa();
};
	Unitatea de învăţare 2, 3, 5

class Stud_Ang:**public** Student, **public** Angajat{
public:
Stud_Ang(){tip="Student angajat";}
void citire();
AnsiString afisare();
};
	Unitatea de învăţare 2, 3, 5

class celula{
public:
Persoana *p; celula *next;
celula(){next=NULL;}
void cit(**int**);
AnsiString afis(){**return** p->afisare(); }
double consum();
};
	Unitatea de învăţare 1, 3, 4, 5

class lista{
celula *prim;
public:
lista(){prim=NULL;}
void afisare();
void inserare(**int**);
void exclude(AnsiString);
double bani();
~lista();
};
	Unitatea de învăţare 1, 3, 4, 5

În baza acestui exemplu (situaţii) sunt dezvoltate majoritatea competenţelor formate în unitate de învăţare prezentă. Aceasta demonstrează că sunt formate şi dezvoltate competenţe caracteristice primului modul. Rezolvarea de probleme după un astfel de model va reduce în mod esenţial dificultatea de prelucrare a tipurilor de date. Or, gradul de dificultate a unui program creşte odată cu numărul tipurilor de date. Tehnologia POO dispune de mecanisme prin intermediul cărora obiectele cu structură similară pot fi dezvoltate, prelucrate şi acestea pot să comunice între ele. Aceasta reduce în mod substanţial gradul de dificultate al programului. Acest lucru se poate observa şi în cazul nostru, de exemplu, pentru afişarea informaţiei corespunzătoare personalului instituţiei, evenimentul *butonului*, care urmează a fi prelucrat va conţine instrucţiunea: *p.afisare()*, unde *p* este un obiect de tip listă, iar *afisare* este metoda prin intermediul căreia este extrasă informaţia din listă în componenta de tip *TMemo*.

3. ARGUMENTAREA EXPERIMENTALĂ A EFICIENŢEI APLICĂRII MODELULUI ŞI METODOLOGIEI ELABORATE

Modelul pedagogic elaborat, ca şi întreaga cercetare, promovează ideea abordării prin competenţe a procesului de instruire.

Pe durata desfăşurării experimentului pedagogic au fost urmărite efectele variabilelor *independente*: modalităţile de promovare a formelor de organizare în sensul formării şi dezvoltării CPOO, asigurarea studenţilor cu materiale didactice, prezentarea cazurilor de aplicabilitate a POO etc.; asupra variabilelor *dependente:* nivelul de formare şi dezvoltare a CPOO la studenţi, care determină nivelul actual de pregătire a lor. Suplimentar a fost comparat nivelul de motivaţia al studenţilor - variabila *intermediar*ă.

Obiectivele experimentului:

- Demonstrarea eficienţei modelului de formare şi dezvoltare a CPOO;
- Formarea şi dezvoltarea CPOO;
- Stabilirea nivelului de formare şi dezvoltare a CPOO la studenţi.

Locaţia şi perioada desfăşurării: Experimentul a fost organizat şi s-a desfăşurat în anul universitar 2011-2012 la Universitatea de Stat Tiraspol (cu sediul la Chişinău), Universitatea de Stat „Alecu Russo" din Bălţi şi la Colegiul Financiar–Bancar din Chișinău.

Volumul populaţiei: experimentul a cuprins o populaţie de 81 studenţi, dintre care 23 studenţi anul III a UST, 25 studenţi anul II a USB şi 33 studenţi anul IV a CFBC. Componența eșantioanelor și numărul de studenți din fiecare instituție este prezentat în tab. 3.1.

Tabelul 3.1. Componenţa eşantioanelor experimentale.

Eşantionul experimental		*Eşantionul de control*	
Instituția/grupa	Nr.	Grupa	Nr.
UST			
I31	14	IM31	9
USB			
Inform. şi l. eng.	12	Fizica şi informatica	13
CFBC			
Sub. Inf0809G	17	Sub. Inf0808G	16

Prelucrarea rezultatelor experimentului: În statistică există mai multe metode pentru estimarea deosebirilor/asemănărilor dintre două eşantioane. „Eşantionul reprezintă o submulţime a populaţiei statistice avute în vedere” [13, p.148].

Vom numi *eşantion experimental* eşantionul care a fost instruit în baza modelului şi metodologiei elaborate şi *eşantion de control*, eşantionul care a fost instruit în mod tradiţional.

Aşa cum volumul populaţiei în eşantioanele de control nu este identic cu volumul populaţiei în eşantioanele experimentale, pentru validarea rezultatelor experimentale vor fi utilizate două criterii statistice, după cum urmează:

1. Criteriul Cramer-Welch, conform căruia pentru două eşantioane X (cu un volum al populaţiei egal cu *N1)* şi Y (cu un volum al populaţiei egal cu *N2)* se determină valoarea empirică T în baza mediilor şi dispersiilor pentru cele două eşantioane.

Conform criteriului vor fi formulate:

ipoteza nulă H_0: este cea a identităţii repartiţiilor. Nivelul mediu de pregătire în eşantionul de control este apropiat de nivelul mediu de pregătire în eşantionul experimental;

ipoteza alternativă H_1: afirmă că repartiţiile sunt distincte între cele două eşantioane, adică nivelul mediu de pregătire în eşantionul de control se deosebeşte considerabil în sens statistic de nivelul mediu de pregătire în eşantionul experimental.

Estimarea asemănărilor dintre două eşantioane, conform criteriului Cramer-Welch, se realizează conform algoritmului: „ (1) Calculăm valoarea empirică T conform formulei:

$$T = \frac{\sqrt{N1 * N2} * abs(Mx - My)}{\sqrt{N2 * Dx + N1 * Dy}} \tag{3.1}$$

unde, *N1, N2* reprezintă volumul eşantioanelor *X* şi Y respectiv;

$$Mx = \frac{1}{N1}\sum_{i=1}^{N1} x_i \tag{3.2}$$

Mx, My – media caracteristicilor eşantioanelor *X* şi Y respectiv;

$$Dx = \frac{1}{N1-1}\sum_{i=1}^{N1}(x_i - Mx)^2 \quad (3.3)$$

Dx, Dy – dispersia caracteristicilor eşantioanelor *X* şi Y respectiv.

(2) Comparăm valoarea obţinută cu valoarea critică $T_{0.05}$=1.96" [102, p.47].

Decizia în test presupune:

- Se acceptă ipoteza nulă a identităţii repartiţiilor, la nivel de semnificaţie =0,05 fixat, dacă valoarea calculată $T_{emp} \leq 1{,}96$.
- Se respinge ipoteza nulă în caz contrar.

2. Criteriul U a lui Mann-Whitney, este aplicat în cazul estimării deosebirilor dintre două eşantioane după nivelul unei caracteristici concrete. În baza acestui criteriu se verifică următoarele ipoteze:

ipoteza nulă H_0: este cea a identităţii repartiţiilor. Nivelul mediu de pregătire în eşantionul de control este apropiat de nivelul mediu de pregătire în eşantionul experimental.

ipoteza alternativă H_1: afirmă că repartiţiile sunt distincte între cele două eşantioane, adică nivelul mediu de pregătire în eşantionul de control se deosebeşte considerabil în sens statistic de nivelul mediu de pregătire în eşantionul experimental.

Conform algoritmului pentru estimarea deosebirilor dintre două eşantioane independente X (cu un volum al populaţiei egal cu *N1)* şi Y (cu un volum al populaţiei egal cu *N2)* se determină conform algoritmului: „(1) cele *N1+N2* valori se consideră ca fiind împreună şi se ordonează crescător; (2) se atribuie ranguri valorilor ordonate ale şirurilor reunite; (3) se identifică rangurile valorilor aparţinând fiecărui eşantion; notăm cu T_1 suma rangurilor pentru primul eşantion, respectiv, cu T_2 suma rangurilor pentru al doilea eşantion; (4) se calculează cantităţile:

$$W_1 = N1 * N2 + \frac{N1(N1+1)}{2} - T_1, \quad (3.4)$$

$$W_2 = N1 * N2 + \frac{N2(N2+1)}{2} - T_2, \quad (3.5)$$

unde, *N1, N2* reprezintă volumul eşantioanelor *X* şi Y respectiv;

T_1, T_2 – suma rangurilor caracteristicilor eşantioanelor *X* şi Y respectiv;

(5) statistica testului este valoarea minima dintre W_1 şi W_2: $U=\min(W_1,W_2)$" [23, p.208].

Decizia în test presupune:

- Se respinge ipoteza nulă a identităţii repartiţiilor, la nivel de semnificaţie fixat, dacă valoarea. [13, p.209];
- Nu se respinge ipoteza nulă în caz contrar.

3.1. Descrierea experimentului de constatare

Experimentul de constatare a avut ca obiectiv determinarea calităţii formării viitorilor profesori de informatică din perspectiva programării orientată pe obiecte.

Metodele de cercetare utilizate în cadrul experimentului de constatare au fost: *observaţia*, *chestionarul* şi *analiza* publicaţilor de specialitate. Conform S. Cristea „observaţia reprezintă o metodă didactică în care predomină acţiunea de cercetare directă a realităţii prin dirijarea învăţării în secvenţe didactice proiectate la nivelul interacţiunii dintre cunoaşterea intuitivă şi cunoaşterea logică” [15, p.265].

În cadrul experimentului de constatare studenţii din eşantionul UST au fost chestionaţi, pentru determinarea nivelului de pregătire a lor, din perspectiva POO.

În rezultatul prelucrării răspunsurilor s-a constat:

1. Un nivel relativ mic de cunoaştere a principalelor noţiuni (definiţii), din perspectiva limbajelor de programare, de către studenţii eşantionului de control este în mărime de 56,25%, iar în eşantionul experimental în proporţie de din 63,64%.
2. Nu toţi studenţii au deprinderi de lucru cu funcţii/proceduri, astfel:
    a. 67,5% din studenţii eşantionului de control pot să definească corect o funcţie, care să satisfacă cerinţelor problemei, iar în eşantionul experimental 38,18% pot defini funcţii.
    b. 67,19% din studenţii eşantionului de control pot să apeleze o funcţie, fiind dată prototipul acestea, iar în eşantionul experimental corect pot să apeleze o funcţie 36,36%.
3. Evaluarea execuţiei unei secvenţe de program. În acest sens 25% prezintă rezultatul corect, 50% prezintă rezultatul greşit şi 25% nu prezintă nici un rezultat în eşantionul de control. Rezultate relativ mai bune sunt prezentate de către studenţii eşantionul experimental, astfel

72,73% prezintă rezultatul corect, 18,18% prezintă rezultatul greşit şi 9,09% nu prezintă nici un rezultat.

4. Din punct de vedere al realizării unui program, pentru soluţionarea unei probleme concrete 12,5% din studenţii eşantionului de control rezolvă problema pe o notă mai mare sau egală cu 8 (bine), 25% rezolvă problema pe o notă mai mare decât 5 şi mai mică decât 8 şi 62,5% rezolvă problema pe o notă mai mică decât 5 (insuficient). O situaţie similară este şi în cazul eşantionului de control, astfel 9,09% sunt notaţi cu calificativul bine, 27,27% - suficient şi 63,64% - insuficient.

Studiul efectuat a demonstrat existenţa unor lacune importante în formarea şi dezvoltarea competenţelor (în special, a celor ce ţin de programare) la viitorii profesorilor de informatică. Analiza rezultatelor obţinute de către studenţi specialităţii ”Informatica” la disciplina POO în 2010-2011 atestă un nivel slab de pregătire, cu o notă medie la disciplină egală aproximativ cu 6.

În cadrul experimentului a fost efectuată o analiză comparativă a modului de instruire a studenţilor specialităţii ”Informatica” la mai multe instituţii de învăţământ din Republica Moldova şi într-un şir de ţări de peste hotare: România, Belgia, Canada, S.U.A., Franţa (planul de studii). În rezultat s-a constat că la formarea viitorilor specialişti, în cadrul instituţiilor din Republica Moldova, accentul este pus pe instruirea teoretică, în detrimentul instruirii practice. Astfel, după finisarea studiilor, studenţii nu au practică, nefiind apţi de a se angaja în câmpul muncii. Această metodă de formare a specialiştilor în domeniul informaticii are un efect nefast asupra calităţii pregătirii absolvenţilor.

O soluţie viabilă, care face posibilă profesionalizarea formării viitorilor profesori de informatică, o reprezintă instruirea în bază de competenţe.

3.2. Organizarea şi descrierea experimentului de formare

Experimentul de formare a fost realizat în anul universitar 2011-2012 pentru a determina eficienţa modelului şi a metodologiei elaborate. Caracteristic experimentului de formare sunt:

Variabilele de intrare:

A) *ţinute sub control*:

- curriculum disciplinei;
- resursele materiale şi mijloacele tehnice utilizate;
- cadrul didactic (acelaşi).

B) *variabila factor*: prezentă doar în eşantionul experimental. În cazul nostru această variabilă ţine modalitatea de instruire: *abordarea prin competenţe*.

Schema experimentul pedagogic este prezentată în fig. 3.1.

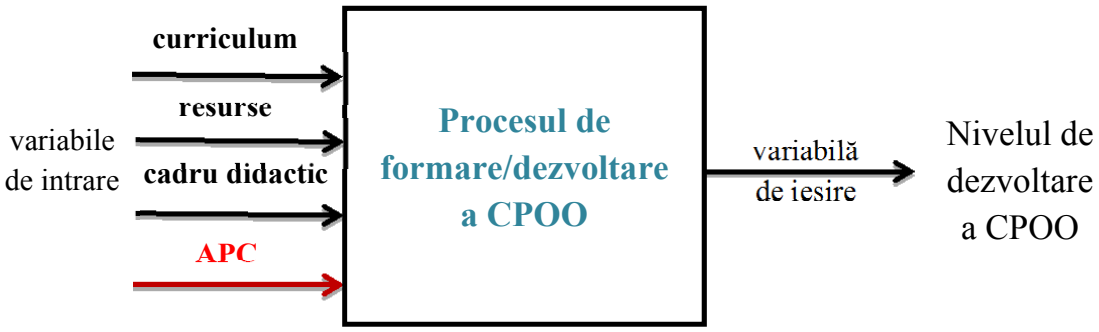


Fig. 3.1. Schema experimentului pedagogic.

La selectarea eşantioanelor s-a urmărit respectarea omogenităţii componenţei lor în raport cu nivelul mediu de pregătire al acestora. Nivelul de pregătire al studenţilor a fost determinat în baza mediei notelor la disciplinele informatice până la semestrul în care s-a predata cursul de POO; pentru studenții de la colegiu drept criteriu a servit nota medie a rezultatelor obţinute la examenul de bacalaureat .

Pentru a demonstra că la începutul experimentului eşantioanele nu diferă în mod substanţial (au un nivel de pregătire apropiat) s-au utilizat două criterii statistice:

1. Pentru criteriul Cramer-Welch au fost formulate ipotezele statistice:

ipoteza nulă H_0: nivelul mediu de pregătire în eşantionul de control (eşantionul 1) este apropiat de nivelul mediu de pregătire în eşantionul experimental (eşantionul 2).

ipoteza alternativă H_1: nivelul mediu de pregătire în eşantionul 1 se deosebeşte considerabil în sens statistic de nivelul mediu de pregătire în eşantionul 2.

În rezultatul aplicării formulei 3.1 s-a determinat valoarea empirică T pentru toate eşantioanele (tab. 3.2). Valoarea critică a criteriului T_{crit}=1,96. Prin urmare, $T \leq T_{crit}$.

Tabelul 3.2. Valorile empirice calculate ale criteriilor statistice.

Valoarea T	Eşantioanele
0,13	UST
1,29	USB
0,49	CFBC

Conform criteriului Cramer-Welch, se acceptă ipoteza H_0 – între caracteristicile eşantioanelor nu există diferenţe esenţiale.

2. Pentru criteriul U a lui Mann-Whitney au fost formulate ipotezele statistice

ipoteza nulă H_0: Nivelul de pregătire al studenţilor în eşantionul 1 nu este mai mic decât nivelul de pregătire al studenţilor în eşantionul 2;

ipoteza alternativă H_1: Nivelul de pregătire al studenţilor în eşantionul 1 este mai mic decât nivelul de pregătire al studenţilor în eşantionul 2.

Tabelul 3.3. Valorile empirice calculate ale criteriilor statistice.

