25. PROCESE DE AGREGARE SOFTWARE

25.1 Componente software

Tehnicile moderne de programare sunt insoţite de instrumente puternice care au menirea de a creşte productivitatea programatorilor.

Elaborarea de programe, de componente, de secvenţe sau de proceduri necesită efort de programare direct proporţional cu nivelul de complexitate cu care se doreşte a fi înzestrate aceste construcţii.

Activitatea de dezvoltare a aplicaţiilor informatice presupune lucrul în echipă.

Respectarea cerinţelor definite în specificaţii generează consumuri de resurse care sunt atenuate prin:

- creşterea gradului de calificare a personalului;
- acumularea de experienţă;
- utilizarea de instrumente pentru asistarea proceselor de analiză, proiectare, programare şi testare;
- masurarea nivelului de calitate şi efectuarea de corecţii;
- asigurarea unui management adecvat pentru fiecare etapă a ciclului de dezvoltare.

Prezintă un interes special ca la dezvoltarea de aplicaţii informatice noi să fie identificate o serie de componente existente în aplicaţii deja aflate în uz curent, remarcate ca fiind deosebit de performante, componente care să fie reutilizate.

Reutilizarea este un proces care ia proceduri, baze de date, fluxuri şi le integrează în aplicaţii informatice noi, fără a fi necesare modificări radicale.

Reutilizarea presupune că elementele alese să îndeplinească urmatoarele cerinţe:

- să fie corecte din punct de vedere al rezultatelor care trebuie obţinute, caracteristică obţinută printr-un proces de testare deosebit de complex şi mai ales complet;
- să fie validate de practică şi efortul de a scrie o alta componentă echivalentă să nu se justifice;
- să existe dreptul de proprietate sau de utilizare, astfel încât reutilizarea să nu ridice probleme de ordin juridic;
- să fie accesibile fie că text sursă, fie într-o forma integrabilă, pentru a permite evaluarea şi preluarea;
- să conţină parametri de acelaşi tip cu cei din aplicaţia care se construieşte.

Agregarea software este un proces care:

- presupune existenţa de componente software de foarte bună calitate;
- conduce la generarea de componente software prin reutilizare de componente existente prin dezvoltarea de procese de transformare a secvenţelor existente;
- asigură un nivel de complexitate mai redus decât suma complexităţilor componentelor iniţiale;
- realizează componente cu nivel de performanţă mai mare sau egal cu nivelul de performanţă a componentelor iniţiale ca părţi ale unei structuri liniare.

Agregarea de software este obţinută prin procedee mecanice care garantează calitatea produsului final, ca fiind strict dependentă de nivelul calităţii componentelor iniţiale, fără a deteriora acest nivel.

Procesele de elaborare componente software se caracterizează prin:

- durate de execuţie a operaţiilor;
- resurse materiale;
- utilizarea forţei de muncă de înaltă calificare;
- aplicarea unor proceduri de execuţie a operaţiilor;
- masurarea calităţii pentru fiecare componentă;
- estimarea riscurilor şi luarea de măsuri pentru diminuarea efectelor;
- obţinerea unui produs precis definit ca output.

Pentru producţia de componente software procesele includ: limbaje de programare, instrumente de asistare, resurse infinite sau fără uzură fizică.

Procesele de agregare au ca intrari:

- componente software validate de practică;
- instrumente de asistare;
- forţa de munca de înaltă calificare;
- un obiectiv foarte precis definit care spune clar ce rezultate trebuie să ofere noua procedură rezultată din procesul de agregare.

Procesele de dezvoltare de componente şi de agregare trebuie să se bazeze pe planuri de realizare definite prin:

- nivel maxim de resurse de consumat;
- durata maximă acceptată de realizare;
- limita maximă de suportabilitate a costurilor.

25.2 Agregarea pe orizontală

Se consideră componentele software $P_1, P_2, P_3, \ldots, P_N$.

Procesul de agregare pe orizontală pentru aceste componente constă în:

- realizarea unei liste de parametri prin reuniunea listelor de parametri, cel mult cu repoziţionare în vederea obţinerii de subliste omogene;
- concatenarea de secvenţe de instrucţiuni $S_1 \,||\, S_2 \,||\, S_3 \,||\, \ldots \,||\, S_n$ în vederea obţinerii secvenţei agregate S_a.

Agregarea pe orizontală este o reutilizare de secvenţe din N proceduri, obţinând o singură procedură PA.

