**AZƏRBAYCAN RESPUBLİKASI TƏHSİL NAZİRLİYİ**

**AZƏRBAYCAN DÖVLƏT İQTİSAD UNİVERSİTETİ**

**MAGİSTRATURA MƏRKƏZİ**

***Əlyazması hüququnda***

**Əsədova Aynurə Ehtiram**

# "İNSTRUMENTAL PROQRAMLAŞDIRMA SİSTEMLƏRİNİN YARADILMASI TEXNOLOGİYASININ BƏZİ PROBLEMLƏRİ"

***mövzusunda***

**MAGİSTR DİSSERTASİYASI**

**İxtisasın şifri və adı:** **060509 "Kompüter Elmləri"**

**İxtisaslaşma:** **" İnformasiya sistemləri"**

**Elmi rəhbər** **dos. Musayev M.N.**

**Magistr proqramının rəhbəri** **akad. Abbasov Ə.M.**

**Kafedra müdiri** **akad. Abbasov Ə.M.**

**BAKI – 2019**

## MÜNDƏRİCAT

# Giriş

**XX** əsrin səksəninci illərindən başlayaraq kompüterlərin tətbiq olunma sahələrinin genişlənməsi, həll olunacaq məsələlərin xeyli mürəkkəbləşməsi, sistem və tətbiqi proqramların hazırlanmasına sərf olunan proqramçı əməyinin və hazırlanmaya sərf olunan vaxtn həddindən artıq çox olması, tətbiqi proqram paketlərinin generatorlarının, dialoq rejimində işləyən insan – maşın sistemlərinin, avtomatlaşdırılmış informasiya sistemlərinin, infornasiya-axtarış sistemlərinin, hesab-məntiq problem-istiqamətli dialoq sistemlərinin, süni intellekt sahəsində alınan nəticələrin tətbiq olunduğu ekspert sistemlərin, ən başlıcası isə dialoqlu instrumental proqramlaşdırma sistemlərinin, yaradılmasına olan marağı xeyli artırmış oldu. Bununla yanaşı, o vaxt mövcud olan proqramlaşdırmanın avtomatlaşdırılması vasitələri hesab olunan ümumi təyinatlı translyatorlar, redaktorlar, yükləyicilər və digər çoxsaylı sistem proqramları istifadəçiləri kifayət qədər təmin etmirdi. Odur ki, müəyyən sinif proqram məhsulunun hazırlanmasına istiqamətlənmiş xüsusi proqram vasitələrinin yaradılması ciddi bir problem kimi qarşıda dururdu.

Bu problemin həllinə əsas iki yanaşma mövcud idi – texnoloji və instrumental. Texnoloji yanaşma bu problemin həllini modul proqramlaşdırma metodlarının inkişafında, ayrı-ayrı hazırlanmış prosedurların standartlaşdırılmasında, tətbiqi proqramlar paketlərinin layihələndirilməsi və yaradılması prosesinin formallaşdırılmasında, uyğun köməkçi proqram vasitələrinin yaradılmasında görürdü. Texnoloji yanaşmanın cəlbedici tərəfi onun universal olmasıdır. Bu və yaxud digər proqramlaşdırma texnologiyasının effektivliyi yaradılan proqram məhsulunun struktur-funksional xassələri, proqram məhsulunu hazırlayanların sayı və bir sıra digər faktorlarla şərtləndirilməsinə baxmayaraq, hər bir texnologiya, onun problem oriyentasiyasından asılı olmayaraq, kifayət qədər geniş spektr proqram sistemlərinə tətbiq oluna biləndir. Texnoloji yanaşmaya ən uğurlu misal kimi R- texnologiyanı, SDEM/SDSS texnoloji komplekslərini göstərmək olar.

İnstrumental yanaşmanın məqsədi, dialoqlu PL-MS mayak sistemlərin generatorlarının - instrumental proqramlaşdırma sistemlərinin yaradılmasıdır. İstifadəçilərin tələblərini əks etdirən bu və ya digər formal spesifikasiyalara görə generator tələb olunan dialoqlu PL-MS mayak sistemini qurur.

**Mövzunun aktuallığı.** Hal-hazırda dialoqlu proqramlaşdırma sistemlərinin generatorları əsasən problem-oriyentasiyalı xarakter daşıyır. Lakin dialoqlu proqramlaşdırma sistemlərinin əsas funksional, struktur və dil xassələri bir çox tətbiq sahələri üçün ümumi olan faktorlarla müəyyən olunur. Bu səbəbdən də ideyaların diffuziyası baş verir, bəzi ümumi metodlar və prinsiplər aydın görünməyə başlayır.

Bu metodlardan biri dialoqlu proqramlaşdırma sistemləri generasiya olunan zaman bu sistemlərin giriş dilinin yaradılmasından ibarətdir. Bu dillər, baza dili adlanan dilin genişlənmələri kimi yaradılırlar. Dilin genişləndirilməsi bu dilə uyğun anlayışların daxil edilməsi və tətbiq sahəsinin formal modelinin yaradılması ilə yerinə yetirilir. İnstrumental sistem generasiya olunan PL-MS mayak sistemin idarə olunması, yaradılan sistemin giriş dilində tapşırığını qəbul edib bu tapşırığa əsasən məsələnin həlli planını quran, yerinə yetiriləcək modullar ardıcıllığını müəyyən edən vasitələrlə təchiz olunur.

Təsvir olunan generatorun tətbiqi zamanı yaradıcı konkret dialoqlu mayak sistem layihələndirərkən əvvəlcədən sistemin əhatə etdiyi tətbiq sahəsinin uyğun məsələlərinin həlli üçün problem modulları hazırlamalı, həm də tətbiq sahəsinin formal modelini yaratmalıdır. Bundan başqa o, tətbiq sahəsinin formal modelini şərh etmək üçün xüsusi dil yaratmalıdır.

Belə metod bir sıra instrumental proqramlaşdırma komplekslərində, tətbiqi proqram paketləri generasiya edən PRİZ, VİLNÜS, SPORA komplekslərində, o cümlədən, dialoqlu mayak sistemlər generasiya edən DİPS kompleksində də tətbiq olunmuşdur. Bu metodlar və prinsiplər daim təkmilləşdirilir, yeni funksiyalarla genişləndirilir, yeni altsistemlər əlavə olunmaqla imkanları artırılır. Bu səbəbdən də bir neçə onilliyin keçməsinə baxmayaraq bu komplekslərin yaradılması hal-hazırda da aktual olaraq qalır.

**Tədqiqatın məqsədi.** Müasir instrumental proqramlaşdırmanın, o cümlədən proqram təminatının yaradılması metodları və texnologiyalarını öyrənmək, bu sistemlərdə proqramların testləşdirilməsinin psixologiyası və iqtisadiyyatı problemini araşdırmaq, testləşdirmə üsullarının təsnifatını aparmaq, dialoqlu mayak sistemlərinin tətbiq sahəsi modelinin qurulması üçün təlimat hazırlamaq, proqram məhsulunun hazırlanmasının təşkili mərhələləri araşdırmaq, proqram məhsulunun yaradılma prosesinin planlaşdırılmasının və məhsulun layihələndirilməsinin idarə olunmasının təşkili üsullarını öyrənmək tədqiqatın məqsədinə daxildir.

**Tədqiqatın obyekti**. Dialoqlu instrumental proqramlaşdırma sistemləri, bu sistemlər vasitəsi ilə generasiya olunan dialoqlu mayak sistmlər və onların tətbiq sahələri üçün yaradılan modellər, inteqrasiya olunmuş proqramlaşdırma sistemləri, generatorlarda tətbiq sahəsi modelinin qurulması altsistemləri, dialoqlu instrumental proqramlaşdırma sisteminin generasiya vasitələri, proqramların yaradılması texnologiyası, proqram məhsulunun yaradılması prosesinin planlaşdırılmasının təşkili və mərhələləri tədqiqat obyekti kimi araşdırılmışdır.