Valoarea U_{emp}	Valoarea critică U_{cr} 0,05	Eşantioane
62,5	26 - 36	UST
51	37-45	USB
110	75 - 86	CFBC

Dat fiind faptul că $U_{cr} < U_{emp}$ (tab. 3.3), ipoteza H_0 nu se respinge. Ambele criterii statistice indică diferenţe nesemnificative între nivelurile pregătirii studenţilor din eşantioanele supuse experimentului.

Deoarece instituţiile de învăţământ superior din Republica Moldova dispun de autonomie universitară, fiecare instituţie are dreptul de a-şi elabora planurile de studii pentru fiecare dintre specialităţi. În acest sens, numărul de ore în care s-a desfăşurat experimentul de formare diferă, în dependenţă de instituţie.

LA UST experimentul a fost desfăşurat în cadrul cursului ”Programare4”, care conform planului de studii conţine *5 credite*, dintre care 75 ore contact direct (30 – ore teoretice (prelegere), 15 – ore seminar şi 30 ore – laborator) şi 75 ore lucru individual. Proiectarea orelor de contact direct pe unități de învățare este prezentată în tab. 3.4.

Tabelul 3.4. Proiectarea pe unităţi de învăţare

	Unitatea de învăţare	Nr. ore		
		Preleg.	Semin.	Labor.
Unit.1	Clase şi obiecte	8	2	4
Unit.2	Moştenire	4	2	4
Unit.3	Polimorfism	6	2	6
Unit.4	Agregare	4	4	4
Unit.5	Aplicaţii Windows orientate pe obiecte	4	2	6
Unit.6	Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date	4	3	6
Total		30	15	30

Spre deosebire de UST, în cadrul USB, experimentul de formare s-a realizat în cadrul cursului „*Limbajul de programare C++ Builder*” cu un total de 4 credite. Din numărul total de 120 ore, 60 sunt cu contact direct în formă de prelegeri şi laboratoare, iar 60 de ore sunt destinate pentru lucru individual. În acest sens, modelul de formare şi dezvoltare a CPOO a fost aplicat parţial (datorită numărului mai mic de ore) prioritate acordându-se modulul doi, conform căruia formarea şi dezvoltarea CPOO se realizează în baza mediul de programare vizuală C++Builder.

Pentru a verifica fiabilitatea modelului şi metodologiei elaborate, experimentul de formare şi dezvoltare a CPOO a fost realizat în CFBC pentru studenţii anului IV a specialităţii ”Informatica”. Planul de studii prevede un număr de 160 ore (contact direct): teoretice (32 ore), practice (24 ore) şi laborator (104 ore).

Pentru proiectarea unei unităţi de învăţare a fost utilizată metodologia Tuning, elaborată şi implementată din 2000 de structuri educaţionale din Europa. Această metodologie a fost adoptată pentru a facilita abordarea curriculară şi pentru a face comparabile programele curriculare. În acest sens la proiectarea conţinuturilor pentru o unitate de învăţare se va ţine cont de:

A. *Finalităţile de studii* – unde se vor specifica competenţele care urmează a fi formate;

B. *Activităţi educaţionale* – rubrică în care se vor descrie formele de organizare şi activităţile întreprinse de către profesor şi student şi formele;

C. *Timpul* – spre deosebire de alte modele de proiectare, în acest caz se va specifica în ore, necesarul de timp pentru un student atât în cadrul orelor, cât şi de lucru în mod individual;

D. Modalităţi de evaluare – se vor indica formele de evaluare a studentului pe parcursul unei unităţi de învăţare.

Proiectarea unităţi de învăţare: ***Aplicaţii Windows orientate pe obiecte***

Tabelul 3.5. Proiectarea unităţii de învăţare *Aplicaţii Windows orientate pe obiecte*

Finalitatea de studii (competenţe)	**Activităţi educaţionale**	**Timp estimat**		**Modalităţi de evaluare**
		D	I	
Descrierea structurii unei componente; *Descrierea principalelor proprietăţi ale instanţelor clasei utilizate în procesul de studiu; Descrierea principalelor evenimente ale instanţelor clase utilizate în procesul de studiu; Modalitatea de modificare a proprietăţilor unei componente; Modalitatea de prelucrare a evenimentelor unei componente; Descrierea algoritmului de creare/salvare/executare a unui proiect;*	**Prelegere**. *Structura unei aplicaţii Windows orientată pe obiecte. Clasa TForm.* **Familiarizare**: Particularităţi ale mediilor de programare vizuală. **Structurare**: Prezentarea claselor specifice mediului BCB. Definirea noţiunii de proiect. Fişierelor care fac parte dintr-un proiect: .bpr, .h, .cpp etc. Gestiunea proiectelor: creare, salvare, deschidere, etc. **Structurare**: stabilirea modalităţii de creare a unei aplicaţii Windows orientată pe obiecte. Prezentarea conceptului de componentă. Structura ei, proprietăţi evenimente,	2	2	Prezent. orală; Prezent. la calculator

	modalitatea de prelucrare a lor. **Integrare**: Analiza structurii componentei de tip TForm. Modificarea comportamentului unei forme prin modificarea de proprietăţi şi prelucrarea de evenimente.			
Descrierea principalelor proprietăţi ale instanţelor clasei utilizate în procesul de studiu; *Descrierea principalelor metode ale instanţelor clase utilizate în procesul de studiu*; *Descrierea principalelor evenimente ale instanţelor clase utilizate în procesul de studiu*; *Modalitatea de modificare a proprietăţilor unei componente*; *Modalitatea de prelucrare a evenimentelor unei componente; Crearea dinamică a componentelor.*	**Prelegere**. *Componente frecvent utilizate. Alocarea dinamică a memoriei*. **Familiarizare**: caracteristici ale componentelor. **Structurare**: Prezentarea componentelor mediului BCB. Prezentarea componentelor prin prisma caracteristicilor sale. **Integrare**: Crearea de aplicaţii Windows cu utilizarea componentelor frecvent utilizate. **Structurare**: Stabilirea modalităţii de alocare şi eliberare a memoriei pentru creare/distrugerea componentelor. **Integrare**: Elaborarea unei aplicaţii cu alocare de memorie.	2	2	Prezent. orală; Prezent. la calculator
Identificarea componentelor necesare pentru elaborarea unui proiect; *Identificarea*	**Laborator**. *Realizarea de aplicaţii Windows orientate pe obiecte cu prelucrarea componentelor*.	2	2	Lucrare de laborator

proprietăţilor/ metodelor/evenimentelor ale instanţei unei clase ce necesită a fi modificate/ apelate/prelucrate; *Modificarea comportamentului unei componente conform datelor problemei*; *Realizarea operaţiilor de intrare/ieşire a informaţiei la nivel de componente*; *Crearea dinamică a componentelor*; *Elaborarea de proiecte cu un nivel de complexitate medie*;	**Integrare**: Analiza unor exemple rezolvate pentru elaborarea de aplicaţii. **Transfer**: Rezolvarea unor probleme ce ţin de gestiunea componentelor şi comunicarea dintre acestea.			
Elaborarea de proiecte cu un nivel de complexitate medie; *Dezvoltarea proiectelor existente; Crearea şi utilizarea obiectelor în cadrul unui proiect; Dezvoltarea proiectelor existente.*	**Seminar**. *Concepte POO în C++ Builder.* **Familiarizare**: reamintirea principalelor concepte care stau la baza POO. **Structurare**: Prezentarea modului de gestiune a claselor în C++ Builder. **Integrare**: Utilizarea obiectelor create în mediul de programare vizual. **Transfer**: Prelucrarea obiectelor create cu ajutorul componentelor. **Extindere**: Analiza modalităţilor de gestiune a	2	2	Prezent. orală; Lucru în grup.

	noi obiecte prin intermediul componentelor.			
Descrierea paşilor ce necesită a fi efectuaţi pentru crearea unui proiect; Dezvoltarea proiectelor existente; Modificarea comportamentului unei componente conform datelor problemei; *Realizarea operaţiilor de intrare/ieşire a informaţiei la nivel de componente*;	**Laborator**. *Elaborarea de proiecte BCB.* **Familiarizare**: caracteristicile unui proiect. **Integrare**: Analiza unui proiect cu un nivel de complexitate mai ridicat. **Transfer**: dezvoltarea proiectului analizat cu noi posibilităţi. Prezentarea proiectului elaborat.	2	2	Lucrare de laborator
Dezvoltarea proiectelor existente; Elaborarea de proiecte cu un nivel sporit de complexitate; *Crearea şi utilizarea obiectelor în cadrul unui proiect.*	**Laborator**. *Aplicaţii Windows orientate pe obiecte cu crearea de obiecte proprii*. **Integrare**: Modalităţi de creare a propriilor obiecte în C++ Builder. **Transfer**: crearea obiectelor cu utilizarea caracteristicilor mediului BCB. **Extindere**: Crearea unui proiect cu prelucrarea obiectelor create prin intermediul componentelor.	2	2	Lucrare de laborator
Total ore		12	12	

În cadrul experimentului pedagogic prelegerile s-au promovat în serii, iar lecţiile practice (seminar, laborator) s-au petrecut independent cu fiecare grupă. Toate lecţiile practice în eşantionul de control au avut loc în mod tradiţional, iar în eşantionul experimental conform metodologiei elaborate.

Caracteristica orelor de laborator

În mod tradiţional, la orele de laborator studentul este pus în situaţia de a rezolva o problemă pentru o lucrare de laborator. Adesea una şi aceeaşi problemă este propusă tuturor studenţilor, iar la final studentul prezintă lucrarea. Profesorului îi revine misiunea de a intervieva studentul în sensul determinării gradului de implicare al acestuia în rezolvarea lucrării. Nivelul de apreciere a studentului depinde, în mare măsură, de severitatea profesorului. Pentru eliminarea acestor dezavantaje a fost elaborată lucrarea metodică [27], structurată în trei capitole: Limbajul C++, Mediu de programare C++ Builder şi Baze de Date. Fiecare capitol conţine mai multe compartimente. În primul capitol, fiecare dintre compartimente conţine o problemă-model rezolvată şi o serie de alte 10 probleme propuse spre rezolvare din familia respectivă de situaţii.

Conform modelului elaborat, formarea şi dezvoltarea CPOO se va realiza în baza a două mijloace. Prin urmare, studentului i se va propune două tipuri de lucrări:

I. Lucrări de laborator caracteristice primului modul. Ele urmează a fi elaborate în baza limbajului de programare C++. Pentru elaborarea acesteia studentul va parcurge etapele:

a) analiza exemplului propus şi interpretarea corectă a rezultatelor programului în urma execuţiei acestuia. Prin urmare pentru realizarea lucrării de laborator studentul va avea două surse: cele primite de student în cadrul prelegerii (cu trimiteri la diferite surse bibliografice) şi un model, de rezolvare a programului.
b) discuţii cu profesorul, după caz, în sensul elaborării lucrării de laborator;
c) elaborarea lucrării de laborator;
d) prezentarea lucrării.

Dacă în grupele de control studenţii nu susţineau lucrările de laborator în termen, atunci în grupele experimentale majoritatea studenţilor s-au încadrat în termenele date pentru elaborarea lucrării.

Caracteristic primului modul au fost o serie de şase lucrări de laborator (tab. 3.6).

Tabelul 3.6. Tematica orelor de laborator pentru modulul 1.

Nr. lucrării	Tematica	Conţinutul
Nr. 1	Proiectarea claselor. Constructori	27, p. 9-14
Nr. 2	Moştenire simplă	27, p. 14-17
Nr. 3	Moştenire multiplă	27, p. 21-26
Nr. 4	Polimorfismul dinamic	27, p. 17-21
Nr. 5	Polimorfismul parametric	27, p. 26-29
Nr. 6	Agregare	27, p. 29-34

Ca exemplu, propunem analiza lucrării de laborator nr.4.

Lucrare de laborator

*Tema***:** Polimorfismul dinamic

Scopul lucrării: Formarea şi dezvoltarea de competenţe caracteristice situaţiilor specifice din familia *polimorfismul dinamic*.

Mijloace didactice: calculator, limbajul de programare C++.

Consideraţii teoretice: conspect, [4], [27], [59], [61].

Model de problemă: Să creăm un program în care vom defini clasa de bază *triunghi* (corespunzătoare triunghiului echilateral) şi clasele derivate *piramida* şi *prisma* (corespunzătoare piramidei triunghiulare regulate şi prismei triunghiulare regulate). Programul va citi datele despre *n* (n<1000) figuri şi corpuri geometrice, apoi va afişa denumirile celor cu aria maximă, respectiv, cu volumul maximal. Numărul *n* de figuri/corpuri geometrice se va citi, de asemenea, de la tastatură.

Realizare: Metodele *afis* şi *volum* sunt metode abstracte, adică pentru clasa *triunghi* ele nu efectuează nimic, dar vom avea nevoie de aceste metode în clasele derivate.

Codul programului:

```
#include<math.h>
#include<conio.h>
#include<iostream.h>
class triunghi{
 protected: double  lat;
 public:  virtual void citire();
 virtual double arie();
```

```
 virtual void afis(){};
 virtual double volum(){return 0;}
};
void triunghi::citire(){
 cout<<"Introdu lungimea laturii (bazei): ";cin>>lat;}
double triunghi::arie(){return lat*lat*sqrt(3)/4;}
class piramida:public triunghi{
 protected: double a,h;
 public:
 void citire();
 double arie();
 double volum();
 void afis();
};
void piramida ::citire() { // redefineste metoda parintelui
  cout<<"Introdu masurile piramidei"<<endl;
  triunghi::citire(); // apeleaza metoda parintelui
  cout<<"Introdu lungimea apotemei :";cin>>a;
  cout<<"Introdu inaltimea piramidei:";cin>>h;
 }
double piramida ::arie(){
 double s; s=triunghi::arie(); // aria bazei
 s=s+triunghi::lat*a/2*3; return s;
 }
double piramida ::volum(){return triunghi::arie()*h/3;}
void piramida ::afis(){
cout<<"Piramida Aria "<<arie()<<" Volumul ";
cout<<volum()<<endl;
}
class prisma:public triunghi{
 protected: double H;
 public:
 void citire();
 double arie();
 double volum();
```

```
 void afis();
};
void prisma::citire(){
 cout<<"Introdu masurile prismei "<<endl;
 triunghi::citire();  cout<<"Introdu inaltimea:";cin>>H;
}
double prisma::arie(){
 double s; s=2*triunghi::arie(); // ariile bazelor
 s=s+triunghi::lat*H;  return s;
}
 double prisma::volum(){  double s; s=triunghi::arie();  s=s*H;return s; }
void prisma::afis(){
 cout<<" prisma Aria "<<arie();
 cout<<" Volumul "<<volum()<<endl;
}
main(){
 triunghi *fig[10];
 int n,i,k,arie,volum;
 double amax,vmax;
cout<<"Introdu numarul figurilor ";cin>>n;
amax=vmax=0.0;
 for(i=0;i<n;i++){ clrscr();
  cout<<" Tastati"<<endl<<"1 Piramida"<<endl;
  cout<<"2 prisma "<<endl; cin>>k;
  if(k==1)  fig[i]=new piramida;
  else  fig[i]=new prisma ;
  fig[i]->citire();
  if(amax<fig[i]->arie()) {amax=fig[i]->arie(); arie=i;}
  if(vmax<fig[i]->volum()) {vmax=fig[i]->volum();volum=i;}
 }
 cout<<"In depozit sunt urmatoarele figuri"<<endl;
 for(i=0;i<n;i++) fig[i]->afis();
 cout<<"Figura cu aria maxima"<<endl;
 fig[arie]->afis();
 cout<<"Figura cu volumul maxim"<<endl;
```

```
fig[volum]->afis();
for(i=0;i<n;i++) delete fig[i];
}
```

Interpretarea rezultatelor

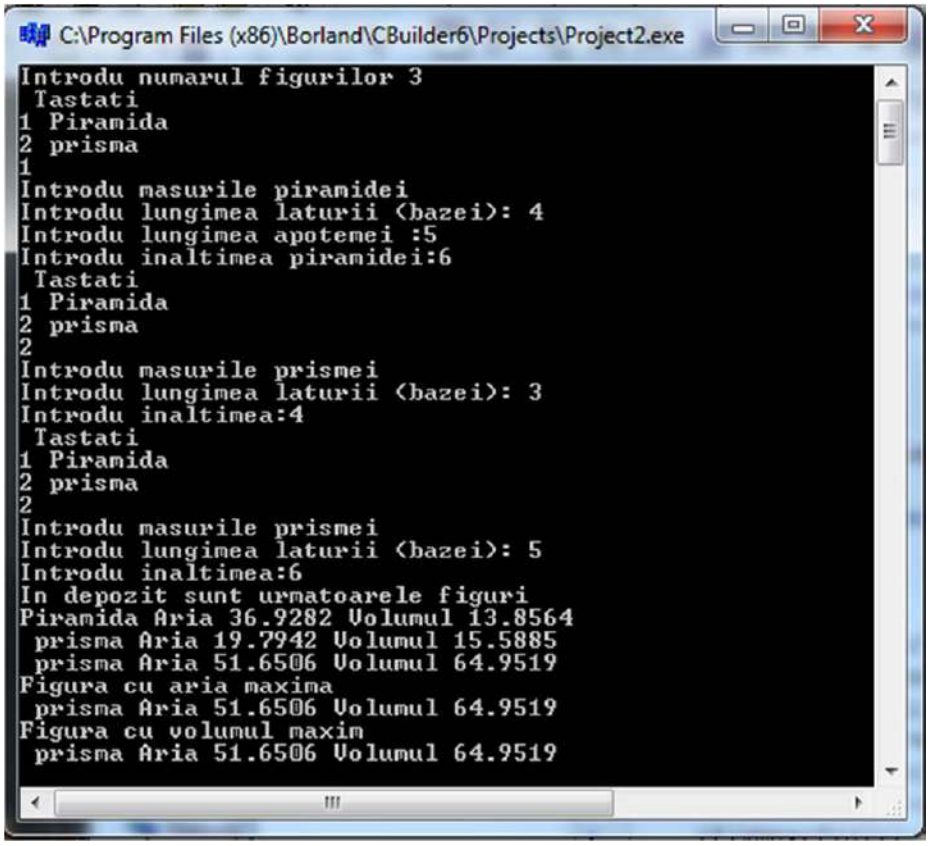


Fig. 3.2. Rezultatul obţinut în urma execuţiei programului.