Dacă există procedurile care stabilesc:

- elementul minim dintr-un sir şi pozitia acestuia;
- elementul maxim dintr-un sir şi pozitia acestuia;

atunci procedura agregată:

- preia că parametri şirul X şi numarul N de componente ale acestuia;
- concatenează secvenţele de alegere a elementelor minim şi maxim;
- concatenaza secvenţele de stabilire a poziţiilor minimului şi maximului.

- initializează un vector de patru componente care returneaza cele patru valori găsite (minimul, maximul, poziţia minimului şi poziţia maximului).

Reprezentarea grafică a procesului de concatenare include:

- cele două proceduri iniţiale;
- procedura rezultat;
- sageţile care indică modul în care se efectuează operaţia de reuniune;
- sageţile care definesc procesul de concatenare.

Agregarea pe orizontală presupune numai operaţii de intersecţie, reuniune, concatenare.

Efortul de a dezvolta agregări pe orizontală este redus, iar riscurile de a obţine pierderi de ordin calitativ, sunt, de asemenea, reduse.

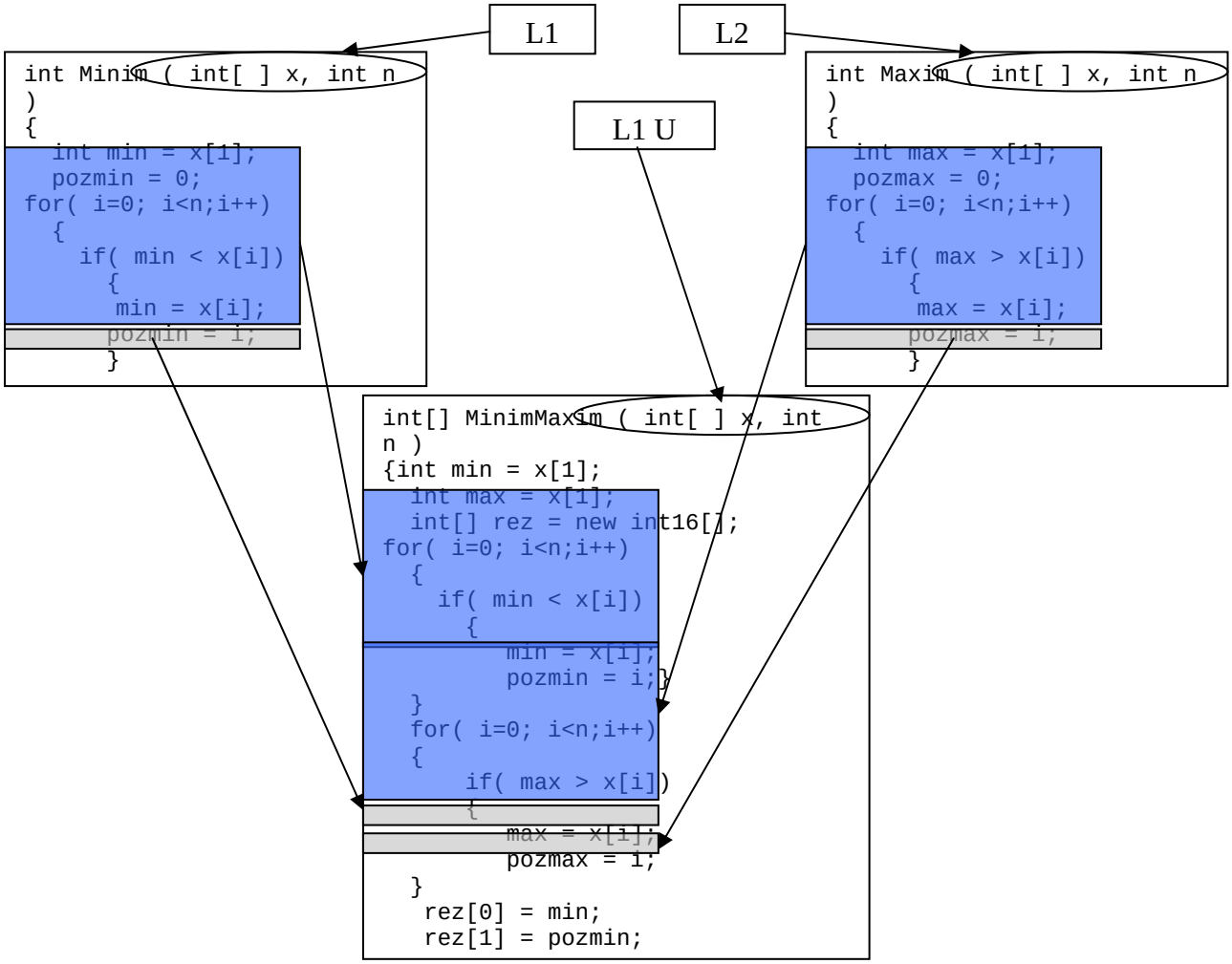

Figura 25.1 Procedura agregata MinimMaxim()

25.3 Agregarea pe verticală

Este un proces mai complex, care presupune reutilizare de software pentru a obţine o componentă cu alte caracteristici.