**Tədqiqatın nəzəri və praktiki əhəmiyyəti**. Müasir instrumental proqramlaşdırmanın problemləri, o cümlədən proqram təminatının yaradılması metodları, proqramların yaradılması texnologiyası, proqramların testləşdirilməsinin psixologiyası və iqtisadiyyatı, dialoqlu instrumental proqramlaşdırma sistemləri və onların generasiyası problemi öyrənilmiş, **PDL** instrumental proqramlaşdırma dili və bu dildə istifadə olunan verilənlərin strukturları gələcəkdə istifadə üçün geniş araşdırılmış, verilənlərin strukturlarının kompüterin yaddaşında mayak qrafı şəklində saxlanılması təklif olunmuşdur. Enən (azalan) yaradılma prosesinin formallaşdırılması üçün **"sadə proqram"** anlayışı daxil edilmiş, testləşdirmə anlayışı isə dəqiqləşdirilmişdir. Proqramların ənənəvi testləşdirilməsi statik və dinamik metodları öyrənilmiş, onların müqayisəli təhlili aparılmışdır. PL-MS mayak sistemin informasiya mühitinin fraqmentlərinin ikitərəfli adlandırılması mexanizmi layihələndirmənin ixtiyari mərhələsində bu

sistem haqqında toplanmış məlumatların tamlığının birqiymətli olduğunu deməyə imkan verir.

**Tədqiqatın elmi yeniliyi.** Müasir instrumental proqramlaşdırmanın, o cümlədən proqram təminatının yaradılması metodları və texnologiyalarının inkişafı, bu sistemlərdə proqramların testləşdirilməsinin araşdırılması, testləşdirmə üsullarının yaradılması, dialoqlu mayak sistemlərinin tətbiq sahəsi modelinin qurulması üçün təlimat hazırlanması, proqram məhsulunun hazırlanmasının təşkili mərhələlərinin yaradılması tədqiqatın elmi yeniliyinə daxildir.

Dissertasiya işinin birinci fəsli – «Müasir instrumental proqramlaşdırmanın problemləri»- sistem proqramlaşdırılmasının əsas istiqamətlərindən biri olan instrumental proqramlaşdırma sistemlərinin yaradılması probleminə həsr olunmuşdur. Proqram təminatının yaradılması metodları tədqiq olunmuş, proqramların yaradılmasının effektiv metodlarından biri olan təkmilləşdirmə metodu, artan və azalan yaradılma prosesləri araşdırılmış və bu metodların müqayisəli təhlili aparılması üçün "sadə proqram" və "mürəkkəb proqramlaşdırma" anlayışları daxil edilmişdir. Proqramlaşdırma dillərində və translyatorlarda çox geniş yayılmış verilənlərin strukturları – sətirlər, massivlər, növbələr, sektorlar, cədvəllər, ağac, qraf, nəzərdən keçirilmişdir. Proqramların yaradılması texnologiyası və proqramların testləşdirilməsinin psixologiyası və iqtisadiyyatı araşdırılmış, proqramların ənənəvi testləşdirilməsinin statik və dinamik metodları araşdırılmış və müqayisə olunmuşdur.

İkinci fəsil - « Proqram məhsulunun layihələndirilməsinin idarə olunması metodları » - sistem proqram təminatının elementləri yaradılarkən, proqram məhsulunun layihələndirilməsinin idarə olunması metodlarının tədqiqinə həsr olunmuşdur. Proqram məhsulunun layihələndirilməsinin idarə olunmasının təşkili prosesi araşdırılmış, texniki məmulatların yaradılması zamanı görülən işlərin analoqu elə proqram məhsulunun hazırlanması zamanı görülən işlərin sırasında olduğu aşkar olunmuşdur. Proqram məhsullarının layihələndirilməsinin idarə olunmasının planlaşdırması hazırlanma, xidmət, sənədlərin buraxılışı (çap edilməsi), sınaq, kömək (himayə etmək), müşayət kimi funksiyaların icrasından

ibarət olduğu qeyd olunmuşdur. Proqram məhsulunun yaradılma prosesinin planlaşdırılmasının təşkili prosesinin yaradılma, sənədləşdirmə, sınaq, istifadəçilərin öyrədilməsi, müşayət mərhələlərini əhatə etdiyi eşkar edilmişdir. Proqram məhsulunun hazırlanmasının təşkili mərhələləri öyrənilmişdir.

Üçüncü fəsil - « Proqram vasitələrinin yaradılmasının dialoqlu instrumental proqramlaşdırma sistemi »- tətbiq oblastının modelinin mayak qrafları ilə şərh olunduğu dialoqlu instrumental proqramlaşdırma sistemlərin bir sinfinə həsr edilmişdir. Belə problem-istiqamətli sistemlər Dialoqlu instrumental proqramlaşdırma kompleksi vasitəsi ilə yaradılırlarlar və dialoqlu mayak sistemlər adlandırılırlar. Baxılan fəsildə instrumental proqramlaşdırma kompleksləri yaradılarkən akademik L. V. Kantoroviç tərəfindən irəli sürülən iribloklu proqramlaşdırma ideyasının tətbiq olunduğu qeyd edilmişdir. **DİPS** instrumental kompleksinin bir çox dünya ölkələrində tətbiq olunan məşhur **PRİZ, SPORA** instrumental kompleksləri ilə eyni sinifdən olduğu qeyd olunmuş və bu komplekslərin müqayisəli təhlili aparılmışdır. Dialoqlu instrumental proqramlaşdırma sisteminin generasiya vasitələrinin tərkibi öyrənilmiş, bu kompleksin dörd altsistemdən ibarət olduğu və bu altsistemlərdən üçünün **PL-MS** sisteminin generasiyası zamanı istifadə olunduğu göstərilmişdir. Bu sistemin generasiya mərhələsində istifadə olunma ardıcıllığı ilə bütün baza vasitələri sadalanmış və bu sistemdə modullararası interfeysin inkişaf perspektivləri və **PL-MS** sisteminin layihələndirilməsi prosesi araşdırılmışdır.

Dissertasiya işinə giriş, üç fəsil, nəticə və təkliflər, istifadə edilmiş ədəbiyyat siyahısı daxildir. Dissertasiya işində 14 şəkildən, 2 cədvəldən istifadə olunmuş və işin ümumi həcmi 87 səhifədə verilmişdir.

# FƏSİL I. Müasir instrumental proqramlaşdırmanın problemləri

## 1.1 Proqram təminatının yaradılması metodları

Sistem proqram təminatı sahəsində aparılan elmi araşdırmalar göstərir ki, hal hazırda da instrumental proqram təminatı elementlərinin yaradılması metodları durmadan inkişaf etdirilir. Çoxsaylı metodların hazırlanmasına baxmayaraq bu metodların yalnız bəziləri praktikada geniş tətbiqini tapır.

Böyük proqram kompleksləri yaradılarkən proqramı yaradanlar iki üsuldan istifadə edirlər. Birinci üsula görə proqramın alqoritmi, yəni srtuktur sxemi yaradılır. İkinci üsul isə birbaşa proqramın yazılmasıdır. Göründüyü kimi, ikinci üsulda, yəni proqramlar alqoritm qurulmadan yazılarkən proqramın strukturu korrekt olmur. Ancaq bilirik ki, qənaətbəxş alqoritmə əsasən yazılmış proqram heç də həmişə yüksək keyfiyyətli proqram hesab olunmur. Baxılan problemin həlli yollarından biri də uyğun yazılar sisteminin yaradılmasıdır. Bu yazılar sisteminin köməyi ilə proqramın formal layihəsinin yaradılması mümkün hesab edilir.

**PDL** – proqramların layihələndirilməsi dili yazılar sisteminin yaradılması üçün təyin olunmuşdur. **PDL** instrumental proqramlaşdırma dilləri qrupuna daxildir və layihələndirmə işlərinin icrası üçün nəzərdə tutulmuşdur.

Proqramlaşdırma və layihələndirmə dillərinin köməyi ilə proqramlaşdırılan məsələlər çoxluğu bir birlərindən xeyli fərqlənirlər, yəni müxtəlif məqsədlərə xidmət edirlər. Ona görə də bu dillərin müqayisəsinin aparılması doğru hesab edilə bilməz. Prosedur – oriyentasiyalı proqramlaşdırma dilləri qoyulmuş məsələnin həlli üçün lazım olan verilənlərin işlənməsini həyata keçirdiyi halda, layihələndirmə işləri üçün təyin olunmuş dillərdə ifadələr aşkar şəkildə verilməyə bilərlər, yaxud elementar konstruksiyalar hesab olunan "matrislərin transponirə edilməsi", yaxud "funksiyanın qiymətinin hesablanması" ola bilər.

Əgər məsələnin həlli alqoritmi prosedur – oriyentasiyalı dildə proqramlaşdırılıbsa, onda layihələndirmə addımı lazım gəlmir.