Sarcini pentru rezolvarea independentă (o variantă admisibilă):

Se consideră ierarhia: *persoană*, *angajat* şi *student*. Clasele *angajat* şi *student* sunt derivatele clasei *persoană*.

În clasa *persoană* vor fi definite următoarele:

date: numele, prenumele, anul naşterii;

metode:

Citire – citirea de la tastatură a datelor despre persoană,

Afisare – afişarea la consolă a datelor despre persoană;

Virsta – va returna vârsta persoanei.

În clasa *angajat* suplimentar vor mai fi declarate:

date: Ore – numărul de ore lucrate, pl_ora – plata pentru o oră, funcţia;

metode: *Bani* – va determina salariul angajatului conform formulei: *ore*pl_ora.*

În clasa *student* suplimentar vor mai fi declarate

date: media, grupa;

metode: *Bani* – va determina bursa studentului conform formulei: media*75 lei, dacă

media studentului este mai mare decât şapte, în caz contrar returnează 0.

Pentru această ierarhie se va implementa polimorfismul pentru metodele: *citire*, *afişare*, *bani*. De la tastatură se citeşte numărul total de persoane *n* (n<100). Tipurile acestora se citesc în momentul execuţiei. La ecran va fi afişat un meniu cu următoarele opţiuni:

a) Afişarea tuturor persoanelor;
b) Afişarea persoanelor angajate;
c) Afişarea persoanelor care sunt studenţi;
d) Suma totală de bani, pe care o primesc toate persoanele citite;
e) Persoana sau persoanele ce primesc cei mai mulţi bani;
f) Persoana sau persoanele ce primesc cei mai puţini bani;
g) Persoana sau persoanele cele mai tinere;
h) Afişarea persoanelor, în ordine descrescătoare a veniturilor acumulate;
i) Afişarea persoanelor în ordine crescătoare vârstei.

Modul de lucru

1. Compilaţi şi executaţi programul (model de problemă);
2. Analizaţi rezultatele obţinute;
3. Realizaţi sarcina propusă pentru rezolvare independentă;
4. Verificaţi rezultatele obţinute, cu executarea programului pentru diferite date de intrare;
5. Prezentaţi avantajele obţinute în urma elaborării programului în stilul orientat pe obiecte;
6. Formulaţi concluziile cu referire la rezultatele obţinute.

Întrebări de verificare

1. Ce se numeşte polimorfism ?
2. Care este necesitatea declarării unei metode ca virtuală ?

3. Prin ce se caracterizează o clasă abstractă ?
4. Care este necesitatea utilizării funcţiilor virtuale pure ?
5. Ce avantaje se obţin în rezultatul utilizării polimorfismului în rezolvarea de probleme ?
6. Prezentaţi situaţii din viaţa reală a polimorfismului dinamic.
7. Explicaţi funcţionalitatea situaţiilor de polimorfism utilizate în lucrarea elaborată.

II. Lucrări de laborator caracteristice modului doi. Conform conţinuturilor, în cadrul acestui modul se prezintă tehnologia POO prin prisma componentelor mediului de programare vizuală C++ Builder. În cadrul experimentului de formare, la acest modul, studenţilor li s-au propus pentru elaborare două lucrări de laborator, structurate după cum urmează:

a. serie de proiecte cu un grad mai mic de dificultate, pentru a scoate în evidenţă anumite proprietăţi ale componentelor şi a permite studentului mai îndeaproape să studieze structura componentelor respective. În acest sens în lucrarea metodică [27, p.50-80, 95-110] sunt formulate o serie de probleme care pot fi propuse studentului spre rezolvare.
b. analiza unui proiect cu dificultate medie, propus de către profesor studenţilor;
c. dezvoltarea proiectului analizat, prin completarea acestuia cu noi funcţionalităţi.

Datorită numărului de probleme incluse în lucrare şi a complexităţii proiectelor, au fost propuse pentru fiecare student câte două lucrări de laborator (câte una pentru fiecare unitate de învăţare, tab. 3.7).

Tabelul 3.7. Tematica orelor de laborator pentru modulul 2.

Nr.	*Tematica*	*Conţinutul*
Nr. 7	Aplicaţii Windows orientate pe obiecte	27, p. 35-80
Nr. 8	Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date	27, p. 81-110

Aceste două lucrări de laborator, în linii mari, vor fi structurate în mod similar. Pentru lucrarea de laborator Nr. 7 se propune a o structura, după cum urmează:

Lucrare de laborator

Tema: Aplicaţii Windows orientate pe obiecte

Scopul lucrării: Formarea şi dezvoltarea competenţelor necesare pentru a elabora o aplicaţie Windows orientată pe obiecte.

Mijloace didactice: calculator, mediul de programare vizuală C++ Builder.

Consideraţii teoretice: conspect, [4], [27], [45], [74], [77], [96].

Sarcini pentru rezolvarea independentă (o variantă admisibilă):

a.1) Clasa **TForm**: Elaboraţi o aplicaţie care va afişa iniţial forma la centrul suprafeţei de lucru. La apăsare tastelor ↑,↓ forma se va deplasa pe diagonala principală a suprafeţei de lucru. Apăsarea caracterului '*s*' va poziţiona forma în centru, iar la apăsarea tastelor ↑,↓ forma se va deplasa pe diagonala secundară. Apăsarea caracterului '*p*' va poziţiona forma în centru, iar forma se va deplasa pe diagonala principală la apăsarea tastelor ↑,↓.

a.2) Clasa **TButton**: Creaţi aplicaţia MOUSE. Pe suprafaţa formei cu titlul "MOUSE" sunt plasate 3 butoane, cu titlurile respective: crCros, crHelp, crNoDrop. La efectuarea unui click pe unul dintre butoane, pe suprafaţa formei cursorul va lua forma cu proprietatea corespunzătoare titlului butonului.

a.3) Clasa **TEdit**: Creaţi aplicaţia VOCALE. Pe suprafaţa formei este plasată componenta *Edit1*. La introducerea textului în cutie, titlul formei va indica numărul de apariţii ale vocalelor în cutie.

a.4) Clasa **AnsiString, TLabel**: Creaţi aplicaţia *Replace*. Pe suprafaţa formei cu titlul "Replace" se plasează 3 componente Edit, un buton cu titlul *Replace* şi o componentă *Label*. În prima componentă *Edit* se va scrie o frază, în a doua un cuvânt (silabă, caracter) care urmează a fi înlocuit, în a treia componentă se va scrie textul cu care urmează a fi înlocuit. La efectuarea unui click pe butonul *Replace*, va avea loc înlocuirea textului.

b) completarea proiectului **Calculatorul** [4, p.33-38] cu noi funcţionalităţi, după cum urmează:

b.1) determinarea valorii absolute;

b.2) extragerea rădăcinii pătrate;

b.3) construirea şirului Fibonnaci;

b.4) determinarea factorialului;

b.5) determinarea soluţiilor reale ale unei ecuaţii de gradul doi.

Modul de lucru

1. Elaboraţi aplicaţiile propuse pentru rezolvarea independentă de la cazul a);
2. Compilaţi şi executaţi aplicaţia **Calculator**;
3. Analizaţi rezultatele obţinute;
4. Completaţi aplicaţia **Calculator** cu noi funcţionalităţi b);
5. Verificaţi rezultatele obţinute, cu executarea;
6. Formulaţi concluziile cu referire la rezultatele obţinute.

Întrebări de verificare

1. Ce se numeşte componentă ?
2. Descrieţi paşii ce necesită a fi parcurşi pentru modificarea proprietăţilor componentei prin intermediul *Object Inspector* şi la nivel de cod.
3. Ce înţelegeţi prin prelucrarea de evenimente ?
4. Ce componente aţi utilizat pe parcursul elaborării lucrării de laborator ?
5. Descrieţi algoritmul de elaborare a unei aplicaţii Windows orientată pe obiecte.
6. Prezentaţi cazuri din viaţa reală când pot fi utilizate aplicaţii Windows orientate pe obiecte.
7. Explicaţi funcţionalitatea unei aplicaţii elaborate în lucrarea Dvs.

Caracteristica seminarelor

În cadrul orelor de seminar s-a efectuat câte un studiu de caz pentru fiecare unitate de învăţare. În cadrul studiului s-au analizat diverse situaţii caracteristice familiei de situaţii unităţii de învăţare (tab. 3.8).

Tabelul 3.8.Corelaţia dintre unitate de învăţare şi studiu efectuat

Unitatea de învăţare	*Proiectul analizat*
Clase şi obiecte	Prelucrarea tipurilor ordinale la nivel de obiecte
Moştenire	Organizarea unei situaţii complexe prin obiecte aflate în relaţie de moştenire
Polimorfism	Posibilitatea creării unor noi tipuri de date, cu diferite funcţionalităţi
Agregare	Prelucrare structurilor dinamice de date la nivel de obiecte
Aplicaţii Windows orientate pe	Corelaţia dintre obiectele create de

obiecte	utilizator şi componentele oferite de BCB
Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date	Avantaje ale utilizării mediului de programare vizuală BCB în gestiunea BD

În cazul în care în planul de studii nu au fost prevăzute ore de tip seminar, studiul se propune a fi organizat în cadrul orelor de laborator.

Propunem spre analiză studiul efectuat pentru unitatea de învăţare

Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date.

În cadrul acestui studiu se propune spre analiză un proiect pentru gestiunea informaţiei despre cărţile unei biblioteci. Proiectul este descris în totalitate în [48, p. 84-94]. Descriem etapele caracteristice studiului:

A. Selectarea şi prezentarea cazului – se prezintă structura bazei de date şi se descriu cerinţele faţă de proiect;
B. Organizarea echipelor de lucru – studenţii se organizează în grupuri (patru grupuri) conform operaţiilor ce necesită a fi efectuate: formulare pentru introducerea datelor, formulare pentru căutare, formulare pentru afişare şi administrare, rapoarte;
C. Prelucrarea şi conceptualizarea – fiecare grup vine cu propuneri asupra modului de structurare a proiectului. Sunt analizate propunerile studenţilor.
D. Structurarea finală a studiului – profesorul prezintă proiectul elaborat, acesta este analizat, comparat cu propunerile studenţilor. Sunt formulate concluzii care vor conţine anumite cerinţe ce necesită a fi respectate în elaborarea de proiecte pentru gestiunea unei baze de date.

3.3 Analiza statistico-matematică a rezultatelor investigaţiei ştiinţifice

Procesul de formare şi dezvoltare a CPOO la studenţii specialităţii ”Informatica” (profil pedagogic) s-a finalizat prin susţinerea examenului în scris. Pentru a demonstra că rezultatele obţinute de către studenţii din eşantionul experimental şi rezultatele obţinute de către studenţii din eşantionul de control au atins niveluri diferite au fost aplicate aceleaşi criterii statistice.

Pentru criteriul Cramer-Welch au fost formulate ipotezele statistice:

ipoteza nulă H_0: nivelul mediu de pregătire în eşantionul de control (eşantionul 1) este apropiat de nivelul mediu de pregătire în eşantionul experimental (eşantionul 2).

ipoteza alternativă H_1: nivelul mediu de pregătire în eşantionul 1 se deosebeşte considerabil în sens statistic de nivelul mediu de pregătire în eşantionul 2.

Tabelul 3.9. Valorile empirice calculate ale criteriilor statistice.

Valoarea T	Eşantioanele
2,88	UST
2,14	USB
3,73	CFBC

Observăm că toate valorile primei coloane T>1,96. Conform criteriului Cramer-Welch (formula 3.1), se respinge ipoteza H_0, ceea ce semnifică că între caracteristicile eşantioanelor există diferenţe semnificative.

Pentru criteriul U a lui Mann-Whitney au fost formulate ipotezele statistice:

ipoteza nulă H_0: Nivelul de pregătire al studenţilor în eşantionul 1 nu este mai mic decât nivelul de pregătire al studenţilor în eşantionul 2.

ipoteza alternativă H_1: Nivelul de pregătire al studenţilor în eşantionul 1 este mai mic decât nivelul de pregătire al studenţilor în eşantionul 2.

Tabelul 3.10. Valorile empirice calculate ale criteriilor statistice.

Valoarea U_{emp}	Valoarea critică U_{cr} 0,05	Eşantioanele
25	26 - 36	UST
40,84	37 - 45	USB
71,5	75 - 86	CFBC

Dat fiind faptul că $U_{cr} > U_{emp}$ (tab. 3.10), se respinge ipoteza H_0. Ambele criterii statistice indică diferenţe semnificative între nivelurile pregătirii studenţilor din eşantioanele supuse experimentului.

Conform modelului şi metodologiei elaborate, CPOO include în structura sa mai multe situaţii, care în mod convenţional le-am grupat în trei categorii: micro-competenţe, competenţe şi macro-competenţe. Pentru

determinarea cotei studenţilor care deţin competenţe, rezultatele acestora au fost măsurate prin prisma a patru nivele propuse de В.П. Беспалько, după cum urmează:

Nivelul I – indică cota studenţilor care nu deţin competenţe, au formate doar cunoştinţe teoretice;

Nivelul II – indică cota studenţilor care deţin *micro-competenţe*, adică pot rezolva situaţii cu apelare de până la cinci resurse;

Nivelul III – indică cota studenţilor care deţin *competenţe*, adică pot rezolva situaţii care necesită a fi tratate prin apelarea a cinci - zece resurse;

Nivelul IV – indică cota studenţilor care deţin *macro-competenţe*, adică pot rezolva situaţii semnificative prin utilizarea unui număr de resurse mai mare decât zece.

Analiza datelor experimentale indică o cotă mai înaltă a studenţilor care deţin competenţe, în grupele experimentale decât în cele de control. Prezentăm rezultatele obţinute:

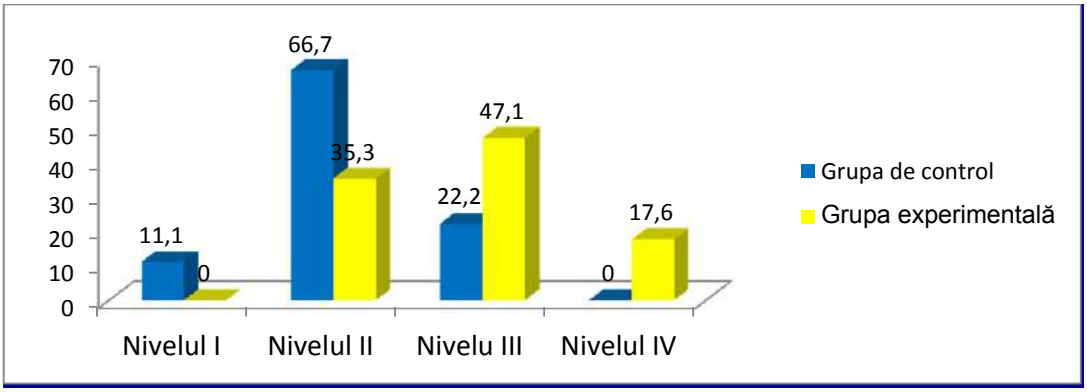


Fig. 3.3. Ponderea studenţilor care deţin competenţe (UST)

Astfel, în grupa experimentală, UST, constatăm un rezultat de 17,6% la **nivelul IV**, 47,1% la **nivelul III** şi 35,3% la **nivelul II**, iar în grupa de control constatăm că ponderea studenţilor care deţin competenţe este mult mai scăzut faţă de nivelul grupei experimentale, acesta fiind de 22,2% la **nivelul III**, 66,7% **la nivelul II** şi 11,1% la **nivelul I**.