Programatorul care dezvoltă procese de agregare software pe verticală porneşte de la două sau mai multe proceduri, preia secvenţe din acestea şi le integreaza într-o construcţie proprie.

Noua construcţie trebuie să fie mai performantă decât dacă la execuţia aplicaţiei informatice sunt apelate procedurile aşa cum sunt ele.

Prin agregarea pe verticală:

- se reduce numărul de linii sursă;
- se reduce numărul de instrucţiuni care se repetă;
- se introduc teste pentru selecţia de secvente ce corespund anumitor prelucrari;
- conduce la obţinerea unei construcţii cu grad de complexitate mai ridicat decât complexităţile procedurilor iniţiale;

Dacă se agregă pe verticală procedurile de aflare a minimului şi, respectiv a maximului, se va obţine o procedură în care:

- testele au în comun instrucţiunea de ciclare;

- se introduce un test pentru situaţia în care trebuie ales numai minimul sau numai maximul.

Schema grafică de agregare pe verticală include:

- cele două proceduri iniţiale;
- procedura rezultat;
- fluxurile de includere a secvenţelor existente, pentru alegere;
- fluxul corespunzator structurii repetitive.

Lungimea programului este data ca număr de instrucţiuni sau număr de linii sursă.

Lungimile procedurilor $P_1, P_2, P_3, ..., P_N$ sunt $L_1, L_2, L_3, ..., L_n$, unde L_i reprezintă numărul de linii sursă. S-a avut grijă ca o linie sursă să corespundă unei instrucţiuni sau unei delimitari de bloc.

Textul sursă este:

Lg(lungime secvenţă) = 14 linii sursă (25.1)

- Procedura pentru aflarea minimului – Minim( )

```
1     int Minim ( int[ ] x, int n )
2     {
3     int min = x[0];
4     int pozmin = 0;
5           for( i=0; i<n;i++)
6           {
7                 if( min < x[i])
8                       {
9                             min = x[i];
10                            pozmin = i;
11                      }
12          }
13    return min;
14    }
```

- Procedura pentru aflarea maximului – Maxim( )

```
1     int Minim ( int[ ] x, int n )
2     {
3     int max = x[0];
4     int pozmax  = 0;
5           for( i=0; i<n;i++)
6           {
7                 if( max > x[i])
8                       {
9                             max = x[i];
10                            pozmax = i;
11                      }
12          }
13    return max;
14    }
```

- Procedura concatenată de aflare a minimului şi maximului – MinMax()

```
1     int[] MinimMaxim ( int[ ] x, int n )
2     {
3     int[] rez = new int16[];
4     int min = x[0];
```

```
5     int max = x[0];
6     pozmin = 0;
7     pozmax = 0;
8     for( i=0; i<n;i++)
9     {
10           if( min < x[i])
11          {
12                min = x[i];
13                pozmin = i;
14                }
15           if( max > x[i])
16          {
17                          max = x[i];
18                          pozmax = i;
19          }
20    }
21    rez[0] = min;
22    rez[1] = pozmin;
23    rez[2] = max;
24    rez[3] = pozmax;
25    return rez;
26    }
```

Lg(Minim) = 14 linii sursă (25.2)

Lg(Maxim) = 14 linii sursă (25.3)

Lg(MinimMaxim) = 26 linii sursă- procedura agregată (25.4)

Lg(procedură agregată) <= L(Minim) + L(Maxim) (25.5)

Se calculează şi complexităţile procedurilor Minim() şi Maxim(), MinimMaxim(). Definire operatori şi operanzi:

*e = a + b + c + d + e * f (25.6)*

operanzi: a,b,c,d,e,f - 6 operanzi
operatori: =,+,+,+,+, - 6 operatori*

$C = operanzi * \log_2 operanzi + operatori * \log_2 operatori$ *(25.7)*

Se consideră procedurile pentru:
- adunarea matricelor A şi B pentru a obţine matricea rezultat C;
- scăderea matricelor A şi B pentru a obţine matricea rezultat C;
- transpunerea matricei A;
- înmulţirea de matrice A şi B pentru a obţine matricea rezultat C.