Layihələndirmə məqsədi ilə istifadə olunan dillər əsasən iki seqmentə bölünür.

- Birinci seqmentə prosedur – oriyentasiyalı dillərin operatorlarının oxşarı kimi yaradılan operatorlar qrupu;
- İkinci seqmentə tətbiq sahəsindən olan məsələlərin şərhi üçün istifadə olunan sintaksis.

Proqramların yaradılmasının effektiv metodlarından biri də addımlı təkmilləşdirmə metodudur. **PDL** dilinin istifadə olunması bu metodla çox yaxşı uzlaşır. Hər bir addım məsələnin "intellektual idarə olunan" komponenti olmaqla, proqramçı məsələnin layihəsini çox dəqiqliklə düşünür.

Əvvəlcə proqramçı məsələni məsələlər toplusu kimi təsəvvür edir:

Do task A;

Do task B;

Do task C;

Hər bir məsələ spesifikasiyalar vasitəsi ilə müəyyən olunur və detallaşdırılır. Hər bir çox da böyük olmayan məsələni, hər hansı bir prosedura daxil olan **PDL** dilinin bir neçə cümləsi kimi təsəvvür etmək olar. Əgər məsələ mürəkkəbdirsə, onda onu ayrıca prosedur şəklində təsəvvür etmək olar.

Hər bir məsələ detallaşdırılarkən yalnız **PDL** dilinin operatorlarından istifadə etmək olar. Bu dilin operatorlarının seçilməsi təsadüfi deyildir – onlar layihələndirilən proqramların idarə olunan konstruksiyalarını təmin edir. Bu zaman proqramlara funksiyalar kimi baxılır. İxtiyari operator y = f(x) şəklində təqdim oluna bilər. (Çoxsaylı dəyişənləri olan məsələlərdə x və y uyğun sayda komponenti olan vektorlardır). Beləliklə, mənimsətmə operatoru

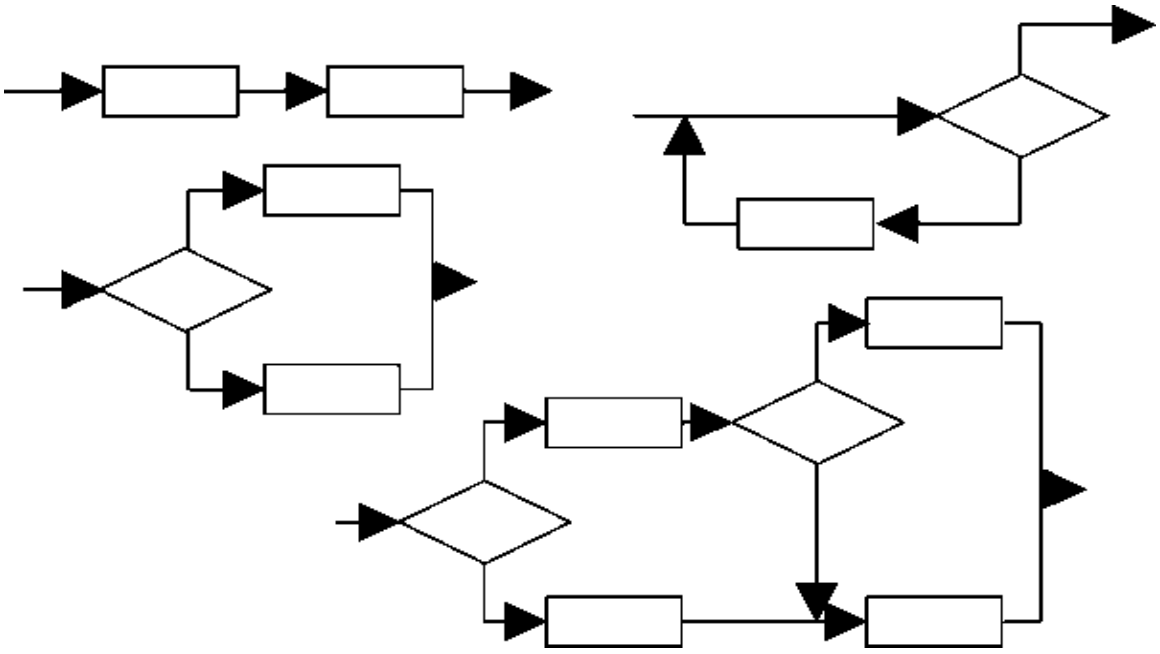
A = B + C * D

F(X, Y, Z) = X + Y * Z şəklində göstərilə bilər və o A = F(X, *Y,* Z) operatoru ilə əvəz oluna bilər.

İxtiyari çətinlikli operatoru analoji şəkildə təqdim oluna bilər. Beləliklə, hər bir proqramı ilkin verilənləri çıxış verilənlərinə çevirən funksiya kimi yazmaq olar.

Enən (azalan) yaradılma prosesinin formallaşdırılması üçün **"sadə proqram"** anlayışı daxil edilir. ***Sadə proqram*** aşağıdakı xassələrə malik struktur sxemi şəklində təqdim oluna bilən proqram kimi müəyyən olunur:

1) yalnız bir daxil olan və bir çıxan til vardır;
2) hər bir düyün (təpə nöqtəsi) üçün bu düyündən keçən daxil olan tildən çıxan tilə gedən yol vardır (şək. 1.1)



Şəkil 1.1 Sadə proqramlara misal

Sadə proqram anlayışının müəyyən olunmasından istifadə edərək enən yaradılmanı aşağıdakı alqoritm şəklində göstərmək olar:

Tutaq ki, proqram bir funksional düyünlə verilmişdir;

**Do while** (layihələndirmə yekunlaşmamışdır);

Hər hansı bir düyünü sadə proqramla əvəz etməli;

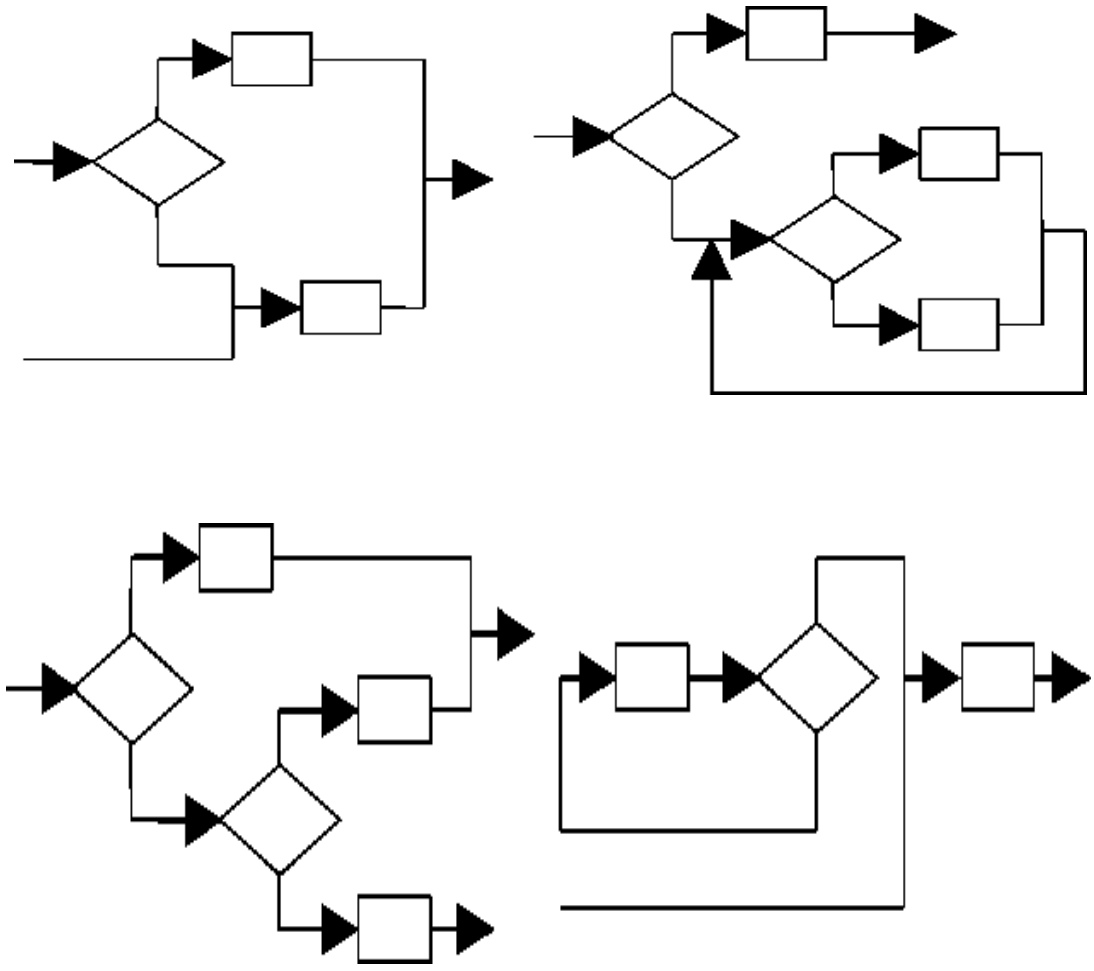
**end**;

Bu alqoritm **goto** operatorundan istifadə etməyə imkan vermir və proqramçı yaradılmaya adi halda olduğundan daha çox vaxt sərf etmiş olur.