Cu toate că rezultatele obţinute de către studenţii din eşantionul experimental sunt mai mari decât cele obţinute de studenţii din eşantionului de control, în cazul USB constatăm că ponderea studenţilor care deţin macro-competenţe este mai mică decât în cazul eşantionului UST. Considerăm că aceasta se datorează lipsei primului modul din cadrul experimentului de formare (formarea şi dezvoltarea CPOO în baza unui limbaj de programare

orientat pe obiecte). Totuşi observăm o diferenţă considerabilă la **nivelul I** de 53,85% pentru grupa de control şi 8,33% în grupa experimentală. Conform acestui rezultat mai mult de 50% din studenţii eşantionului de control nu deţin competenţe POO. Un rezultat diametral opus constatăm la **nivelul II** 15,38% în eşantionul de control şi 58,33% în eşantionul experimental. Constatăm că 30,77 % din studenţii eşantionului de control au **nivelul III** şi 16,67% din studenţii eşantionul experimental. Ca şi în cazul eşantioanelor UST, constatăm că nici un student din eşantionul de control nu a ajuns la **nivelul IV**, iar în eşantionul experimental avem un rezultat de 16,67%.

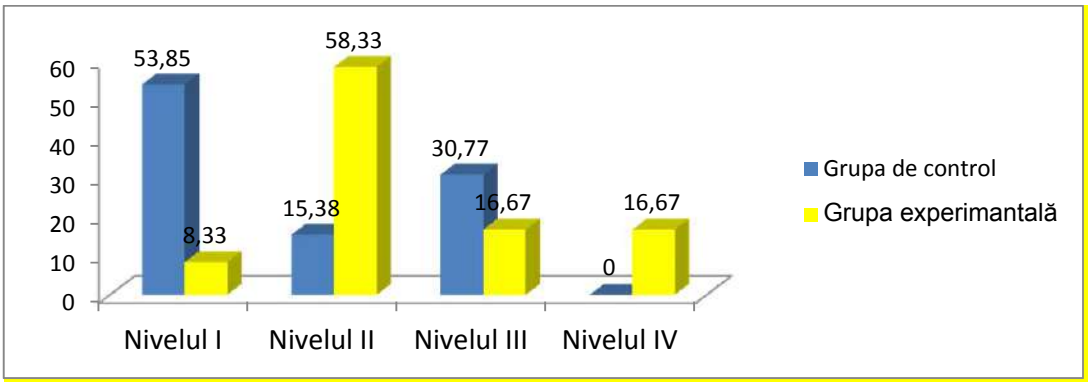


Fig. 3.4. Ponderea studenţilor care deţin competenţe (USB)

Şi în cazul eşantionului CFBC s-au obţinut rezultate care vin să demonstreze eficienţa modelului elaborat. Astfel, în grupa experimentală se constată un rezultat de 29,4% la **nivelul IV**, 29,4% la **nivelul III**, 23,5% la **nivelul II** şi 17,6% la **nivelul I**, pe când în grupa de control nivelul de competenţă la studenţi este în măsură de 11,8% la **nivelul IV**, 11,8% la **nivelul III** 11,8% **la nivelul II** şi 64,7% la **nivelul I**.

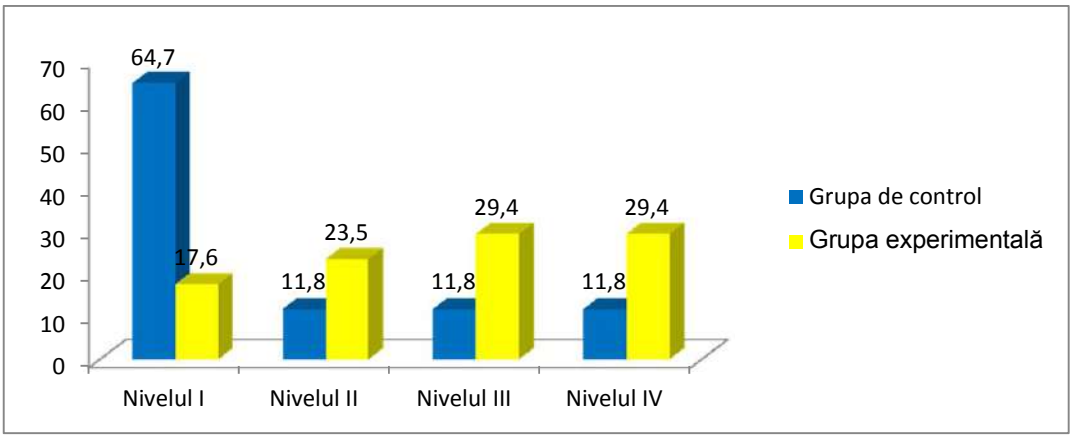


Fig. 3.5. Ponderea studenţilor care deţin competenţe (CFBC)

Suplimentar, în cadrul cercetării noastre:

1. A fost efectuat un studiu în baza căruia au fost analizate rezultatele obţinute de către studenţii specialităţii ”Informatica” în anul academic 2010/2011 cu rezultatele obţinute în anul academic 2011/2012 în cadrul disciplinei ”Programare4”, disciplina în care se studiază tehnologia POO. Pentru compararea rezultatelor obţinute au fost aplicate aceleaşi criterii statistice:

Pentru criteriul Cramer-Welch au fost formulate ipotezele statistice:

ipoteza nulă H_0: nivelul mediu de pregătire în eşantionul 1 este apropiat de nivelul mediu de pregătire în eşantionul 2.

ipoteza alternativă H_1: nivelul mediu de pregătire în eşantionul 1 se deosebeşte considerabil în sens statistic de nivelul mediu de pregătire în eşantionul 2.

Valoarea T obţinută este egală cu 4,81, care este mai mare decât 1,96. Conform criteriului Cramer-Welch, respinge ipoteza H_0 ceea ce semnifică că între caracteristicile eşantioanelor există diferenţe semnificative.

Pentru criteriul U a lui Mann-Whitney au fost formulate ipotezele statistice:

ipoteza nulă H_0: Nivelul de pregătire a studenţilor în eşantionul 1 nu este mai mic decât nivelul de pregătire a studenţilor în eşantionul 2.

ipoteza alternativă H_1: Nivelul de pregătire a studenţilor în eşantionul 1 este mai mic decât nivelul de pregătire a studenţilor în eşantionul 2.

Valoarea critică $U_{14;20}$ este egală cu 83. În urma prelucrării datelor am obţinut U_{emp}=35,5. Aşa cum 35,5<83 se respinge ipoteza H_0.

2. A fost calculat coeficientul de corelaţie (3.6) pentru a compara rezultatele obţinute de către studenţii eşantionului experimental la examenul de disciplina ”Programare4” şi rezultatele obţinute de către ei la examenul de Stat (specializare). Rezultatele sunt prezentate în anexa 17.

$$r = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{(\sum(x_i - \bar{x})^2)}\sqrt{(\sum(y_i - \bar{y})^2)}} \quad (3.6)$$

unde x_i şi y_i reprezintă valorile obţinute

$\bar{x}$, $\bar{y}$ – mediile obţinute pentru fiecare serie.

Prin urmare $r = \frac{15,077}{20,30*14,77} = \frac{15,077}{299,94} = 0,87$

Coeficientul de determinare este apropiat de +1. Rezultă că corelaţia dintre rezultatele obţinute de către studenţii din eşantionul experimental UST, la examenul ”Programare4” şi rezultatele obţinute la examenul de Stat (specializare) este pozitivă.

Altfel spus, ponderea mai înaltă a studenţilor care deţin competenţe POO în grupul experimental *nu este întâmplător. El se datorează aplicării APC în procesul de formare a ei*.

3. A fost realizat un studiu privind gradul de motivaţie al studenţilor din eşantionul UST la cursul de ”Programare4”. În cercetare noastră variabila intermediară este determinată de gradul de motivaţie al studenţilor. R. Viau [170, p.7] defineşte motivaţia drept „un concept dinamic, care are originea în percepţia educatului, a propriei personalităţi şi a mediului înconjurător şi care îl incită să aleagă activitatea, să se angajeze şi să persevereze în realizarea ei pentru atingerea unui obiectiv”. În concepţia autorului menţionat, indicatorii motivaţiei se manifestă prin: angajamentul studentului în soluţionarea unei probleme, prin perseverenţă şi performanţă, iar sursele motivaţiei se regăsesc în percepţia propriei persoane şi în gradul de complexitate a problemei. Pentru măsurarea variabilei intermediare (componentele motivaţiei) a fost utilizat un chestionar propus de către R. Viau şi adaptat specificului cercetării noastre (anexa 2).

Chestionarul este compus din 33 de afirmaţii. Fiecare dintre afirmaţii va fi apreciată de către student prin calificativul: foarte rar – 0, rar – 1, uneori – 2, frecvent – 3, aproape întotdeauna – 4. Afirmaţiile se referă la 11 surse ale motivaţiei: (1) importanţa/sensul activităţii/învăţării; (2) atribuirea reuşitei/succesului; (3) atribuirea nereuşitei/eşecului; (4) anxietatea în situaţie de evaluare; (5) perceperea competenţei sale; (6) scopul urmărit: performanţa; (7) scopul urmărit: învăţarea; (8) scopul urmărit: efortul minim; (9) voinţa de a învăţa; (10) atractivitatea; (11) valoarea intrinsecă a cursului/interesul.

Pentru eşantionul din cadrul UST, chestionarul a fost propus la finele experimentului. Rezultatele (în %), după cum urmează:

1. *Importanţa/sensul activităţii/învăţării* (fig. 3.6). Un factor necesar în formarea şi dezvoltarea de competenţe îl reprezintă utilitatea învăţării, în eşantionul de control aproximativ 59% consideră utilă învăţarea, iar în

eşantionul experimental aproximativ 77% din studenţi consideră utilă învăţarea tehnologiei orientată pe obiecte.

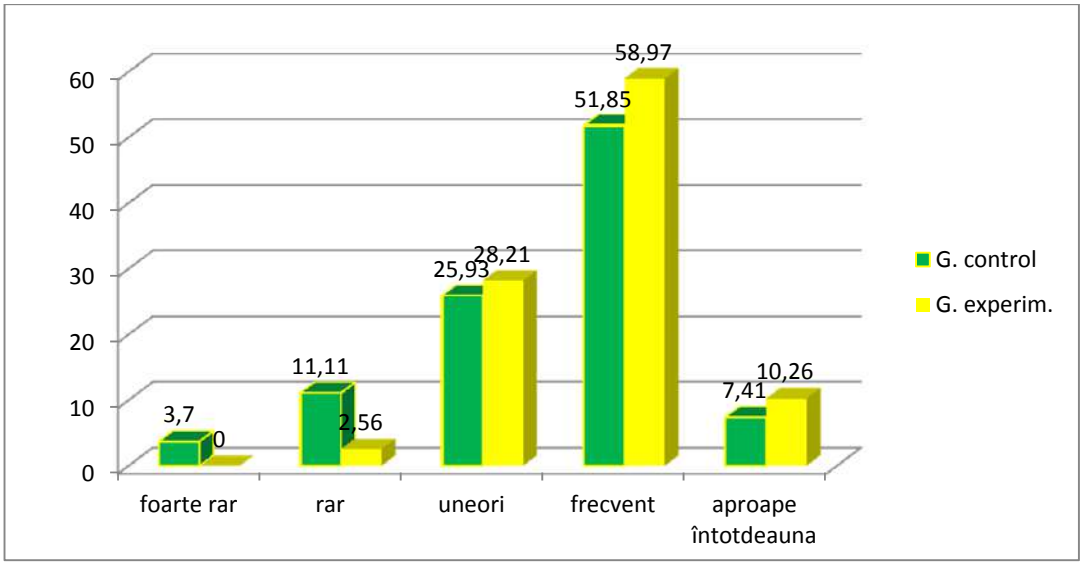


Fig. 3.6. Importanţa/sensul activităţii/învăţării.

2. *Anxietatea în situaţie de evaluare* (fig. 3.7). Datorită gradului de încredere în forţele proprii, în eşantionul experimental numărul studenţilor ce manifestă anxietate în situaţii de evaluare este mai mic faţă de numărul studenţilor din eşantionul de control.

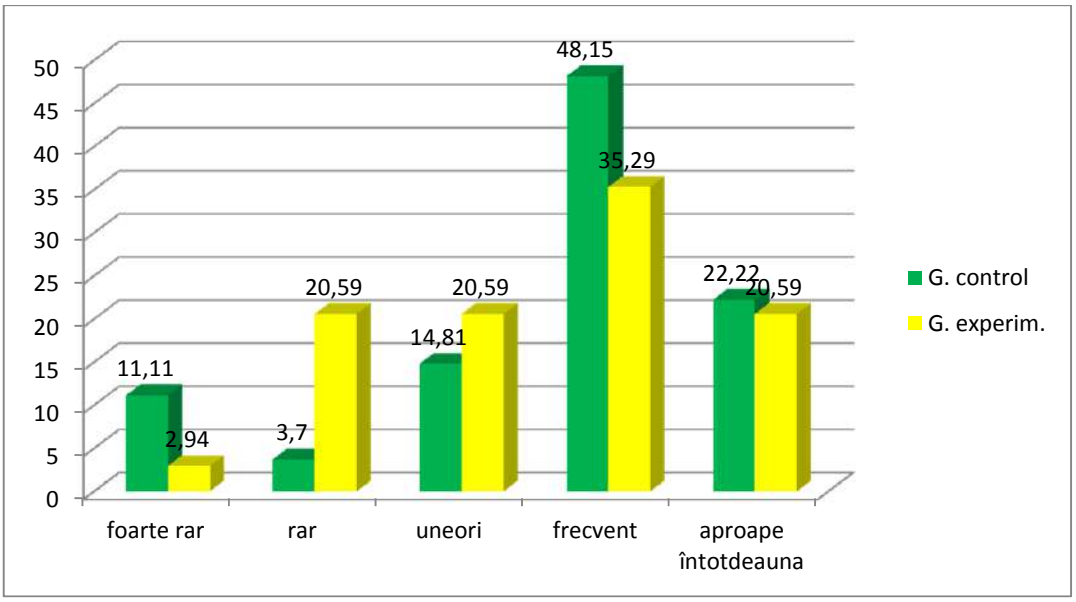


Fig. 3.7. Anxietatea în situaţie de evaluare

3-4. *Atribuirea reuşitei/succesului şi atribuirea nereuşite/insuccesului* (fig. 3.8). Se observă că nivelul de percepţie a succesului/insuccesului în

ambele eşantioane este aproximativ acelaşi: în eşantionul experimental 54% atribuie reuşita unor factori interni (48% din eşantionul de control), iar insuccesele sunt atribuite în proporţie de 56% resurselor interne în eşantionul de experimental şi 48% în eşantionul de control.