Construirea unei proceduri care să permită evaluarea expresiilor:

C=A+B
C=A'+B
C=A+B'
C=A'+B' *(25.8)*
*C=A*B*
*C=A'*B*

*C=A*B'*
*C=A'*B',*

presupune agregare pe verticală ce include:

- secvenţa de selecţie;
- secvenţele de prelucrare;
- implementarea structurilor repetitive;
- includerea transpunerilor de matrice prin intermediul expresiilor indiciale.

Reprezentarea grafică a procesului de agregare este formată din:

- procedurile de calcul;
- fluxuri de includere;
- fluxuri de reluare pentru expresii indiciale în cazul operaţiilor cu matrice transpuse.

Trebuie găsite modalităţi foarte clare de a asigura echilibrul între lungimea procedurii rezultate şi volumul de prelucrări.

25.4 Ortogonalitatea software agregat

Ortogonalitatea este dependentă de complexitatea software.
Pentru complexitate se ia în considerare modelul HALSTEAD, CH() dat de relaţia:

$$CH(P_r) = n_1 * \log_2(n_1) + n_2 \log_2(n_2) \qquad (25.9)$$

în care se definesc:

- n_1 - numărul de operanzi definiţi în procedura P_r ;
- n_2 - numărul de operatori definiţi în procedura P_r ;

Pentru procedurile independente $P_1, P_2, P_3, ..., P_N$ se calculează complexităţi individuale.

Tabelul nr. 25.1 *Complexitatea procedurii Minim()*

Procedura Minim()	**Operand**	**Frecvenţă**	**Operator**	**Frecvenţă**
int Minim (int[ ] x, int n)	x n	1 1	int [] ()	3 1 2
{	-	-	{	1
int min = x[1];	min x 1	1 1 1	int [] =	1 1 1
pozmin = 0;	pozmin 0	1 1	=	1
for(i=0; i<n;i++)	i 0 n	3 1 1	for () = < ++	1 1 1 1 1
{	-	-	{	1
if(min<x[i])	min x	1 1	if ()	1 1

	i	1	[] <	1 1
{	-	-	{	1
min = x[i];	min x i	1 1 1	= []	1 1
pozmin = i;	pozmin i	1 1	=	1
}	-	-	}	1
}	-	-	}	1
}	-	-	}	1
return (min);	min	1	return ()	1 1
}	-	-	}	1
TOTAL	**-**	**21**	**-**	**31**

C = 246,37, complexitatea pentru procedura de Minim().

Tabelul 25.2 *Complexitatea procedurii Maxim()*

Procedura Minim()	**Operand**	**Frecvenţă**	**Operator**	**Frecvenţă**
int Maxim(int[ ] x, int n)	x n	1 1	int [] ()	3 1 2
{	-	-	{	1
int max = x[1];	max x 1	1 1 1	int [] =	1 1 1
pozmax = 0;	pozmax 0	1 1	=	1
for(i=0; i<n;i++)	i 0 n	3 1 1	for () = < ++	1 1 1 1 1
{	-	-	{	1
if(max>x[i])	max x i	1 1 1	if () [] >	1 1 1 1
{	-	-	{	1
max = x[i];	max x i	1 1 1	= []	1 1
pozmax = i;	pozmax i	1 1	=	1
}	-	-	}	1
}	-	-	}	1
}	-	-	}	1
return (max);	max	1	return ()	1 1
}	-	-	}	1
TOTAL	**-**	**21**	**-**	**31**

C = 246,37 , complexitatea pentru procedura de Maxim().

Tabelul nr. 25.3 *Complexitatea procedurii agregate MinimMaxim().*

Procedura MinimMaxim()	**Operand**	**Frecvenţă**	**Operator**	**Frecvenţă**
int[] MinimMaxim (int[ ] x, int n)	x n	1 1	int [] ()	2 2 1
{	-	-	{	1
int min = x[1];	min x 1	1 1 1	int = []	1 1 1
int max = x[1];	max x 1	1 1 1	int =	1 1
int[] rez = new int16[];	rez	1	int int16 [] = new	1 1 2 1 1
for(i=0; i<n;i++)	i 0 n	3 1 1	for () = ++ <	1 1 1 1 1
{	-	-	{	1
if(min < x[i])	min x i	1 1 1	if () [] <	1 1 1 1
{	-	-	{	1
min = x[i];	min x i	1 1 1	= []	1 1
pozmin = i;	pozmin i	1 1	=	1
}	-	-	}	1
}	-	-	}	1
for(i=0; i<n;i++)	i 0 n	1 1 1	for () = < ++	1 1 1 1 1
{	-	-	{	1
if(max > x[i])	max x i	1 1 1	if () [] >	1 1 1 1
{	-	-	{	1
max = x[i];	max		=	1