***Elementar proqramı,*** birdən çox düyündən ibarət sadə proqram saxlamayan sadə proqram kimi təyin edək.

Adətən proqramçı, proqramı analiz edərək, ayrı ayrı operatorları öyrənir və opeatorları seçərək onları birləşdirir. Proqramın belə öyrənilməsi prosesi addımlarla təkmilləşdirmə metoduna diametrial olaraq əksdir. Hər hansı bir

funksional düyün mütləq mənimsətmə operatoru olacaq, və onun funksiyasını müəyyən etmək nisbətən sadədir. Əgər çox da böyük olmayan sayda düyün elementar proqramda birləşirsə, onda onların funksiyalarını da müəyyən etmək çətin deyildir. **İf – then – else** konstruksiyası üç düyündən, bir predikat və iki funksiyadan ibarətdir. Bu konstruksiya üçün özündə **İf – then – else** operatorunun **then** və **else** hissələrinə uyğun olan funksiyaları saxlayan funksiya müəyyən etmək olar. Mürəkkəb proqramların öyrənilməsi onların nisbətən kiçik hissələri haqqında məlumatların birləşdirilməsinə əsaslanər.



Şəkil 1.2 Sadə olmayan proqramlara misal

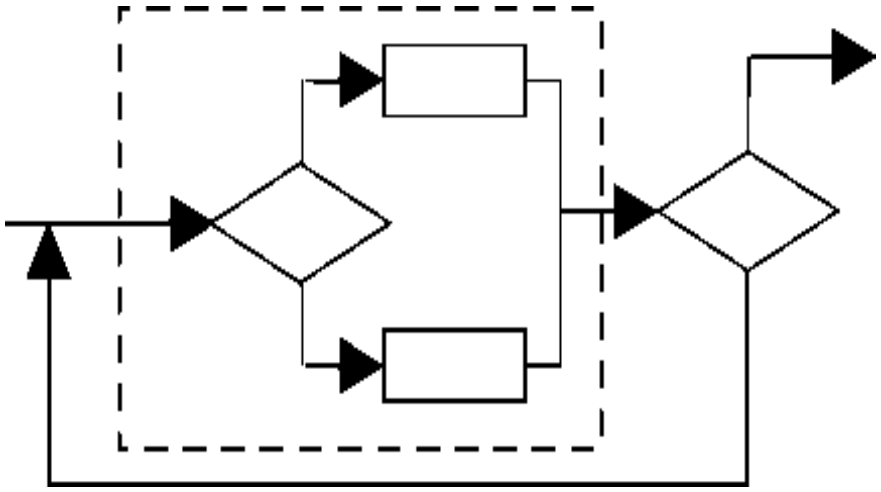
Bir neçə **GOTO** operatorunun olduğu proqrama baxaq. Belə olan halda bütün proqram (yaxud onun böyük bir hissəsi) elementar, yəni çox sadə ola bilər. Beləliklə, proqramın təhlilinə bütün düyünləri nəzərdən keçirməklə başlamaq olar,

çünki onda sadə olmayan proqramlar ola bilməz.

Adətən struktur layihələndirmə üçün aşağıdakı operatorlardan ardıcıl istifadə olunur: **if – then – else, while – do**. Göstərilən idarəedici strukturlar proqramçılara sadə proqramlar (giriş verilənlərini çıxış verilənlərinə çevirən funksiya) yaratmağa kömək edir. Tutaq ki, *f(x)* - **if – then – else** operatorundan ibarət proqram seqmentidir (şək. 1.3).

**if** *p(x)* **then** *g(x)* **else** *h(x)*

İF – THEN - ELSE

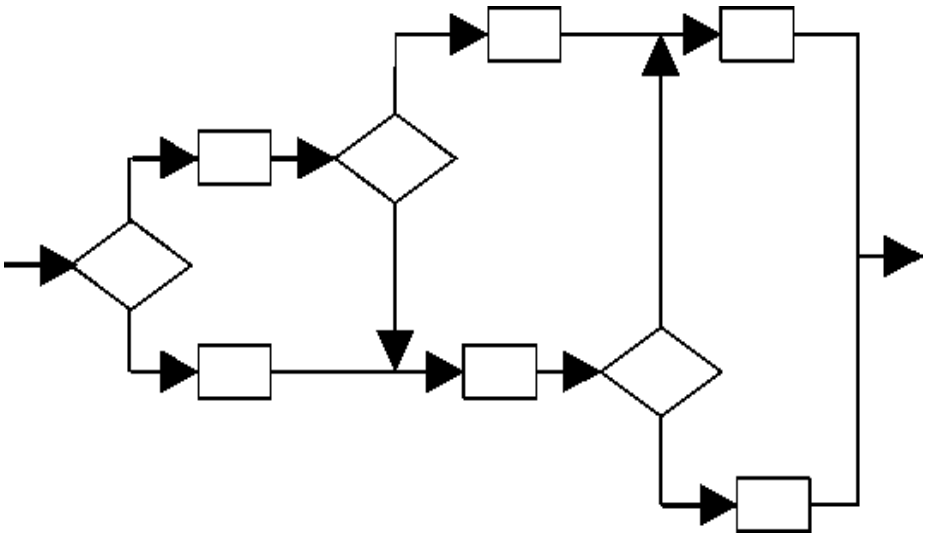


Şəkil 1.3 Elementar proqramlara bölünmə

*g(x)* və *h(x)* funksiyaları *f(x)* funksiyasına nisbətən daha sadədirlər; beləliklə, onların spesifikasiyaları da nisbətən sadə olmalıdırlar. Əgər onların spesifikasiyaları məlumdursa, onda *f(x)* funksiyası a.ağıdakı kimi müəyyən olunur:

$$f(x) = (p(x)\ \ g(x))\quad (p(x)\ \ h(x))$$

Proqramçı nisbətən sadə *g(x)* və *h(x)* bilməklə, formal olaraq *f(x)* funksiyasını müəyyən edə bilər (şək. 1.4).



Şəkil 1.4 Mürəkkəb proqram

**PDL** dilində ən az sayda düyün saxlayan elementar alt proqramlardan istifadə olunur**. İf** və **do while** operatorları minimum toplu hesab olunurlar. İsbat olunmuşdur ki, ixtiyari proqram funksiyası iki idarəedici srtukturu göstərməklə proqramla verilə bilər. Lakin onların əvəzinə başqa konstruksiyalardan da istifadə etmək olar, məsələn, **repeat – until.**

**repeat**

**Operatorlar toplusu;**

**until (ifadə);**

Bu proqram hissəsi aşağıdakına ekvivalentdir:

**Operatorlar toplusu;**

**do while (ifadə);**

**Operatorlar toplusu;**

**end;**

Bundan başqa, belə funksiyaları ixtiyari sayda düyünlərlə **do – case** operatoru da reallaşdıra bilər. İdarəni ötürmək üçün leave operatoru təyin olunmuşdur. Bu operatorun istifadəsi zamanı idarənin ötürülməsi "verilmiş funksiyanın yerinə yetirilməsini dayandırmalı" şəklini alır.

Qeyd edək ki, "GOTO operatorlarının olmaması" deyil, "intellektual idarə oluna bilmə" layihələndirmə prosesinin hərəkərverici qüvvəsidir.