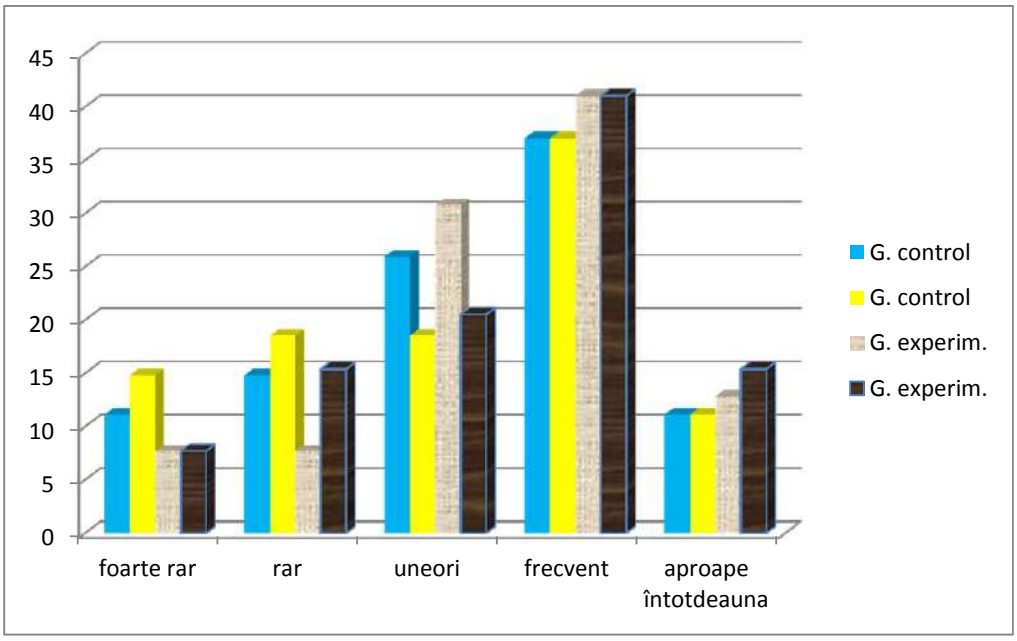


Fig. 3.8. Atribuirea reuşitei/nereuşitei.

5. *Perceperea competenţei sale* (fig. 3.9). Studenţii din eşantionul experimental consideră, în proporţie de 74%, că sunt capabili să soluţioneze cu succes o situaţie problemă, pe când în eşantionul de control nivelul de percepere a propriei competenţe este în proporţie de 67%.

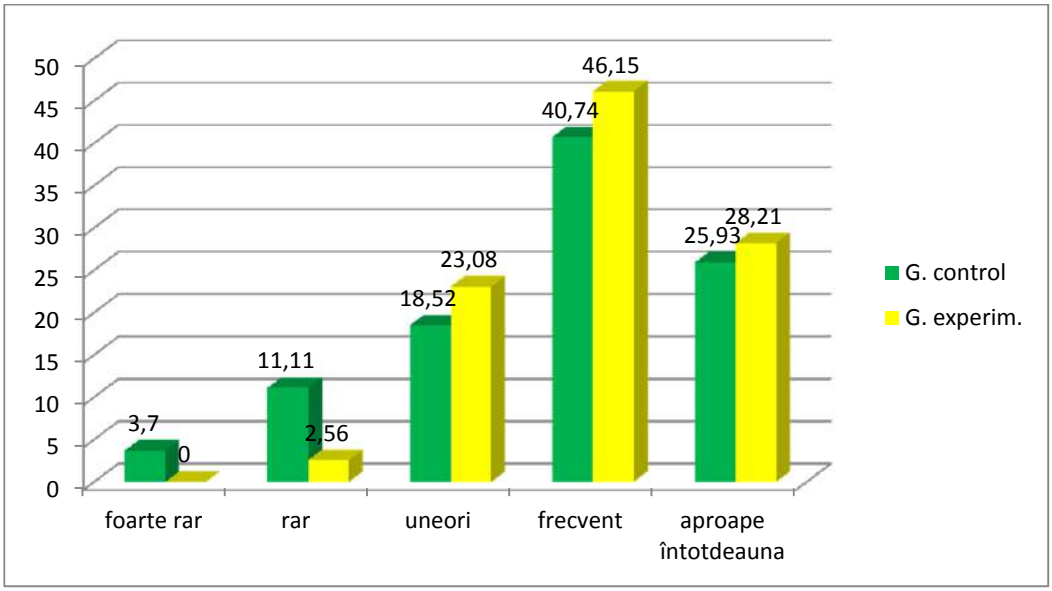


Fig. 3.9. Perceperea competenţei sale

6. *Scopul urmărit: performanţa* (fig. 3.10). În baza rezultatelor obţinute, putem menţiona că scopul urmărit de către studenţi ambelor eşantioane este performanţa (87% în eş. exp. şi 67% în eş. de control).

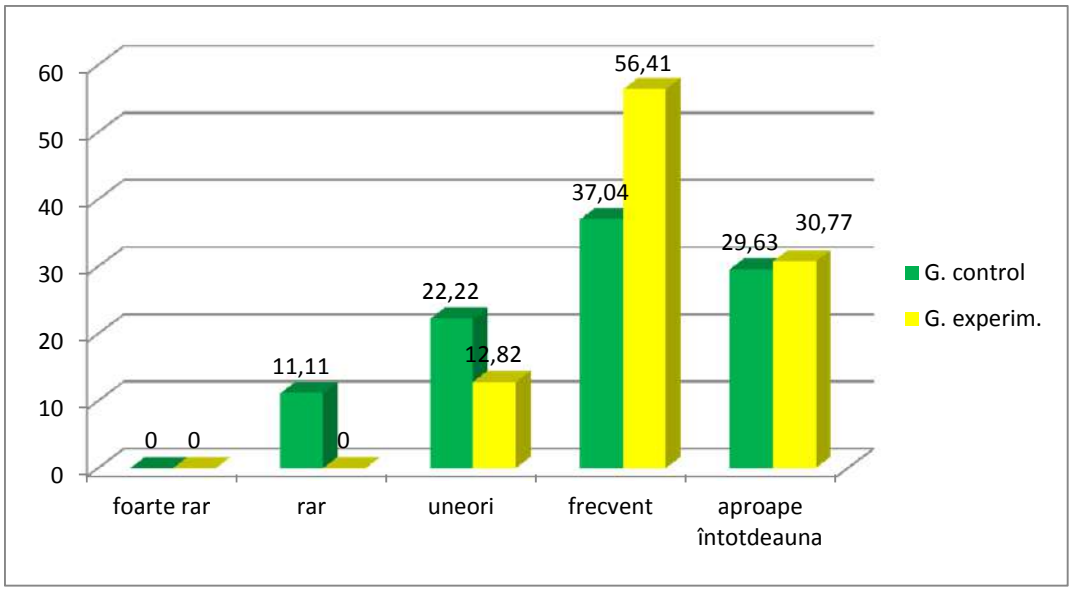


Fig. 3.10. Scopul urmărit: performanţa

7. *Scopul urmărit: învăţarea* (fig. 3.11). Scopul a 82% din studenţii eşantionul experimental este învăţarea faţă de 59% a celor din eşantionul de control.

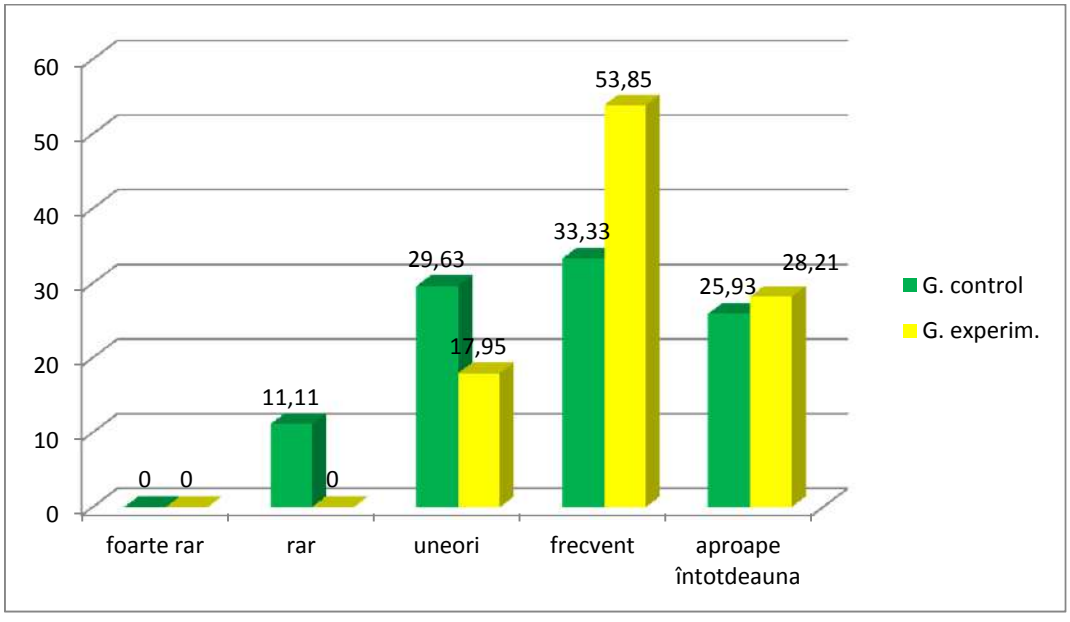


Fig. 3.11. Scopul urmărit: învăţarea

8. *Scopul urmărit: efortul minim* (fig. 3.12). Competenţele nu pot fi formate dacă efortul depus este minim, în acest sens observăm că numărul studenţilor care aproape întotdeauna au ca scop efortul minim este în proporţie de 15% în eşantionul experimental şi 26% în eşantionul de control.

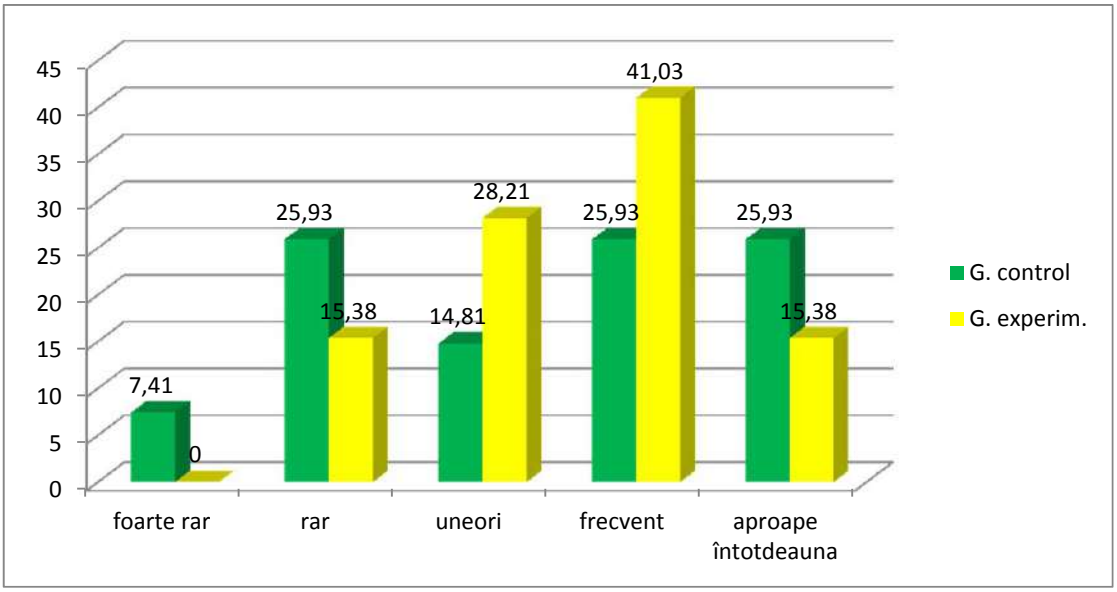


Fig. 3.12. Scopul urmărit: efortul minim

9. *Voinţa de a învăţa* (fig. 3.13). În ambele eşantioane voinţa de a învăţa depăşeşte pragul de 60%. Considerăm că acest nivel se datorează metodelor utilizate în cadrul orelor teoretice. Voinţa de învăţare în eşantionul experimental este prezentă în proporţie de 64%, iar în eşantionul experimental în proporţie de 63%.

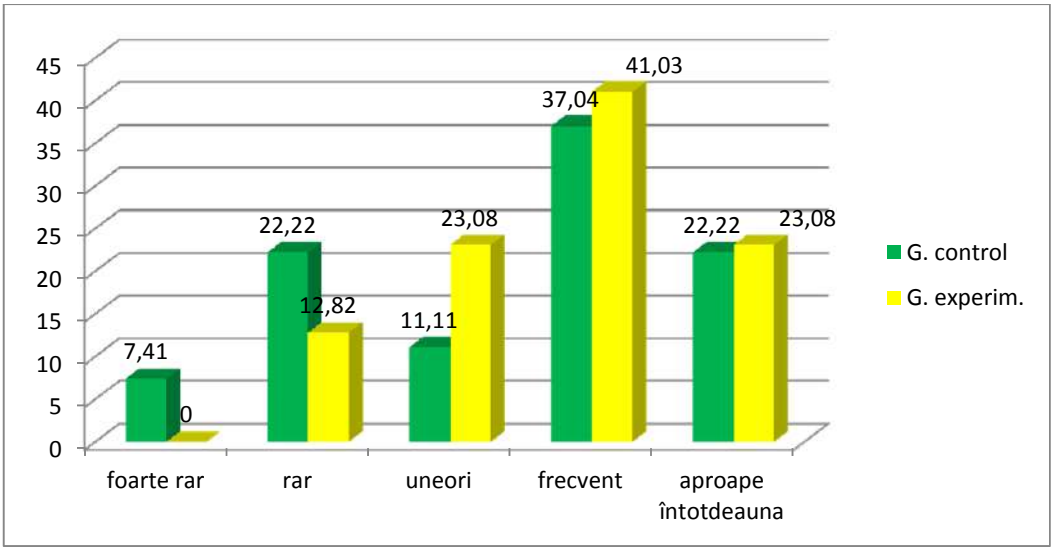


Fig. 3.13. Voinţa de a învăţa

10. *Atractivitatea* (fig. 3.14). Cu toate că gradul de complexitate al exemplelor prezentate studenţilor a fost unul mai înalt, în comparaţie cu alte discipline, 23% din studenţii eşantionului experimental consideră că POO nu este atractivă faţă de 30% din eşantionul de control.

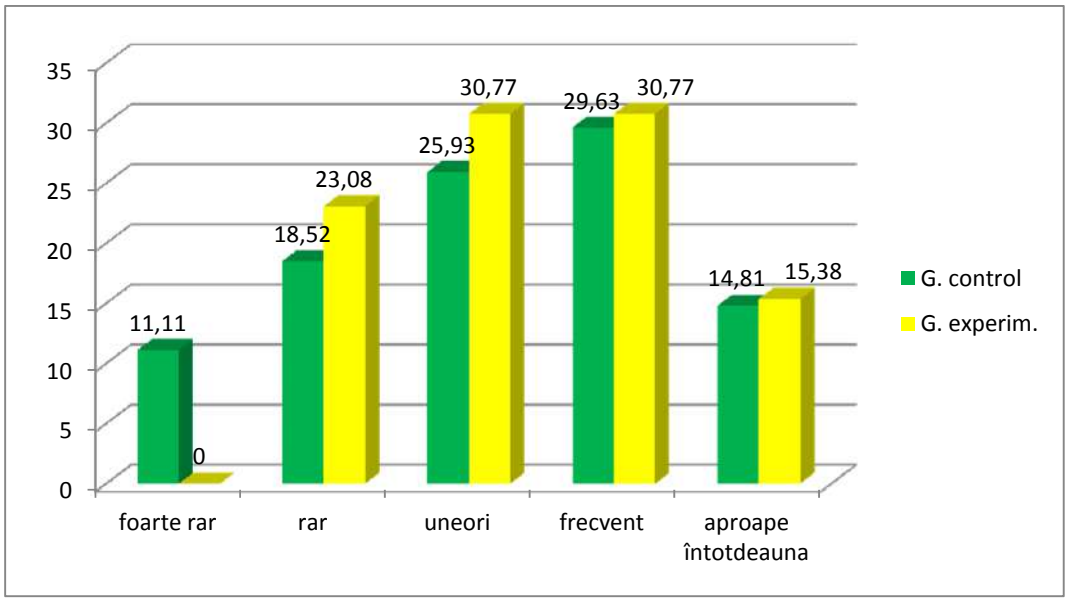


Fig. 3.14. Atractivitatea

11. *Valoarea intrinsecă a cursului/interesul* (fig. 3.15). Tehnologia POO în prezent este utilizată în majoritatea cazurilor la elaborarea produselor software. În acest sens 67% din numărul studenţilor eşantionului experimental manifestă interes sporit faţă de conţinuturile ei, iar în eşantionul de control – 52%.