	x i		[]	1
pozmax = i;	pozmax i	1 1	=	1
}	-		}	1
rez[0] = min;	rez min 0	1 1 1	[] =	1 1
rez[1] = pozmin;	rez pozmin 1	1 1 1	[] =	1 1
rez[2] = max;	rez max 2	1 1 1	[] =	1 1
rez[3] = pozmax;	rez pozmax 3	1 1 1	[] =	1 1
}	-	-	}	1
return (rez);	rez	1	return ()	1 1
}	-	-	}	1
TOTAL	**-**	**43**	**-**	**60**

C = 589,27 , complexitatea pentru procedura de MinimMaxim().

Pentru ansamblul de proceduri $P_1, P_2, P_3, ..., P_N$ notat şi $P_1, ..., P_N$ care sunt intrări în procesul de agregare, se calculează complexitatea globală CG, dată de relaţia:

$$CG = SUMA(C(P_i)),\ i=1, 2, 3.., N \qquad (25.10)$$

Ortogonalitatea a doua proceduri Pi şi Pj este indicatorul prin care se evidenţiaza cât de diferite sunt instrucţiunile care alcătuiesc cele două proceduri.

Indicatorul de ortogonalitate H() pentru procedurile P_i şi P_j este date de relaţia:

$$H(P_i, P_j) = f(C_i, C_j) \qquad (25.11)$$

unde:

- C_i - complexitatea procedurii P_i;
- C_j - complexitatea procedurii P_j.

Dacă $H(P_i, P_j) = 0$ înseamnă că procedurile sunt identice.

Dacă $H(P_i, P_j) = 1$ înseamnă că procedurile sunt total diferite.

În cazul procedurilor rezultate în procesul de agregare se observă că:

- ortogonalitatea presupune o procedură rezultat şi o mulţime de proceduri iniţiale;
- indicatorul de ortogonalitate corectat este dat de relaţia:

$$H(PA;\ P_1, P_2, ..., P_N) = f(C_a,\ C_1, C_2, ..., C_N) \qquad (25.12)$$

$$H(PA,\ P_1, ..., N) = f(C_a, CG) \qquad (25.13)$$

Este de aşteptat ca atunci când se studiază complexitatea procedurilor rezultate din procesul de agregare să se studieze ortogonalitatea produsului final.

Inegalitatea proceselor de ortogonalitate: ortogonalitatea procedurilor din agregare pe verticală.

Trebuie evidenţiată această inegalitate prin calcule matematice şi experimental.

25.5 Optimizarea în procese de agregare

Optimizarea proceselor de agregare trebuie privită ca proces de îmbunătăţire.

Se definesc criteriile de optim:

- minimizarea complexităţii;
- minimizarea duratei de elaborare proceduri;
- maximizarea calităţii;
- minimizarea duratei de execuţie a programului;
- minimizarea volumului de prelucrări;
- maximizarea ortogonalităţii.

Având în vedere că agregarea este un proces de transformare, optimizarea presupune menţinerea caracteristicilor de calitate şi de performanţă între limite deja stabilite şi deplasarea spre una dintre extreme, când este definit un anumit obiectiv.

Dacă se consideră procedurile P_i şi P_j având nivelurile Nh_i şi Nh_j pentru caracteristica de calitate CL_h, procedura agregată PA_{ij} are nivelul de calitate care indeplineste conditia:

$$NA_h < MIN\{NH_i, NH_j\} \qquad (25.14)$$

Pentru a dezvolta procese de optimizare în agregarea de proceduri P_i şi P_j se iau în considerare:

- eliminarea de subexpresii comune procedurilor P_i, P_j;
- eliminarea de invarianti rezultati din concatenare de secvenţe;
- regruparea de cicluri repetitive;
- lucrul cu variabile elementare;
- eliminarea codului mort;
- reconstruirea expresiilor relaţionale;
- restructurarea secvenţelor de salt condiţional;
- alegerea tipurilor care evită conversiile;
- eliminarea operaţiilor de citire/scriere.

Dacă se consideră procedurile pentru calcul de medii aritmetice, geometrice, armonice ponderate, agregarea pe orizontală conduce la o procedură prin care se calculează toate mediile.