Proqram təminatının yaradılmasının bütün mərhələlərində verilənlər toplularından istifadə olunur. Mənası nəzərə alınmadan ixtiyari işarələr toplusu **verilənlər** adlanırlar. Verilənlər adətən onlara mənimsədilən məna məlumdursa, hər hansı bir informasiyanı əks etdirirlər. Lakin proqramlaşdırmada, xüsusilə də sistem proqramlaşdırmasında bilavasitə verilənlərlə əməliyyat aparılır. Məsələn, hər hansı bir mətnlərin saxlanılması və axtarılması sistemini yaradarkən proqramçı onun məzmununu bilməyə bilər. Onun əsas məsələsi – yaddaşdan səmərəli istifadənin təmin, verilmiş əlamətinə görə tələb olunan mətnin etibarlı şəkildə saxlanılması və tez bir zamanda axtarılıb tapılmasıdır. Bu məsələni həll etmək üçün verilənlər kimi baxılan mətnlərin miqdarı xarakteristikalarını bilmək kifayətdir.

Proqramlaşdırma dillərində və translyatorlarda tez tez müəyyən şəkildə təşkil olunmuş verilənlər toplularından istifadə olunur. Müəyyən şəkildə təşkil olunmuş verilənlər toplusu **verilənlərin strukturu** adlanır. Hər bir struktur elementlərdən yaxud yazılardan ibarət olur. Struktur, elementlər arasındakı münasibətləri tənzimləyən qaydalarla müəyyən olunur. Ümumi halda hər bir element özü də öz növbəsində struktur ola bilər. Mürəkkəb iyerarxik strukturlar belə yaradılırlar. Müəyyən məna kəsb edən ən kiçik yazı elementləri **sahələr** adlanırlar. Məsələn, maşın əmrlərində, əməliyyatın koduna, unvanlara, əlamətlərə uyğun hissələr sahəyə misal ola bilər.

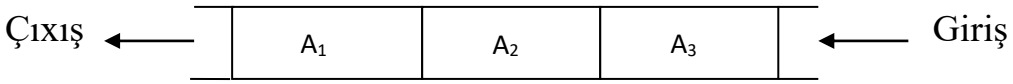
Çox geniş yayılmış verilənlərin strukturlarını nəzərdən keçirək.

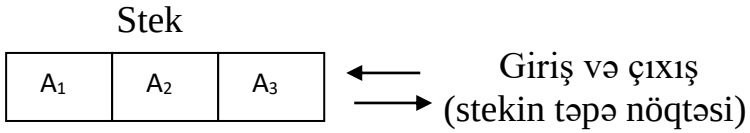
**Sətir** dedikdə hər bir elementi, birinci elementdən başqa, özündən əvvəlki və özündən sonrakı elementi olan elementlər toplusu başa düşülür (sonuncu elementin özündən sonrakı elementi yoxdur). Sətrin hər hansı bir elementinə müraciət etmək üçün sətrin sonlarından birindən elementlərə ardıcıl baxmaq lazımdır. Xüsusi halda giriş proqramı translyator üçün sətir hesab oluna bilər.

**Massiv** dedikdə, hər bir elementi ilə indeks adlanan nizamlanmış tam ədədlər toplusu əlaqədə olan elementlər toplusu başa düşülür. İndekslər massivin hər bir elementinin mövqeyini birqiymətli müəyyən edir. Düzbucaqlı massivlər massivin xüsusi halı hesab oluna bilər, belə ki, burada hər bir indeks eyni bir

addımla ən aşağı qiymətdən ən son qiymətə qədər dəyişir. Alqol-60 dilindəki massivlər düzbucaqlı massivlərə ən yaxşı misaldır. N indeksi olan massiv n-ölçülü massiv adlanır.

**Növbələr** və **steklər** – birölçülü dinamik dəyişən nizamlanmış elementlər toplusudur. Yeni elementlər bu topluya onun bir sonluğundan daxil edilir. Növbələrdən element həmişə onun digər sonluğundan "birinci gəldi, birinci getdi" prinsipi ilə silinir (çıxarılır). Stekdən isə element elə onun əlavə olunduğu sonluqdan "sonuncu gəldi, birinci getdi" prinsipi ilə silinir. Steki həm də maqazin adlandırırlar, belə ki, tapançanın maqazininin analoqu olaraq fəaliyyət göstərir. Elementlərin əlavə olunduğu və silindiyi sonluğu onun təpəsi adlanır.




**Cədvəl** dedikdə, hər bir elementi açar adalanan xüsusi fərqləndirici əlaməti olan elementlər toplusu başa düşülür. Elementlər cədvələ açarına görə əlavə olunur və cədvəldən açarına görə seçilir (çixarılır). Adətən açardan əlavə cədvəlin elementi (cədvəl yazısı) həm də müəyyən informasiya daşıyan verilənləri saxlayır. Cədvələ misal olaraq ixtiyari bir funksiyanın qiymətlər cədvəlini göstərmək olar. Belə cədvəldə açar rolunu arqumentin qiyməti oynayır, funksiyanın qiyməti isə informasiya hesab olunur. Arqumentin və funksiyanın qiymətləri birlikdə **cədvəl yazısı** əmələ gətirir. Translyatorlarda demək olar ki, həmişə adlar cədvəlindən istifadə olunur. Belə cədvəldə açar rolunu adətən identifikator oynayır, verilənlər

isə kəmiyyətin tipi haqqında göstərişi və onun qiymətinin saxlandığı ünvanı göstərir.

Sin (x) Funksiyasının cədvəli

| Açar (arqumentin qiyməti) | Verilənlər (funksiyanın qiyməti) |
|---|---|
| 0,00 | 0,0000 |
| 0,01 | 0,0100 |
| 0,02 | 0,0200 |
| ... | |
| | |

Adlar cədvəli

| Açar (identifikator) | Verilənlər (tip və ünvan) | |
|---|---|---|
| X1 | real | 0301 |
| X2 | real | 0302 |
| Y | real | 0303 |
| ... | ... | |
| | | |

**Ağac** düyünlər toplusundan (yaxud təpə nöqtələrindən) ibarətdir, belə ki, bu düyünlərin hər biri verilənlərdən başqa aşağı səviyyə təpə nöqtələrinə göstəricini də saxlayır. Ən yuxarı səviyyə yeganə təpə nöqtəsi **ağacın kökü**, ən aşağı səviyyə təpə nöqtələri **ağacın yarpaqları** adlanırlar. Kökdən başqa hər bir təpə nöqtəsi özündən yuxarı yeganə təpə nöqtəsini müəyyən edir, odur ki, verilmiş təpə nöqtəsindən kök təpə nöqtəsinə yeganə yol vardır. Ağacların köməyi ilə verilənlər arasındakı iyerarxik münasibətləri təqdim etmək çox əlverişlidir.

İstiqamətlənmiş qraf ağacdan onunla fərqlənir ki, burada hər bir təpə nöqtəsi bir deyil, bir neçə təpə nöqtəsi ilə əlaqədar ola bilir. Göstərən təpə nöqtəsini göstərilən təpə nöqtəsindən fərqləndirmək üçün onları birləşdirən xəttə ox işarəsi qoyulur. Ağacda ox işarələri lazım gəlmir, çünki həmişə yuxarı səviyyə təpə nöqtələri aşağıdakı təpə nöqtələrini göstərir. İstiqamətlənmiş qraflarda təpə nöqtəsindən çıxan til lə həmin təpə nöqtəsinə daxil ola bilər. Əgər belə yol başqa təpə nöqtələri saxlamırsa onda belə yol ilgək alanır, əks halda belə yol dövr

adlanır. Dövr saxlamayan qraf dövrü olmayan qraf adlanır. Proqramların blok-sxemləri adətən istiqamətlənmiş qraflarla təsvir olunurlar.

Müxtəlif verilənlərin strukturları kompüterin yaddaşında müxtəlif saxlanma strukturlarında saxlanılırlar. Saxlanma srtukturu topluları məhduddur, çünki bir çox maşınların yaddaşı nizamlanmış bilavasitə ünvanlanan (ünvanlana bilən) maşın sözləri (oyuqlar) yaxud hecalar (bayt) toplusundan ibarətdir. Saxlanma srtukturları saxlanma elementlərindən (yazılardan) ibarətdir. Əgər verilənlərin elementi bir maşın sözünə yerləşmirsə, yaxud əksinə, bir maşın sözündə ola bilsin ki, bir neçə verilənlərin elementi qablaşdırılıbsa, onda yazı bir neçə maşın sözü (bayt) yarada bilər. Minimal saxlanma elementi bir bit yəni bir ikilik mərtəbə hesab olunur.