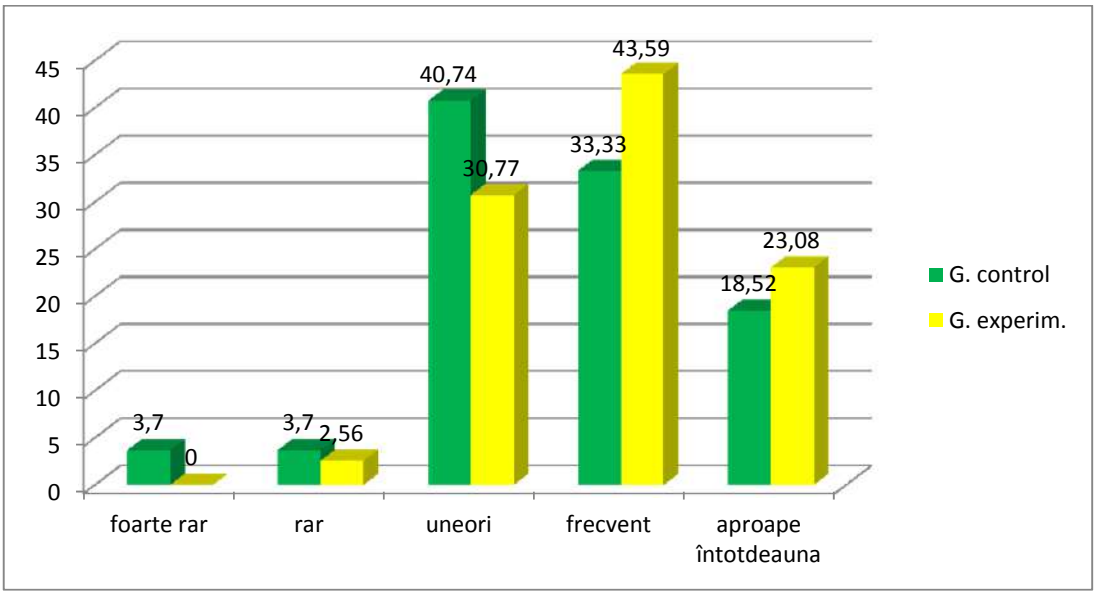


Fig. 3.15. Valoarea intrinsecă a cursului/interesul

Competenţe formate pe parcursul unităţilor de învăţare

Tabelul 1. Competenţe pentru unitatea de învăţare: *Clase şi obiecte*

Unitatea de învăţare	Clase şi obiecte
micro-competenţe	
1) *Descrierea stilurilor de programare;* 2) *Explicarea modului de declarare a unei clase;* 3) *Explicarea noţiunii de date, metode;* 4) *Definirea atributelor (date, metode) unei clase;* 5) *Explicarea conceptului de incapsulare;* 6) *Modalitatea de definire a unei metode;* 7) *Elaborarea de metode pentru efectuarea unei acţiuni;* 8) *Elaborarea de metode pentru efectuarea unei operaţii date;* 9) *Modalitatea de protejare a membrilor clasei;* 10) *Explicarea conceptului de abstractizare;* 11) *Identificarea tipurilor de constructori;* 12) *Descrierea funcţionalităţii constructorului;* 13) *Descrierea funcţionalităţii destructorului;* 14) *Enumerarea avantajelor oferite de funcţii inline;* 15) *Explicarea conceptului de obiect;* 16) *Crearea de obiecte;* 17) *Modalitatea de iniţializare a datelor unui obiect prin intermediul constructorului;* 18) *Realizarea accesului la datele şi metodele unui obiect.*	
competenţe	
1) *Proiectarea unei clase cu o structură redusă;* 2) *Definire domeniu de valori/setului de operatori caracteristici unui obiect;* 3) *Iniţializare datelor obiectului la nivel de constructor;* 4) *Definirea metodelor clasei;* 5) *Prelucrarea obiectului la nivel de metode;* 6) *Gestiunea pointerilor la nivel de date ale unui obiect;* 7) *Gestiunea obiectelor de tip pointer;* 8) *Utilizarea funcţiilor friend pentru realizarea accesului la membrii unei clase;* 9) *Elaborarea de programe cu complexitate medie în stilul orientat pe*	

obiecte.
macro-competenţe
1) *Proiectarea unei clase cu o structură complexă;* 2) *Prelucrarea tablourilor la nivel de obiecte*.

Tabelul 2. Competenţe pentru unitatea de învăţare: *Moştenire*

Unitatea de învăţare	Moştenire
micro-competenţe	
1) *Descrierea tipurilor de moştenire;* 2) *Explicarea conceptului de moştenire;* 3) *Explicarea conceptului de ierarhizare;* 4) *Modalitatea de definire a unei relaţii de moştenire simplă;* 5) *Descrierea caracteristicilor unei clase de bază;* 6) *Descrierea caracteristicilor unei clase derivate;* 7) *Modalitatea de protejare a membrilor clasei în relaţia de moştenire;* 8) *Descrierea funcţionalităţii constructorului şi destructorului în relaţia de moştenire;* 9) *Gestiunea constructorilor cu parametri în relaţia de moştenire;* 10) *Argumentarea necesităţii funcţiilor virtuale;* 11) *Caracteristici ale moştenirii multiple;* 12) *Argumentarea necesităţii utilizării cuvântului virtual în relaţia de moştenire multiplă;* 13) *Modalităţi de redefinire a metodelor;* 14) *Argumentarea necesităţii creării claselor abstracte în relaţia de moştenire.*	
competenţe	
1) *Proiectarea clasei de bază cu o structură redusă;* 2) *Proiectarea clasei derivate cu o structură redusă;* 3) *Crearea unei relaţii de moştenire simplă;* 4) *Crearea unei relaţii de moştenire multiplă;* 5) *Descompunerea unei situaţii de moştenire în clase;* 6) *Definire domeniu de valori/setului de operatori caracteristici fiecărui clase într-o relaţie de moştenire;* 7) *Iniţializare datelor obiectului clasei derivate la nivel de constructor;* 8) *Elaborarea de programe de complexitate medie cu utilizarea*	

conceptului de moştenire; 9) *Implementarea ierarhizării în baza relaţiei de moştenire.*
macro-competenţe
1) *Proiectarea unei situaţii complexe în clase aflate în relaţia de moştenire;* 2) *Elaborarea programelor cu rezolvarea situaţiilor complexe*.

Tabelul 3. *Competenţe pentru unitatea de învăţare:* ***Polimorfism***

Unitatea de învăţare	Polimorfism
micro-competenţe	
1) *Descrierea tipurilor de polimorfism;* 2) *Explicarea noţiunii legare târzie;* 3) *Exemplificarea cazurilor de polimorfism dinamic;* 4) *Argumentarea necesităţii utilizării metodelor virtuale pentru realizarea polimorfismului dinamic;* 5) *Redefinirea metodelor în relaţia de moştenire;* 6) *Explicarea noţiunii de supraîncărcare;* 7) *Exemplificarea cazurilor de polimorfism parametric;* 8) *Descrierea modalităţii de supraîncărcare a operatorilor prin funcţii membru;* 9) *Descrierea modalităţii de supraîncărcare a operatorilor prin funcţii friend;* 10) *Descrierea modului de supraîncărcarea a operatorilor: aritmetici, de comparaţie, de atribuire, de intrare/ieşire;* 11) *Enumerarea restricţiilor specifice supraîncărcării operatorilor*.	
competenţe	
1) *Argumentarea necesităţii implementării conceptului de polimorfism;* 2) *Proiectarea structurii claselor pentru definirea polimorfismului dinamic;* 3) *Prelucrarea obiectelor (aflate în relaţia de moştenire) la nivel de pointeri;* 4) *Crearea polimorfismului dinamic cu utilizarea de metode virtuale pure;* 5) *Motivarea creării unei clase abstracte în relaţia de moştenire;* 6) *Stabilirea modului de supraîncărcare a unui operator;*	

7) *Definire domeniu de valori/setului de operatori caracteristici tipului de date creat;* 8) *Identificarea operatorilor ce necesită a fi supraîncărcaţi;* 9) *Respectarea proprietăţilor operatorilor supraîncărcaţi la crearea unui tip de date.*
macro-competenţe
1) *Rezolvarea unei situaţii complexe cu implementarea conceptului de polimorfism;* 2) *Crearea unui tip de date cu supraîncărcarea operatorilor caracteristici tipului.*

Tabelul 4. *Competenţe pentru unitatea de învăţare:* ***Agregare***

Unitatea de învăţare	Agregare
micro-competenţe	
1) *Descrierea modului de creare a unei funcţii/a unei clase de tip şablon;* 2) *Descrierea avantajelor obţinute în urma utilizării şabloanelor;* 3) *Explicarea conceptului de agregare;* 4) *Prezentarea cazurilor din viaţa cotidiană a relaţiei de agregare;* 5) *Descrierea modului de ierarhizare în baza relaţiei de agregare;* 6) *Proiectarea claselor unei situaţii simple (două clase) de agregare.*	
competenţe	
1) *Utilizarea funcţiilor/claselor template la rezolvarea de probleme;* 2) *Prelucrarea obiectelor prin intermediul altor clase;* 3) *Crearea de obiecte capabile să prelucreze tipuri dinamice de date;* 4) *Prezentarea avantajelor obţinute în rezultatul ierarhizării în baza relaţiei de agregare.*	
macro-competenţe	
1) *Realizarea ierarhizării în baza relaţiei de agregare;* 2) *Crearea şi prelucrarea obiectelor de tip listă*.	

Tabelul 5. Competenţe pentru unitatea de învăţare: *Aplicaţii Windows orientate pe obiecte*

Unitatea de învăţare	Aplicaţii Windows orientate pe obiecte
micro-competenţe	

1) *Descrierea structurii unei componente;* 2) *Descrierea principalelor proprietăţi ale instanţelor clasei: utilizate în procesul de studiu;* 3) *Descrierea principalelor metode ale instanţelor clase utilizate în procesul de studiu;* 4) *Descrierea principalelor evenimente ale instanţelor clase utilizate în procesul de studiu;* 5) *Modalitatea de modificare a proprietăţilor unei componente;* 6) *Modalitatea de prelucrare a evenimentelor unei componente;* 7) *Utilizarea ferestrelor pentru afişare în scopul extragerii unei informaţii;* 8) *Utilizarea ferestrelor pentru intrare în scopul citirii unei informaţii;* 9) *Descrierea algoritmului de creare/salvare/executare a unui proiect.*
competenţe
1) *Descrierea paşilor ce necesită a fi efectuaţi pentru crearea unui proiect;* 2) *Identificarea componentelor necesare pentru elaborarea unui proiect;* 3) *Identificarea proprietăţilor/metodelor/evenimentelor ale instanţei unei clase ce necesită a fi modificate/apelate/prelucrate;* 4) *Modificarea comportamentului unei componente conform datelor problemei;* 5) *Realizarea operaţiilor de intrare/ieşire a informaţiei la nivel de componente;* 6) *Crearea dinamică a componentelor;* 7) *Elaborarea de proiecte cu un nivel de complexitate medie;* 8) *Dezvoltarea proiectelor existente.*
macro-competenţe
1) *Elaborarea de proiecte cu un nivel sporit de complexitate;* 2) *Crearea şi utilizarea obiectelor în cadrul unui proiect.*

Tabelul 6. Competenţe pentru unitatea de învăţare: *Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date*

Unitatea de învăţare	Aplicaţii Windows orientate pe obiecte pentru gestiunea bazelor de date
micro-competenţe	

1) *Descrierea modului de creare a unui alias;* 2) *Descrierea modului de creare a unui tabel, a unei interogări;* 3) *Descrierea principalelor proprietăţi ale instanţelor clasei: din paletele ADO, BDE, DataAcces, DataControls;* 4) *Descrierea principalelor metode ale instanţelor clase utilizate în procesul de studiu;* 5) *Descrierea principalelor evenimente ale instanţelor clase utilizate în procesul de studiu;* 6) *Modalitatea de modificare a proprietăţilor unei componente;* 7) *Modalitatea de prelucrare a evenimentelor unei componente;* 8) *Utilizarea ferestrelor pentru afişare în scopul extragerii unei informaţii;* 9) *Utilizarea ferestrelor pentru intrare în scopul citirii unei informaţii;* 10) *Descrierea algoritmului de creare/salvare/executare a unui proiect.*
competenţe
1) *Descrierea paşilor ce necesită a fi efectuaţi pentru crearea unui proiect;* 2) *Crearea unei legături dintre o componentă şi o sursă (tabel, interogare);* 3) *Identificarea componentelor necesare pentru elaborarea unui proiect;* 4) *Identificarea proprietăţilor/metodelor/evenimentelor ale instanţei unei clase ce necesită a fi modificate/apelate/prelucrate;* 5) *Modificarea comportamentului unei componente conform datelor problemei;* 6) *Realizarea operaţiilor de intrare/ieşire a informaţiei la nivel de componente;* 7) *Crearea formularelor pentru inserare/modificare/vizualizare/excludere a datelor unei BD;* 8) *Elaborarea de proiecte cu un nivel de complexitate medie;* 9) *Dezvoltarea proiectelor existente;* 10) *Elaborarea de rapoarte.*
macro-competenţe
1) *Elaborarea de proiecte pentru gestiunea unei baze de date (cu mai multe de 6 tabele).*

CONCLUZII GENERALE ŞI RECOMANDĂRI

Cercetarea dată vizează formarea şi dezvoltarea competenţei de programare orientată pe obiecte la viitorii profesori de informatică. Analiza datelor obţinute în rezultatul demarării experimentului pedagogic oferă argumente în favoarea confirmării ipotezei cercetării. Rezultatele cercetării sunt expuse prin prisma următoarelor concluzii:

1. A fost argumentată necesitatea revizuirii curriculumului universitar, astfel încât acesta să fie axat pe competenţe pentru fiecare disciplină (conform planului de studii), care va duce în mod direct la formarea de competenţe specifice. Aceasta va permite viitorului profesor de informatică să facă faţă nivelului avansat de informatizare a societăţii.
2. Drept rezultat al analizelor efectuate s-a constat că deşi disciplina programarea orientată pe obiecte figurează în majoritatea planurilor de studii a specialităţilor de informatică, tabla de materii variază de la caz la caz. Diferenţa ţine de unităţile de conţinut şi de mijloacele didactice (limbaje de programare, medii de programare vizuală) utilizate pentru instruire. Frecvent conţinuturile sunt orientate spre studierea unui anumit limbaj şi nu pe mecanismele care stau la baza programării orientate pe obiecte.
3. Cercetările pe care le-am efectuat denotă faptul că competenţa de programare orientată pe obiecte include: cadrul situaţional (familii de situaţii); cadrul de resurse interne/externe; categorii de acţiuni pentru tratarea competentă a unor situaţii concrete dintr-o familie de situaţii, ce ţin de tehnologia programării orientată pe obiecte.
4. S-a constat că formarea şi dezvoltarea de competenţe poate fi realizată prin parcurgerea a cinci etape: familiarizare, structurare, integrare, transfer, extindere. Prin includerea etapelor de familiarizare şi extindere s-au completat unele din variantele existente în literatura de specialitate. Caracteristic etapei de familiarizare sunt aspecte ce ţin de motivarea studentului pentru învăţare, iar etapa de extindere se caracterizează prin dezvoltarea de macro-competenţe în rezultatul tratării competente a unor probleme complexe.

5. S-a demonstrat că formarea de competenţe reprezintă un proces îndelungat, care impune organizarea conţinuturilor disciplinei în unităţi de învăţare. O unitate de învăţare reuneşte resursele necesare pentru formarea de competenţe. La finele fiecărei unităţi de învăţare se evaluează nivelul de formare a competenţelor la studenţi. Conform modelului elaborat în cadrul fiecărei unităţi de învăţare va fi tratată o situaţie complexă în care se vor regăsi majoritatea unităţilor de conţinut studiate.
6. Drept rezultat al cercetărilor efectuate a fost elaborat şi implementat modelul de formare şi dezvoltare a CPOO cât şi instrumentarul metodologic. Astfel, în calitate de mijloace didactice urmează a fi utilizate un limbaj de programare orientat pe obiecte şi un mediu de programare vizuală. Utilizarea cărora impune structurarea unităţilor de învăţare în două module. Datorită specificului programării orientate pe obiecte unităţile de învăţare au fost organizate astfel încât conţinuturile primului modul să se regăsească într-o situaţie complexă, care ulterior va fi adaptată pentru a fi reutilizată în cadrul celui de-al doilea modul.
7. În cadrul cercetării au fost:

- elaborat modelul structural al competenţei de programare orientată pe obiecte;
- elaborat modelul de formare și dezvoltare a competenţei de programare orientată pe obiecte la viitorii profesori de informatică;
- elaborată metodologia de utilizare a modelului;
- formulate șase competențe specifice.