Agregarea pe verticală cu creşterea generalităţii conduce la reducerea numărului de linii sursă şi la reducerea volumului de prelucrări.

Dacă se urmareşte introducerea unor selecţii, se obţine o procedură cu nivel de complexitate intermediară.

25.6 Rezultate experimentale

Se consideră clasele de aplicaţii informatice CL_1, CL_2,...,CL_p. Din fiecare clasă de aplicaţii C_{i_i} se consideră procedurile $P_{i1}, P_{i2}, \ldots P_{ih}$. Se agregă aceste proceduri.

Rezultă proceduri agregate atât pe orizontală cat şi pe verticală. Se calculează complexităţile acestor proceduri. Se calculează ortogonalităţile. Se conchide că efortul de ortogonalitate se justifică.

Rezultatele experimentale presupun:

- crearea de loturi omogene de proceduri;
- stabilirea grupelor de proceduri care intra în procesul de agregare, construind un tabel pe linii cu procedurile iniţiale, independente, iar pe coloane cu procedurile agregate;
- realizarea de agregări pe verticală şi pe orizontală, funcţie de necesităţile aplicaţiilor informatice noi;
- calculul complexitatilor procedurilor iniţiale şi a procedurilor agregate;
- calculul ortogonalităţilor;
- stabilirea de praguri care încadrează procesul de agregare şi-l diferenţiază de procesul de reinginerie a aplicaţiilor informatice.

Organizarea prelucrărilor pentru a obţine rezultate experimentale semnificative presupune:

- stabilirea de proceduri de colectare a procedurilor destinate agregării;
- efectuarea de masurători pe masură ce se constituie baza de proceduri destinate agregării;
- efectuarea de agregări dinamice pe masură ce apar cerinţe în acest sens;
- efectuarea imediată a masurătorilor pentru procedurile agregate;
- preluarea de informaţii şi stabilirea de decizii în vederea utilizării de proceduri agregate;
- stabilirea limitelor procesului de agregare, în ceea ce priveşte criteriile de performanţă ale aplicaţiei informatice în anasamblul ei.

Studierea stabilităţii procesului de agregare se realizează prin:

- constituirea de loturi de proceduri cu grade diferite de omogenitate;
- calcul ortogonalităţii procedurilor agregate din fiecare lot;
- analiza ortogonalităţii între loturi;
- dacă ortogonalitatea nu diferă statistic, rezultă că procesul este stabil.

Stabilitatea se studiază pe clase de proceduri scrise în acelaşi limbaj de programare, chiar cu aceeaşi tehnică.

Extensiile se efectuează din aproape în aproape, spre generalizare atunci când este cazul, dacă ortogonalitatea nu difera semnificativ, pentru:

- proceduri cu grade diferite de omogenitate, scrise în acelaşi limbaj;
- proceduri omogene scrise în limbaje diferite;
- pentru proceduri de complexităţi foarte diferite, scrise în limbaje diferite.

Odată definite complet tipologiile de agregări se crează premise pentru elaborarea de instrumente care să automatieze procesele.

Schemele fluxurilor arată că aceste instrumente sunt realizabile.

Agregarea se planifică, fapt care impune ca procedurile ce fac obiectul agregării să indeplinească cerinţe de calitate, verificate şi mai ales validate în execuţiile curente ale produselor program la beneficiari.

Trebuie să coexiste atât procedurile iniţiale cât şi procedurile agregate.

Referirea unora sau a celorlalte se efectuează strict depinzând de contextul definit de noile aplicaţii informatice aflate în construcţie.

Chiar dacă se lucrează cu proceduri testate şi validate, este deosebit de important ca şi procedura rezultată în procesul de agregare să fie supusă testelor cu aceeaşi rigurozitate cu care au fost testate procedurile iniţiale.

În cazul agregării pe verticală se urmareşte dezvoltarea de procese de optimizare, întrucât, procedura rezultată trebuie să fie performantă chiar dacă îi creşte gradul de complexitate.

Agregarea software este o activitate complexă care presupune:

- reutilizare de software;
- integrare de secvenţe;
- transformări de liste de parametri ;
- transformări de expresii de referire ;
- transformări de expresii indiciale ;
- transformări de expresii relaţionale în sensul creşterii generalităţii acestora.

Studiul proceselor de agregare se aprofundează prin dezvoltarea de biblioteci de proceduri astfel încât apelul procedurilor să conducă la optimizări în execuţia programelor.