## 1.2 Proqramların yaradılması texnologiyası

Proqramların yaradılması yaradıcılıq prosesi olsa da, bu prosesin sadələşdirilməsi üçün tətbiq oluna biləcək çoxsaylı alqoritmlər mövcuddur.

Məsələlərin həlli üçün əsas alqoritm, qoyulmuş məsələnin müstəqil alt məsələlərə bölünməsi alqoritmidir. **A** məsələsini həll etmək üçün onu əvvəlcə müstəqil **B, C, D** və sair alt məsələlərə bölmək zəruridir.

```
A: procedure;
        Məsələ B;
        Məsələ C;
        Məsələ D;
end A;
```

Alt məsələlər ixtiyari alqoritmlə həll oluna bilər. Bu metod proqramlaşdırmada bir çox məsələlərin həlli zamanı istifadə olunur.

Proqramlaşdırmada ən çox istifadə olunan metodlardan biri də məsələnin eyni çətinlikli hissələrə bölünməsi metodudur. Belə yanaşma həlli tələb olunan məsələnin təxminən eyni çətinlikli alt məsələlərə bölünməsi anlamına gəlir. Bu isə imkan verir ki, qoyulmuş məsələyə nisbətən daha sadə məsələlərin həllinə keçilsin.

Cədvəldə lazım olan elementin axtarılması belə alqoritmə xarakterik misal ola bilər:

- Birinci elementi yoxlamalı (başqa sözlə, birinci element axtarılandırmı);
- İF (əgər) axtarılan element tapılmayıbsa, THEN (onda) axtarışı qalan elementlərin arasında davam etdirməli;

Beləliklə, qoyulmuş məsələ iki hissəyə bölünür: birinci elementin yoxlanılması və qalan elementlər arasında axtarış. Aydındır ki, belə alqoritmlər çətinliklərinə görə eyni qiymətə malik deyillər. Modifikasiya olunmuş alqoritm – ikilik axtarış – aşağıdakı şəkildədir:

- Birinci elementi yoxlamalı (başqa sözlə, birinci element axtarılandırmı);
- İF (əgər) axtarılan element tapılmayıbsa, THEN (onda) axtarışı qalan elementlər siyahısının ortasının ya yuxarı yaxud da aşağı hissəsində davam etdirməli.

Bu halda məsələ təxminən eyni çətinlikli iki hissəyə bölünür. Hissələrdən hər biri rekursiv həllə malik ola bilər.

Bəzən məsələnin həllini elə həmin məsələnin daha sadə, məlum dəyişdirilmiş variantı kimi ifadə etmək mümkündür. Əgər bu proses lazım olan qiymət axtarılana qədər davam etdirilərsə, onda bu cür həll rekursiv həll adlanır.

Rekursiv həllə misal olaraq N faktorialın, fact(N) = 1×2× ... ×N hesablanmasını göstərmək olar. Tutaq ki, fact(460) qiyməti məlumdur, əgər fact(459) qiymətini 460 ədədinə vursaq, onda məsələ fact(459) qiymətinin hesablanmasına gətirilmiş olur. Əgər bu dəyişdirilmələri (çevirmələri) 458 dəfə təkrar etmiş olsaq, fact(1) = 1 olduğundan, faktorialın qiymətini hesablamaq olar. Beləliklə, əgər:

1) Hər hansı bir məlum G funksiyası üçün F(N) = G(N, F(N-1)) və
2) F(1) hər hansı bir məlum qiymətə malikdirsə,

onda F funksiyası rekursivdir. F funksiyası bir neçə parametrin funksiyası ola bilər.

N qiyməti aşkar şəkildə verilməlidir. Həm də N kəmiyyəti hər hansı bir çoxluğun elementi, intervalın uzunluğu yaxud başqa bir parametr ola bilər.

Rekursiv həll həmişə uyğun yekursiv olmayan variantdan sadə şəkildə yazılır. Həll sürətinə nisbətən birqiymətli nəticəyə gəlmək olmaz.Bu G funksiyasının nə qədər mürəkkəb olduğundan asılıdır.

Dinamik proqramlaşdırma – həlli tələb olunan məsələnin bütün alt məsələlərinin həlli üçün eyni bir hesablama sxeminin istifadə olunduğu cədvəl metodudur. Bu metod istifadə olunan zaman bir dəfə tapılan nəticə cədvələ yazılır və sonradan təkrar hesablanmır.

Çox vaxt qoyulmuş məsələnin dəqiq həllini onun qiyməti, mürəkkəbliyi yaxud da ölçüsü nöqteyi nəzərincə almaq mümkün olmur. Belə hallar, hərəkətin idarə olunması, iqtisadi proqnozlaşdırma və sair məsələlərin həlli zamanı baş verir. Bu vəziyyətlərdə məsələnin müəyyən xassələrinin "model" adlanan təsəvvürü yaradılır, yerdə qalan digər xassələr nəzərə alınmır. Bu zaman modelin lazım olan xassələrinin davranışının, yəni necə dəyişdiklərinin analizi aparılır. Belə yanaşma modelləşdirmə adlanır. Modelləşdirmə bir çox sahələrdə tətbiq olunur, lakin proqram təminatının yaradılması zamanı kifayət qədər az istifadə olunur. Modelləşdirmə əsasən kompüterlərdə müxtəlif fiziki proseslərin tədqiqi zamanı, xüsusilə də bu proseslərin riyazi şərhi məlum olduqda, istifadə olunur. Buna baxmayaraq, belə yanaşma böyük proqram kompleksləri yaradılan zaman proqramçıların diqqət mərkəzində olmalıdır.

Axtarış dedikdə lazım olan qiymətlərin hər hansı bir çoxluqda axtarılması nəzərdə tutulur. Elementlər çoxluğu strukturlaşdırılmış ola yaxud olmaya bilər. Əgər elementlər çoxluğu strukturlaşdırılmamışdırsa, onda axtarış adətən lazım olan elementin siyahıda axtarılması anlamına gəlir. Elementlər çoxluğu strukturlaşdırılmışdırsa (ağac, qraf və sairə), onda siyahı dedikdə strukturda düzgün yolun tapılması başa düşülür.

Adətən hər bir element üç komponentdən ibarət olur:

- açar – verilənə müraciət üçün istifadə olunan element (bu, sorğu kitabında hər hansı bir ad, indeksdə söz yaxud cədvəldə bölmə adı ola bilər);
- arqument - elementlə bağlı olan ədəd (ünvan, telefon nömrəsi və sair);

- struktur - bu, verilənlərin şərhi üçün tətbiq olunan əlavə informasiyadır., məsələn, çoxluqda növbəti elementin ünvanı və s.

**Siyahılarda axtatış.** Belə axtarış dörd müxtəlif üsullarla aparılır: düz (birbaşa), xətti, ikilik, ***xeş-axtarış***.

**Birbaşa axtarış.** Birbaşa axtarış zamanı elementin olduğu yeri bilavasitə açarın köməyi ilə müəyyən edirlər (məsələn, binada otaq). Birbaşa axtarış o zaman tətbiq olunur ki, elementlərin sayı qeyd olunmuşdur və onların sayı o qədər də çox deyildir.

**Xətti axtarış.** Axtarışın bu növü, onun yerinə yetməsi çox vaxt aparsa da, proqramlaşdırılmaq üçün kifayət qədər sadədir. Elementlər bir bir lazım olan, yəni axtarılan element tapılana qədər ardıcıl yoxlanılır. Birbaşa axtarışdan fərqli olaraq burada elementlərin yerləşməsində (düzülüşündə) sistem yoxdur. Əgər siyahıda N sayda element vardırsa, onda hər dəfə orta hesabla N/2 sayda element yoxlanılacaqdır. Xətti axtarış çox ləng gedir, yəni axtarışın sürəti azdır, lakin imkan verir ki, asanlıqla yeni elementlər siyahının axırına yerləşdirilməklə əlavə olunsunlar. Yaxşı tərtib olunmuş proqram adətən sadə (xətti) struktura malik olur, və, uyğun olaraq, verilənlər də belə sadə struktura malik olmalıdırlar. Ona görədir ki, belə strukturlarda xətti axtarış effektivdir.