8. În rezultatul demarării experimentului de formare, s-a constat că studenţii eşantioanelor experimentale au un nivel mai înalt de dezvoltare a competenţei de programare orientată pe obiecte decât studenţii eșantioanelor de control, totodată: posedă un nivel mai ridicat al percepţiei propriei competenţe; atribuie eșecului soluţionării unei situaţii-problemă cauze ce ţin de propria personalitate; au drept scop învăţarea, performanţa.

Recomandări practice:

A. *Pentru concepătorii de curriculum*:

a. se recomandă implementarea curriculumului axat pe competenţe profesionale la specialitatea informatica a instituţiilor de învăţământ superior din Republica Moldova;
b. asigurarea continuităţii disciplinelor de specialitate în cadrul specialităţii informatica;

B. *Pentru profesorii de informatică*:

a. studierea tehnologiei POO în liceu la orele facultative;
b. utilizarea modelului elaborat pentru şi dezvoltarea CPOO la studenţii specialităţii Informatica din colegiu;
c. includerea în planul de studii al specialităţii informatica a unor discipline care ar îndruma viitorul profesor de informatică să elaboreze aplicaţii Windows cu caracter didactic.

BIBLIOGRAFIE

1. Adăscăliţei A. Instruire asistată de calculator. Didactică informatică. Iaşi: Polirom, 2007. 203 p.
2. Bianka J. Competenţele profesorilor şi ale directorilor de licee din mediul rural din România şi din alte state din Uniunea Europeană – analiză comparativă. Bucureşti: Ministerul Educaţiei, Cercetării şi Tineretului, 2007. 62 p. [online]. Disponibil pe Internet: http://www.scribd.com/doc/56574118/Competente-analiza-comparativa. (vizat 7.09.2012).
3. Bocoş M., Ciomoş F. Didactica chimiei. Colecţia Didactica pentru toţi. Cluj-Napoca: Editura Eurodidact. 2002. 129 p.
4. Braicov A., Gîncu S. C++ Builder. Ghid de Iniţiere. Chişinău: Tipografia centrală 2009. 196 p.
5. Brăduleac I. Rolul motivaţiei în procesul de predare –învăţare a matematicii În: Învăţământul universitar din Republica Moldova la 80 de ani Volumul II „Probleme actuale ale didacticii matematicii, informaticii şi fizicii, Conferinţa internaţională Chişinău UST 28-29 septembrie, 2010, p. 107-115.
6. Budnic A. Formarea competenţei de comunicare interculturală la viitorii profesori de limbă engleză. Autorf.al tezei de doctor în ped. Chişinău, 2006. 25p.
7. Cabac V. Formarea competenţelor didactice la viitorii profesori de informatică. În: ROMAI Educational Journal, 2008, nr. 3. p. 48-50.
8. Cabac V. Noţiunea de competenţă în cursul universitar Didactica informaticii (I). În: Artă şi educaţie artistică, nr. 2 (5), 2007, p. 125-135.
9. Cabac V. ş.a. Design-ul procesului de învăţare bazat pe abordarea centrată pe student. Bălți: Tipogr. Continental Grup SRL 2011. 144 p.
10. Cerghit I. Metode de învăţământ. Bucuresti: EDP, 1997. 271 p.
11. Cerghit I. Sisteme de instruire alternative şi complementare: structuri, stiluri şi strategii/ Ioan Cerghit. Ed. a 2-a rev. Iaşi: Polirom, 2008. 395 p.
12. Chicu V., Dandara O., Goraş-Postică V. Educaţia centrată pe cei ce învaţă: Ghid metodologic Chişinău: CEP USM, 2009. 134 p.
13. Clocotici V., Stan A. Statistica aplicată în psihologie. Iași: Polirom, 2001. 296 p.
14. Copilu D., Copilu V., Dărăbăneanu I. Predare pe bază de obiective curriculare de formare. Bucureşti: Ed. Ddidactică şi Pedagogică R.A., 2002. 184 p.
15. Cristea S. Dicţionar de ped. Chişinău-Bucureşti, Litera Internaţional, 2000, 400p.
16. Cristea S. Teorii ale învăţării : modele de instruire. București: E.D.P, 2005, 192p.
17. Cucos C. Pedagogie. Prefaţă Adrian Necolau. Iaşi: Polirom, 996. 230 p.
18. Cucoş C. Pedagogie. Iaşi: Polirom, 2006. 461 p.
19. Cucoş C. Pedagogie. Iaşi: Polirom, 2002. 463 p.

20. Deinego N. Testarea adaptivă ca factor de optimizare a procesului de instruire în învăţământul universitar, Autoref. tezei de dr. în ped. Chişinău, 2009. 29 p.
21. Franţuzan L. Formarea competenţei de cunoaştere ştiinţifică la liceeni în context intertransdisciplinar. Autoref. tezei de dr. în pedagogie. Chişinău, 2009. 38 p.
22. Gherghinescu R. Conceptul de competenţă didactică / Competenţa didactică: perspectivă psihologică. Bucureşti: Editura ALL Educaţional, 1999. p.11-23.
23. Gîncu S. Abordări metodice privind formarea competenţei de programare orientată pe obiecte în baza unui mediu de programare vizuală, În: Conferinţa Ştiinţifică Internaţională Conference on applied and industrial mathematics, Chişinău UST, 22-25 august 2012, p.214-223.
24. Gîncu S. Aplicarea decompoziţiei în formarea competenţei de programare orientată pe obiecte" În: Învăţământul universitar din R. Moldova la 80 de ani Volumul II „Probleme actuale ale didacticii matematicii, informaticii şi fizicii, Conferinţa internaţională Chişinău UST 28-29 septembrie, 2010, p. 240-246.
25. Gîncu S. Aplicaţii ale metodei proiectului în formarea şi dezvoltarea competenţei de programare orientată pe obiecte, În: Conferinţa Ştiinţifică Internaţională MATEMATICS & INFORMATION TECHOLOGIES: RESEARCH AND EDUCATION (MITRE-2011), Chişinău USM, 22-25 august 2011, p.174.
26. Gîncu S. Despre unele aspecte ale programării generice, Studia Universitatis, Nr. 9(39)/2010, p. 158-163.
27. Gîncu S. Metodologia rezolvării problemelor de informatică în stilul orientat pe obiecte Ch.: Univ. de Stat Tiraspol, 2012. – 112 p.
28. Gîncu S. Noi tendinţe de dezvoltare a limbajelor de programare orientate pe obiecte din perspectiva viitorului profesor de informatică, Studia Universitatis, Nr. 9(49)/2011, p.190-194.
29. Gîncu S. Reflexii asupra formării competenţei de programare orientată pe obiecte în baza unui limbaj de programare, În: Conferinţa Ştiinţifică Internaţională Conference on applied and industrial mathematics, Chişinău UST, 22-25 august 2012, p.202-213.
30. Gîncu S., Crîşmaru A. Metode de învăţământ din perspectiva formării competenţei de programare orientată pe obiecte. În: Univers pedagogic, nr.1, 2012, p. 25-32.
31. Grădinari G. Formarea competenţelor comunicative ale studenţilor prin Case Study (studiu de caz). Autoref. tezei de dr în pedagogie. Chişinău, 2007. 25 p.

32. Guţu V., Muraru E., Dandara O. Proiectarea standardelor de formare profesională iniţială în învăţământul universitar. Ghid metodologic. Chişinău: CE USM, 2003. 15 p.
33. Ionescu M. ş.a. Strategii de predare şi învăţare. Bucureşti: Edituara Ştiinţifică, 1992. 262 p.
34. Isăchioaia D. Elemente de compoziţie utilizate în lecţiile de limba şi literatura română pentru stimularea creativităţii elevilor" [online]. http://www.scribd.com/doc/47357605/creativitatea-referat (vizat 29.07.2012).
35. Iucu R. B., Instruirea şcolară. Iaşi: Polirom, 2001. 299 p.
36. Jinga I., Istrate E. Manual de pedagogie. Bucureşti: ALL Editional, 2006, p.323-415.
37. Marcus S. Competenţa didactică, perspectivă psihopedagogică. Bucureşti: ALL Educaţional, 1999. 173 p.
38. Masalagiu C., Asiminoaei I. Didactica predării informaticii. Iaşi: Polirom, 2004. 232 p.
39. Mândâcanu V. Profesorul – maestru (în contextul pregătirii iniţiale a cadrelor didactice). Chişinău: Pontos, 2009. 628 p.
40. Mocrienco F., Vlădoiu D. Programarea calculatoarelor, Program postuniversitar de conversie profesională pentru cadrele didactice din mediul rural, 2006, 214 p.
41. Negara C. Coordonatorul TIC: o nouă deschidere în formarea profesorilor de informatică. În: Materialele conferinţei internaţionale „Calitatea învăţământului. Teoria şi practica utilizării Tehnologiilor Informaţionale şi Comunicaţionale în Educaţie". Chişinău, 12-13 martie 2008. Chişinău: MET. 2008. 276 p.
42. Negara C. Formarea iniţială a profesorului de informatică: aspecte didactice. În: Studia Universitas. Seria Ştiinţe ale Educaţiei, 2010, nr. 5(35). p. 120-126.
43. Negara C. Instruirea problematizată ca criteriu al maturităţii profesionale a cadrului didactic. În: ROMAI Educational Journal, vol. 3, 2008.
44. Negara C. Strategii didactice în formarea profesorilor de informatică, tehnologia informaţiei şi a comunicaţiilor, Autoreferatul tezei de doctor în pedagogie Chişinău, 2011. 27 p.
45. Oltean M., Groşan C. Programare în C++ Builder. Cluj-Napoca: Editura albastră, 2008. 400 p.
46. Oprea C. Strategii didactice interactive/Crenguţa-Lăcrămioara Oprea. Ed. a 3-a. Bucureşti: Editura Didactică şi Pedagogică, RA, 2008. 314 p.
47. Oprescu N. Pedagogie. Bucureşti: Editura Fundaţiei „România de mâne ", 1996, 403p.

48. Panţuru S., Păcurar D. Didactica. Braşov: Editura Universităţii "Transilvania", 1999. 235 p.
49. Patraşcu D., Patraşcu L., Mocrac A. Metodologia cercetării şi creativităţii psihopedagogice. Chişinău: Ştiinţa, 2003, 252p.
50. Patraşcu D., Tehnologii educaţionale. Chişinău: ÎSFE-P Tipog. Centrală, 2005, 704 p.
51. Păun E. Pedagogie. Fundamentari teoretice şi demersuri explicative / vol. coord. de Emil Păun şi Dan Potolea. Iaşi: Polirom, 2002. 248 p.
52. Roegiers X. În: Centrul Educaţional PRO Didactica/Programul de Dezvoltare curriculară, Chişinău, 2001.16 p.
53. Sadovei L. Formarea competenţei de comunicare didactică prin modulul pedagogic universitar. Autoref. tezei de dr în pedagogie. Chişinău, 2008. 26 p.
54. Sagoian E. Dezvoltarea culturii comunicaţionale a profesorului în condiţiile şcolii contemporane. Autoref. tezei de dr. în pedagogie. Chişinău, 2005. 24 p.
55. Stan P., Necşoi D. Teorie şi metodologia instruirii. Braşov. 2007. 244 p.
56. Ştefan M. Lexicon pedagogic. Bucureşti: Ed. Aramis Print, 2006. 384 p.
57. Temple Ch., Steele J. L., Meredith, K. S. Aplicarea tehnicilor de dezvoltare a gândirii critice. Ghidul IV. Supliment al revistei „Didactica Pro",2003, nr.2(8).96p.
58. Tudor S. Iniţiere în programarea vizuală - Borland C++ Builder, Editura L&S Informat 2011. 192 p.
59. Vsevolod A., Putină V., Andrieş I. Programarea orientată pe obiecte în baza limbajului C++. Chișinău: CEP USM. 2009. 153 p.
60. Vlada M., E-Learning si software educaţional, Noi tehnologii de e-learning, Conferinţa Naţională de Invăţământ Virtual, Software educaţional, Editura Univ. din Bucureşti, 2003.
61. Zmaranda R.D. Elemente de programare orientată pe obiecte în limbajul C++. Oradea: Editura Universităţii din Oradea, 2001. 257 p.
62. Curriculum pentru cl. a 10-a–a 12-a / Min. Educaţiei al Rep. Moldova. Ch.: Î.E.P. Ştiinţa, 2010 (Tipografia „Elena V.I." SRL). 44 p.
63. Dimensiuni psihologice ale limbajului educaţional, [online]. Disponibil pe Internet: http://www.scritube.com/sociologie/psihologie/Dimensiuni-psihologice-ale-lim19569616.php (vizat 27.07.2012).
64. Elemente de programare structurata, [online]. Disponibil pe Internet: http://www.scribd. com/doc/18643757/14/Elemente-de-programare-structurata, (vizat 08.07.2012).

65. Metode de instruire şi mijloace de instruire [online]. Disponibil pe Internet: http://www.docstoc.com/docs/46539614/METODE-DE-INSTRUIRE-SI-MIJLOACE-DE-ÎNVATAMÂNT (vizat 17.07.12).
66. Mijloace de instruire. [online]. Disponibil pe Internet (vizat 19.07.2012): http://www.scribd.com/doc/52167636/3-Clasificarea-mijloacelor-de-instruire
67. Monitorul Oficial nr. 062 din: 09.11.95 articolul 692, legea învăţământului.
68. Plan de studii, specialitatea Informatica UPSC, [online]. Disponibil pe Internet: http://upsc.md/_Plan_Studii/ITII/Informatica.html, (vizat 03.07.2012).
69. Plan de studii, specialitatea Informatica USB [online]. Disponibil pe Internet: http://www.usb.md/fileadmin/catedre/iati/Plan_licenta_Informatica__profil_pedagogic_.pdf, (vizat 03.07.2012).
70. Plan de studii, specialitatea Informatica USM, [online]. Disponibil pe Internet: http://usm.md/admitere/wp-content/uploads/2012/05/Plan-Ciclul-I-Informatica.pdf, (vizat 03.07.2012).
71. Plan de studii, specialitatea Informatica UST [online]. Disponibil pe Internet: http://bologna.ust.md/s_inf.htm, (vizat 03.07.2012).
72. Un posibil cadru european al calificărilor în perspectiva învățării pe parcursul întregii vieți. Document de lucru al Comisiei Europene. Bruxelles, 2005. 54 p.
73. Аржанов И. Н. Методика обучения объектно-ориентированному проектированию студентов педагогических вузов, Autoref. al tezei de doctor în pedagogie. Санкт-Петербург, 2000. 18 с.
74. Архангельский А. Я. Програмирование в C++Builder 6, Москва, БИНОМ 2003, 1152 с.
75. Атанов Г. А. Деятельностный подход в обучении. Донецк: «ЕАИ – пресс», 2001. 100 с.
76. Беспалько В. П. Основы теории педагогических систем: Проблемы и методы психолого-педагогического обеспечения технических обучающих систем. Воронеж: Изд-во Воронеж. Ун-та, 1977. 304 с.
77. Бобровский С. И. Технологии C++ Builder. Разработка приложений для бизнеса. Учебный курс. СПб.: Питер, 2007. 560 с.
78. Вербицкий А. А. Активное обучение в высшей школе: контекстный подход. Москва: Высшая школа, 1991, 207 с.
79. Вербицкий А. А. Компетентностный подход и теория контекстного обучения. Москва: Исследовательский центр проблем качества подготовки специалистов, 2004. 84 с.

80. Вербицкий А. А. Новая образовательная парадигма и контекстное обучение: монография. Москва: Исследовательский центр проблем качества подготовки специалистов, 1999. 75 с.
81. Вербицкий А. А. Основание для внедрения компетентностного подхода в образование. În: Инновации и эксперимент в образовании, № 3, 2009.
82. Выготский Л.С. Проблемы детской (возрастной) психологии /Выготский Л.С. Собр. соч.: В 6 т. Т. 4. Москва: Педагогика, 1984.
83. Гершунский Б.С. Философия образования для XXI века (в поисках практико-ориентированных образовательных концепций). Москва: Совершенство, 1998. 608 с.
84. Гребнев И.В. Дидактика предмета и методика обучения. Педагогика, 2003, № 1.с.14-21.
85. Грейди Б., Айвар Д., UML. Руководство пользователя. Москва: ДМК-пресс, 2001 г. 432с.
86. Гузеев В.В. Проектное обучение как одна из интегральных технологий. În: Метод проектов. Выпуск 2 / Белорусский государственный университет. Центр проблем развития образования. Республиканский институт высшей школы БГУ. Минск: РИВШ БГУ, 2003. 240 с.
87. Дахин А. Н. Моделирование в педагогике: попытка осмысления, [online]. http://www.bestreferat.ru/referat-78582.html (vizat 09.10.2012)
88. Дахин А. Н. Моделирование компетентности участников открытого общего образования. Autoref. tezei de doctor habilitat în pedagogie. Нижний Новгород, 2012. 46 с.
89. Джеймс Р., Блаха М. UML 2.0. Объектно-ориентированное моделирование и разработка. (2-е изд.). СПб.: Питер, 2007. 544 с.
90. Загашев И. О. Критическое мышление: технология развития. Перспективы для высшего образования/ И. О. Загашев, С. И. Заир-Бек. СПБ: Скифия, 2003. 283 с.
91. Зеер Э.Ф., Шахматова О.Н. Личностно ориентированные технологии профессионального развития специалиста: Науч-метод. пособие. Екатерин-бург, изд-во Урал. гос. проф.-пед. ун-та, 1999. 245 с.
92. Зимняя И. А. Ключевые компетентности как результативно-целевая основа компетентностного подхода в образовании. Авторская версия. Москва: Исследовательский центр проблем качества подготовки специалистов, 2004. 42 с.

93. Иванов Д. А. Компетннции и компетентностная модель современного учителя. [online]. Disponibil pe Internet: http://www.sinergi.ru/DswMedia/doc559.doc (vizitat 11.17.2011).
94. Иванова Д. Н., Малащенко М. В., Пшегусева Г. С. Модель компетенций с учетом квалификационных характеристик специалистов. Ростов-на-Дону, 2007. 21 с.
95. Богатырев, А.И., Устинова, И.М., Теоретические основы педагогического моделирования: сущность и эффективность, [online]. http://www.rusnauka.com/SND/ Pedagogica/2_bogatyrev%20a.i..doc.htm (vizat 09.10.2012).
96. Культин Н. Б. C++ Builder в задачах и примерах. СПб.: БХВ-Петербург, 2005. 336 с.
97. Лебедев О.Е. Компетентностный подход в образовании//Школьные технологии.-2004.-№5.-С.3-12. [online]. Disponibil pe Internet: http://www.orenipk.ru/seminar/lebedev. htm (vizat 10.08.2012).
98. Лапчик М. П. и др. Теория и методика обучения информатике: учебник. Москва: Изд. Центр «Академия», 2008. 592 с.
99. Махмутов М. И. Проблемное обучение. Москва: Педагогика, 1975. 126 с.
100. Мещерякова, Н. А. Формирование информационной компетентности студентов экономический специальностей вузов при обучении объектно—ориентированному программированию Autoref. al tezei de doctor în pedagogie. Омск 2005. 25 с.
101. Новиков А.М., Новиков Д.А. Методология.Москва:Синтег, 2007. 668 с.
102. Новиков, Д. А. Статистические методы в педагогических исследованиях (типовые случаи). Москва: М3-Пресс, 2004. 67 с.
103. Роберт И.В. Современные информационные технологии в образовании дидактические проблемы. Москва „Школа- Пресс”, 1994. 205 с.
104. Рубинштейн С. Л. Основы общей психологии. Москва: Педагогика, 1989. 488 с.
105. Сергеев Р. К., Коротков А. М. О новой парадигме высшего профессионального образования. În: Педагогическая информатика, 2008, № 4. с. 43- 49.
106. Стѐганцев А. Компететностный подход: от профессионального образования к образованию профессионалов. [online]. Disponibil pe Internet: http://www.stiogantsev.ru/ st/biz_komp-podhod.html (vizitat 09.08.2012).
107. Фаулер М. UML. Основы / Пер. с англ. 3-е изд. СПб: Символ-Плюс, 2004. 192 с.

108. Образовательные стратегии и технологии обучения при реализации компетентностного подхода в педагогическом образовании с учетом гуманитарных технологий: методические рекомендации. Изд-во РГПУ им. А. И. Герцена, 2008. 108 с.
109. Stroustrup B. Вокруг C++ „Открытые системы” , № 03, 1998, [online]. Disponibil pe Internet: http://www.osp.ru/os/1998/03/179488, (vizat 08.07.2012).
110. Beckers J. Developper et évaluer des compétences à l‟école: vers plus d‟efficacité et ‟équité. Bruxelles: Labor, 2002. 57 p.
111. Bélair L. Les compétences professionnelles en enseignement et leur évaluation. Ottawa: University of Ottawa Press, 2007. 272 p.
112. Booch G. Object-Oriented Design with Applications. Benjamin/Cummings, Redwood City, California, 2nd edition, 1994. 534 p.
113. Bosman C., Gérard F.-M., Roegiers X. Quel avenir pour les compétences? Bruxelles: De Boeck Université, 2000. 202 p.
114. Boulet A., Zajc L. S., Chevrier J. Les strategies d'apprentissage a l'universite. Quebec: PUQ. 1996. 201 p.
115. Cardelli L. and Wegner P. On Understanding Types, Data Abstraction, and Polymorphism. December 1985. ACM Computing Surveys vol.17 (4). p. 481.
116. Chauvigné C., Vandroz D. Comment former aux compétences ? Modèles de la compétence, modèles de la formation, [online]. Disponibil pe Internet: http://www.inrp.fr/ biennale/7biennale/Contrib/longue/7197.pdf, (vizat 08.08.2012)
117. Chomsky N. Aspects of the Theory of Syntax. Cambridge: The MIT Press. 1965. 251 p.
118. Collès L. Ş.a. Compétences, langage et communication. În: Didactiques des langues romanes. Le développement de compétences chez l‟apprenant. Bruxelles: De Boeck Duculot, 2001. p.17-33.
119. Crahay M. Les compétences: concepts et enjeux. Cahiers du Service de pédagogie expérimentale, n° 21, 2005. p.21-22.
120. Dahl O., Dijkstra E., and Hoare C.A.R. Structured Programming. London: Academic Press, 1972. 83 p.
121. De Ketele, J.-M. En guise de syntèse: convergences autour des compétences. În: Quel avenir pour les compétences? Bruxelles: De Boeck Université, 2000. p. 187-191.

122. Denyer M., ş.a. Les compétences: ou en est-on? L‟application du décret „Missions” en Communauté française de Belgique. Bruxelles: De Boeck Education, 2004. 144 p.
123. Dolz J. et Ollagnier E. (éd.). L’énigme de la compétence en éducation. Bruxelles: De Boeck Université, 2002. 232 p.
124. Ettagebi M. Opertti R. Jonnaert Ph. Logique de compétences et développement curriculaire: débats, perspectives et alternative pour les systèmes. Paris: L’Harmattan,2009. 362 p.
125. Facione P. A. „The Delphi Report” Critical Thinking: A Statement of Export Consensus for Purposes of Educational Assessment and Instruction: Executive summary. Melbrae, CA: California Academic Press, 1990. 20 p. [online]. Disponibil pe Internet: www.insightassessment.com/pdf_files/DEX adobe.pdf (vizitat 24.07.2010).
126. Giblons M. The self-directed learning handbook: Challenging adolescent students to excel. San Francisco: John Wiley and Sous, 2002. 183 p.
127. Henri F., Lundgren-Cayrol C. Apprentissage collaboratif à distance: pour apprendre et concevoir les environnements d’apprentissage virtuels. Québec: PUQ, 2001. 184 p.
128. Henri F., Lundgren-Cayrol C. Apprentissage colaboratif et nouvelles tehnologie. Centre de recherche Licef [online]. Disponibil pe Internet: http://education.devenir.free.fr/Documents/ Apprentissage_colaboratif?et? nouvelles?tehnologies.pdf (vizitat 25.07.2010).
129. Honey P., Mumford A. The Manual of Learning Styles. 3rd ed. Maidenhead: Peter Honey, 1992. 103 p.
130. Johnson D., Johnson R. Cooperative Learning. În: Transforming Education, n^{o}. 18, Winter 1988.
131. Johnson D., W. Johnson R. T., Smith K. A. Cooperative Learning Returns to College. What Evidence Is There That It Works? În: Change, 1998. p. 27-35.
132. Jonnaert Ph. Compétences et socioconstructivisme: un cadre théorétique. Bruxelles: De Boeck Université, 2002. 100 p.
133. Jonnaert Ph., ş.a. Contribution critique au développement de programmes d’études: compétences, constructivisme et interdisciplinarité. În: Revue des sciences de l’éducation, 2005, vol. 30, № 3. p. 667-696.
134. Jonnaert. Ph., ş.a. La compétence comme organisateur des programmes de formation revisitée ou la nécessité de passer de se concept a celui de l‟agir compétent. IBE Working Papers Curriculum Issues,no 4. Geneva: IBE, 2006. 29 p.

135. Kilpatrick W. H. The Project-Method. In.; Teachers College Record Vol. XIX, no. 4 (Sept. 1918), p. 319-335.
136. Knoll M., John Dewey und Projektmethode. Zur Aufklaerung eines Missverstaendnisses. In: Bildung und Erziehung 1992. p. 89-108.
137. Kolb D. A. Experential Lerning – Experience as the source of learning and development. N.-J.: Prentice-Hall, 1984. 256 p.
138. Lasnier F. Réussir la formation par compétences. Montréal: Guérin Éditeur, 2005. 485 p.
139. Le Boterf G. Construire les compétences individuelles et collectives: agir et réussir avec compétence. Paris: Edition d'Organisation, 2000. 206 p.
140. Lebow D. Constructivist values for instructional szstems design: Five principles toward a new mindset. În: Educational Technology and Development, 1993, nr. 42 (3). p. 4-16.
141. Lecheb B. Evaluation des pratiques professionnelles et analyse des pratiques: accompagner le developpement des competences individuelles et collectives (Tome 1), [online]. Disponibil pe Internet: http://www.cpn-laxou.com/Instituts_formation/IFCS/memoires/benjamin.pdf (vizat 08.08.2012)
142. Liskov B. A Design Methodology for Reliable Software Systems, in Tutorial on Software Design Techniques. Third Edition. New York, NY: IEEE Computer Society, 1980. 66p.
143. Masciotra D., Jonnaert P., Daviau C. La competence revisitee dans une perspective situee. [online]. Disponibil pe Internet (vizat 3.02.2010): http://www.ore.uqam.ca/Documentation/ Masciorta/Masciorta02.pdf.
144. Masciotra D., L'agir competent: une approche situationnelle. [online]. Disponibil pe Internet (vizat 3.02.2010): http://www.ore.uqam.ca/Documentation/Masciorta/Masciorta04. pdf.
145. Mellouki M. Débutants en enseignement: quelles compétences? comparaison entre Américains et Québéçois. Québec: Presses Université Laval, 2005. 111 p.
146. Miled, M. Un cadre conceptuel pour l'élaboration des curriculums selon l'approche par les compétences. În: La refonte de la pédagogie en Algérie – Défis en enjeux d'une société en mutation. Alger: UNESCO – ONPS, 2005. p. 125-136.
147. Moon B., Vlăsceanu L., Barrows, L. C. (Eds.). Institutional Approaches to Teacher Education within Higer education in Europe: Current models and New Developments. Bucharest: Editura Enciclopediei, 2003. 342 p.

148. Mulder M., T. Weigel & K. Collins (2006). The concept of competence concept in the development of vocational education and training in selected EU member states. A critical analysis. Journal of Vocational Education and Training, 59,1, p. 65-85, [online]. Disponibil pe Internet: http://www.mmulder.nl/PDF%20files/2007-01-19%20Mulder%20Weigel%20Collins%20 JVET.pdf (vizat 10.08.2012).
149. Paquay L. L'évaluation des compétences chez l'apprenant: pratiques, méthodes et fondements. Louvain: Presses Université de Louvain, 2002. 160 p.
150. Paquay L., ş.a. Former des enseignants professionnels. Quelles stratégies? Quelles competences? 3e édition. Bruxelles: De Boeck Université, 2001. 272 p.
151. Perrenoud Ph. Construire des compétences des l‟école. 3e éd. Paris: ESF, 1997. 125 p.
152. Portelance L. L'évaluation intégrée à la formation par compétences. În: Réfléchir pour évaluer des compétences professionnelles a l„enseignement: deux regards, l'un québécois et l‟autre suisse. Québec: Presse de l‟Universite du Québec, 2008. p. 13-35.
153. Rey B., ş.a. Les compétences à l'école: apprentissage et évaluation. Bruxelles: De Boeck Université, 2006. 160 p.
154. Roegiers X. Analyser une action d‟éducation ou de formation: Analyser les programmes, les plans et les projets d‟éducation ou de formation pour mieux les élaborer, les réaliser et les évaluer. De Boeck Université, 2003. 344 p.
155. Roegiers X. et De Ketele, J.-M. Une pédagogie de l‟intégration: compétences et intégration des acquis dans l‟enseignement. Bruxelles: De Boeck Université, 2000. 312 p.
156. Roegiers X. L'école et l'évaluation: des situations pour évaluer les compétences des élèves. Bruxelles: De Boeck Université, 2004. 367 p
157. Roegiers X., De Ketele, F.-M. Une pédagogie de l'intégration. Compétences et intégration des acquis dans l'enseignement. De Boeck Universt., 2001. 312 p.
158. Roegiers, X. Savoirs, capacités et compétences à l‟ecole: une quête de sens. În: Forum-pédagogies. Mars 1999.
159. Romainville M. L'irrésistible ascension de terme "compétence" en éducation. În: Enjeux, n^{o} 37-38, mars/juin 1996.
160. Rumbaugh J., ş.a. Object-Oriented Modelind and Design. Englewood Cliffs, New Jersey: Prentince Hall. 1991.
161. Samurçay R., Raadel P. Modèles pour l‟analyse de l‟activité et des compétences, propositions. În: Samurçay, R.; Pastre, P. (Dir.). Recherches en didactique professionnelle. Toulouse: Octarès, 2004.

162. Sartori C. Cours de sciences de base – 5 anée Lycee Martin V. Ecole d'Application de l' UCL 3, rue du College 1348 Louvain-la-Neuve, 2001.
163. Scallon G. L'évaluation des apprentissages dans une approche par compétences. Bruxelles: De Boeck Université, 2004. 342 p.
164. Scallon G. L'évaluation des apprentissages dans une approche par compétences. Bruxelles: De Boeck Université, 2004.
165. Seidewitz E. and Stark, M. Towards a General Object-oriented Software Development Methodology. Proceedings of the First International Conference on Ada Programming Language Applications for the NASA Space Station. NASA Lyndon B.Johnson Space Center. TX: NASA, 1986. p.D.4.6.4.
166. Shulman L. Those who understand: Knowledge growth in teaching. În: Educational Research, 1987, nr. 15(2). p. 4-14.
167. Siemens G. Knowing Knowledge. [online]. Disponibil pe Internet: http://www. knowingknowledge.com (vizitat 25.07.2010).
168. Tardif J., Fortier G., Prefontaine C. L'évaluation des compétences. Docum. le parcours de développent. Chènevière Éducation, 2006. 363 p.
169. Touzain G. La contribution de l‟approche par compétence a l‟intégration des apprentissages. În: Reflets, vol. 8, no 1, décembre 1997.
170. Viau R. La motivation en context scolaire. Bruxelles: De Boeck&Larcier, 1997. 221 p.
171. Vidal-Gomel C. Compétences pour gérer les risques professionnels: un exemple dans le domaine de la maintenance des systèmes électriques. În: Travail humain, vol. 70, n^{o} 2, 2007.
172. Walckiers M., De Praetere T. L'apprentissage collaboratif en ligne, huit avantages qui en font un must. În: Distances et savoirs, 2004, nr. 1 p. 53-75.
173. Wittorski P. De la fabrication des compétences. Éducation Permanente no135, 1998/2. 166p.
174. European Commission, Directorate-General for Education and Culture, Basic skills, entrepreneurship and foreign languages. În: Implementation of "Education & training 2010" – work programme, Progress Report, 2003.
175. La formation à l'enseignement. Les orientations. Les compétences professionnelles. Québec: Ministère de l'Education, 2001. 253 p.
176. Learning Theories and Learning Styles. [online]. Disponibil pe Internet: http://dev.twinisles.com/research/learningts.htm (vizitat 12.08.2010).

ILIE LUPU, VALERIU CABAC, SILVIU GÎNCU

FORMAREA ŞI DEZVOLTAREA COMPETENŢEI DE PROGRAMARE ORIENTATĂ PE OBIECTE LA VIITORII PROFESORI DE INFORMATICĂ

Tipografia Universităţii de Stat Tiraspol, Chişinău, str. Iablocichin, 5