**İkilik axtarış**. İkilik axtatışı aparmaq üçün açarlar kəmiyyətlərinə yaxud əlifba sırasına görə düzülməlidirlər. Belə olan halda axtarış sütətini orta hesabla $\mathbf{log_2\ N}$ kimi əldə etmək olar. İkilik axtarışda açar siyahının orta elementinin açarı ilə müqayisə olunur. Əgər açarın qiyməti böyükdürsə, onda həmin prosedur siyahının ikinci hissəsi üçün təkrarlanır, yox əgər açarın qiyməti kiçikdirsə, onda onda həmin prosedur siyahının birinci hissəsi üçün təkrarlanır. Beləliklə, bu prosesin bir addımında siyahı yarıya qədər kiçilir.

İkilik axtarış zamanı ən əsas çətinkil yeni elementlərin necə daxil ediləcəyidir. Siyahıda elementlər müəyyən sıra ilə düzüldüklərindən, yeni element daxil edilərkən ola bilsin ki, yeni elementin lazım olan yerdə yerləşdirilməsi üçün siyahı yenidən köçürülsün (yazılsın).

**Xeş-axtarış.** Axtarışın bu növü böyük verilənlər toplusunda axtarış üçün birbaşa axtarışın aparılması cəhdidir. Açarın birinci qiymətindən psevdoaçarın qiyməti hesablanır, bu xeş-kod adlanır və bu kod elementlərin ünvanları cədvəlində indeks kimi istifadə olunur. Əgər bütün açarlar müxtəlif xeş-kodlara uyğundurlarsa, onda ixtiyari element yeganə birbaşa axtarışla tapıla bilər.

*Misal.* Abstrakt kompüterdə dəyişənlərin adları üçün 1000 yer ayrıla bilər. Dildə 6 simvoldan ibarət sözlərə icazə verilir. Bu $26^6$ sayda ad yaratmağa imkan verir, bu isə simvollar cədvəlinin ölçüsündən xeyli böyükdür. Lakin, əgər adı təsvir edən hər bir ədədin qiymətini (0 – 999) intervalında olan ədədə qədər kiçiltmək olarsa, onda adlar cədvələ yazıla bilər. Xeş-kodların hesablanması alqoritmləri müxtəlifdir, məsələn, ada (açara) ikilik ədəd kimi baxmaq olar. Bə ədədi cədvəlin ölçüsünə bölmək, qalığı isə kod kimi istifadə etmək olar. Digər alqoritmlər ədədin daxilində bitləri toplamağı, yaxud açarla bəzi digər funksiyaları yerinə yetirməyi təklif edirlər.

Xeş-axtarışın çatışmayan cəhəti ondan ibarətdir ki, bəzi identifikatorlar eyni xeş-koda malik ola bilərlər. Odur ki, elementlərin, daha doğrusu eyni xeş-koda malik elementlərin, kəsişməsinin işlənməsi problemi meydana çıxır. Xeş-kodun müəyyəm qiyməti birinci dəfə rast gəlindikdə, onu cədvəldə müəyyən yerdə yerləşdirirlər. Əgər digər bir elementlə kəsişmə baş verərsə, onu aşağıdakı alaqoritmə uyğun olaraq başqa bir yerə yerləşdirirlər.

```
İNSERT:  procedure  (açar, arqument);
    declare     1  TABLE (ölçü),
                      2    KEY,
                      2    ARQUMENT;
    declare    xeş-kod;
    Xeş-kodu hesablamalı;
    / * əgər yazı cədvəldə yerləşirsə */
    İF TABLE (xeş-kod) . KEY = açar   then return;
    / * üst üstə düşən keş-kodların işlənməsi */
```

**do while** (TABLE (xeş-kod) . KEY ≠ 0 &

TABLE (xeş-kod) . KEY ≠ açar);

xeş-kod = (xeş-kod, açar) – dan asılı yeni qiymət;

end;

TABLE (xeş-kod) . KEY = açar

TABLE (xeş-kod) . ARGUMENT = argument;

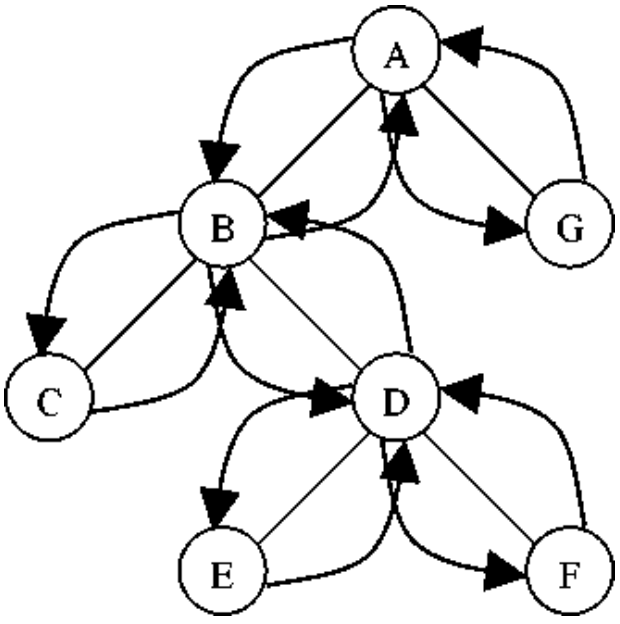
end İNSERT;

Beləliklə, elementlərin kəsişməsi baş verdiyi halda cədvəldə növbəti yerə keçid baş verir. Əgər lazım olan yer tutulubsa, onda tutulmuş yerdən hər hansı məsafədə olan yerə keçid baş verir, yaxud cari açar və koddan istifadə etməklə yeni xeş-kod hesablanır.

Xeş-kodun ən böyük çatışmayan cəhəti ondan ibarətdir ki, cədvəl sıx dolu olduğu halda elementlərin daxil edilməsi alqoritmi çox mürəkkəbləşir.

**Axtarış ağacı.** Verilənlər ağac strukturu şəklində təşkil olunduqda axtarış alqoritmi ağacın bütün düyünlərini nəzərdən keçirən alqoritmə gətirilir. Axtarışın aparılması üçün iki axtarış alqoritmi mövcuddur: dərininə axtarış və eninə axtarış.

Dərininə axtarış zamanı ağacən hər bir budağı soldan sağa nəzərdən keçirilir. İkilik ağaclar üçün axtarış alqoritmi sadə rekursiv struktura malikdir (sək. 1.5):

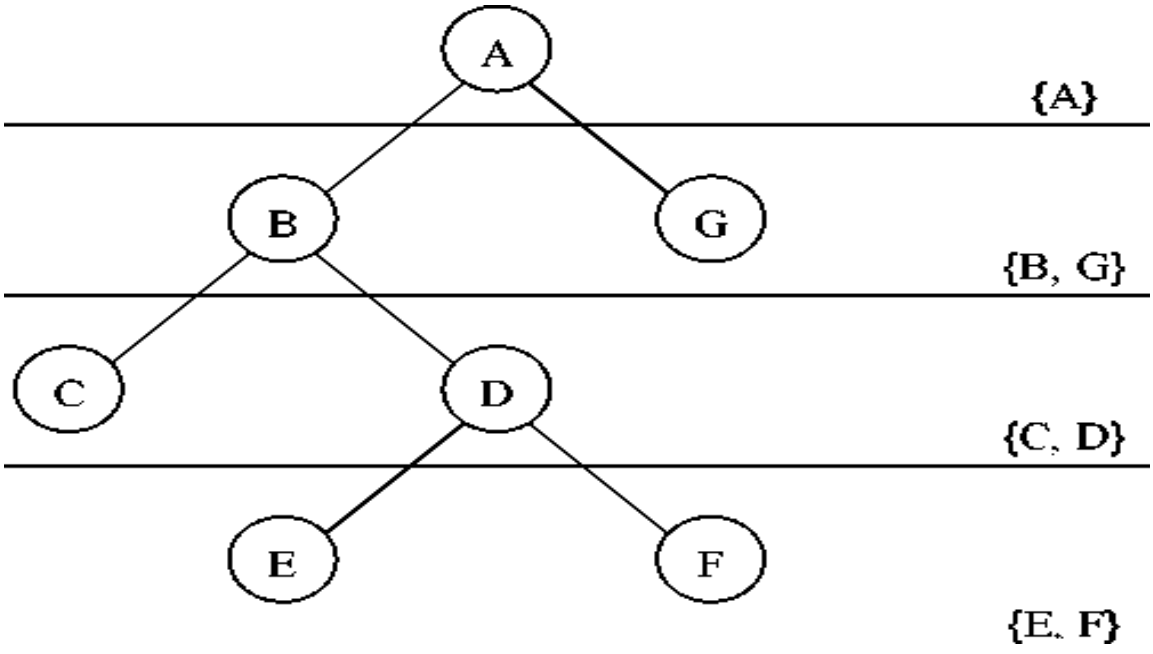



Şəkil 1.5 İkilik ağac üzrə axtarış sxemi

```
DEPTHFİRST: procedure (TREE);
    declare   1  TREE,
                 2  KEY,
                 2  ARGUMENT,   /* verilənlər */
                 2  RightSon,     /* sıfr, əgər oğul yoxdursa */
                 2  LeftSon;     /* sıfr, əgər oğul yoxdursa */
        if  TREE = null  then return;
        if   cari düyün KEY deyil  then do;
             call  DEPTHFİRST (LeftSon);
             call  DEPTHFİRST (RightSon);
        end;
    end  DEPTHFİRST;
```

Qeyd edək ki, ağac elə nizamlanmışdır ki, LeftSon parametrinin açarının qiyməti ağacın kök təpə nöqtəsinin qiymətindən kiçik, amma RightSon üçün açarın kəmiyyəti ağacın kök təpə nöqtəsinin qiymətindən böyük olarsa, onda belə alqoritm ikilik axtarış alqoritminin analoqudur.

Eninə axtarış zamanı axtarış alqoritmi ağacın hər bir səviyyəsinin yuxarıdan aşağıya nəzərdən keçirilməsini təmin edir (şək. 1.6).




Şəkil 1.6 Eninə axtarış

```
BREADTHFIRST: procedure(solution);
    declare 1 TREE,
              2  ARGUMENT, /* verilənlər */
              2  SONS(N);  /* ixtiyari ağaclar */
    declare (solution, NextStep) set of TREE;
    /* hər bir səviyyə üçün aşağıya keçid*/
    do while (solution ≠ 0);
     NextStep = {solution – da bütün I və TREE – lər üçün TREE.SONS(I) };
          call BREADTHFIRST(NextStep);
    end;
 end BREADTHFIRST;
```

Labirint haqqında məsələlərin həlli zamanı eninə axtarış çox populyardır.

Qayıtmaqla axtarış. Bir çox alqoritmlər üçün məsələnin mümkün həlli variantlarının seçilib ayrılması tələb olunur. Belə seçib ayırma üsullarından biri də qayıtmaqla axtarış adlanır.

**Alqoritm:**

```
Birinci hissəni çağırmalı;
do while  (qurtarana kimi);
    növbəti hissəni çağırmalı;
    do while  (davam etdirmək imkanı yoxdur);
        Əvvəlki səviyyəyə qayıtmalı;
        Bu səviyyə üçün alternative həllin mümkünlüyünü yoxlamalı;
    end;
end;
```

Qayıtmaqla axtarışa misal kimi dərininə axtarış alqoritmini göstərmək olar:

```
Ağacın kök təpə nöqtəsindən başlamalı;
do while  (nəzərdən keçirilməyən təpə nöqtələri vardır);
    Əgər mümkündürsə “sol oğul” təpə nöqtəsinə keçməli;
    Əks halda “sağ qardaş” təpə nöqtəsinə keçməli;
    do while (“sol oğul” və “sağ qardaş” təpə nöqtələri yoxdur);
```

“ata” təpə nöqtəsinə keçməli;

Əgər mümkündürsə “sağ qardaş” təpə nöqtəsinə keçməli;

**end;**

**end;**

## 1.3 Proqramların testləşdirilməsinin psixologiyası və iqtisadiyyatı

Proqram təminatının bütün komponentləri yaradılarkən onların testləşdirilməsi mühüm mərhələlərdən biri hesab olunur. Testləşdirməyə tədqiqat obyekti olaraq müxtəlif nöqteyi nəzərdən, həm də texniki nöqteyi nəzərdən baxmaq olar. Lakin testləşdirmə öyrənilərkən mühüm cəhətlətdən biri onun iqtisadiyyatı və proqramın yaradıcısının psixologiyası hesab olunur. Başqa sözlə desək, proqramın testləşdirilməsinin həqiqiliyi birinci növbədə proqramın kim tərəfindən testləşdirildiyi, onun düşüncə tərzi, sonra isə müəyyən texniki aspektlərlə müəyyən olunur. Odur ki, texniki problemlərdən əvvəl bu suallar nəzərdən keçirilməlidir.

“Testləşdirmə” anlayışının müəyyən olunması ilk baxışda çox sadə görünə bilər. Bu anlayışın müzakirə olunmasının zəruruliyi onunla əlaqədardır ki, bir çox mütəxəsislər bu termindən düzgün istifadə etmirlər. Bu isə öz növbəsində pis testləşdirmə ilə nəticələnir. Məsələn, aşağıdakı tərifləri nəzərdən keçirək: “Testləşdirmə proqramda səhvlərin olmadığını nümayiş etdirən prosesdir”, “testləşdirmədə məqsəd – proqramın nəzərdə tutulan funksiyaların korrekt yerinə yetirdiyini göstərməkdir”, “testləşdirmə - proqramın öz funksiyasını yerinə yetirdiyinə əmin olmağa imkan verən prosesdir”.

Bu təriflər, testləşdirmə anlayışının düzgün müəyyən olunmasına təmamilə əks olduğundan, heç biri doğru hesab oluna bilməz. Hələlik testləşdirməyə düzgün tərif verməyi bir tərəfə qoyaraq fərz edək ki, əgər biz proqramı testləşdiririksə, onda biz proqrama yeni qiymər də əlavə etməliyik ( yəni, testləşdirməyə pul sərf olunur və biz sərf olunan pulu geri qaytarmaq üçün proqramın qiymətini qaldırmalıyıq). Qiymətin artırılması proqramın kefiyyətinin artırılması yaxud

etibarlılığının yüksəldilməsi deməkdir. Bu isə proqramdakı səhvlərin aşkar olunması və aradan qaldırılması ilə əlaqədardır. Beləliklə, proqram ona görə testləşdirilmir ki, o normal işləyir, tam əksinə, fərz olunur ki, proqramda səhvlər vardır (bu fərziyyə praktiki olaraq bütün proqramlar üçün doğrudur), sonra isə artıq onların mümkün maksimal sayı aşkar olunur. Beləliklə, nisbətən qəbul oluna biləcək şəkildə testləşdirmənin tərifini formalaşdıra bilərik:

**Testləşdirmə** - səhvlərin aşkar olunması məqsədi ilə proqramın yerinə yetirilməsi prosesidir.

Testləşdirməyə verilən bu tərifdən bir neçə nəticə çıxarmaq olar. Məsələn, belə nəticəyə gəlmək olar ki, testləşdirmə - destruktiv prosesdir (yəni yaradıcıya əks olan struktur proses). Çoxlarının bu prosesi çətin bir proses adlandırması bilavasitə bununla izah olunur. İnsanların bir çoxu obyektlərin konstruktiv yaradılması prosesinin tərəfdarıdır, az bir hissəsi isə hissələrə bölünən destruktivprosesin tərəfdarıdır. Verilən tərifdən həm də o nəticəyə gəlmək olur ki, test verilənləri toplusu necə təşkil etmək olar və baxılan proqramı kim testləşdirməlidir (kim testləşdirməli deyildir).

Testləşdirməyə verilən tərifi daha da dəqiqləşdirmək üçün iki anlayışı, “uğurlu” və “uğursuz” anlayışlarını analiz edək. Onların layihənin rəhbəri tərəfindən testləşdirmənin nəticələrini qiymətləndirərkən istifadə olunmasına baxaq. Bir çox layihələrin rəhbərləri testləşdirmənin aparılması zamanı səhvlər aşkar olunarsa onu uğursuz, və əksinə, səhvlər aşkar olunmazsa uğurlu hesab edirlər. Hər şeydən əvvəl bu “testləşdirmə” anlayışının səhv başa düşülməsinin nəticəsidir, belə ki, əslində “uğurlu” anlayışı “nəticəsi olan”, “uğursuz” anlayışı isə “nəticəsiz”, “arzu olunmayan” mənalarını verir. Lakin əgər test səhvləri aşkar etmirsə, onun yerinə yetirilməsi vaxt və pul itkisi ilə əlaqədardır, və “uğurlu” anlayışı ona şamil edilə bilməz.

Səhvlərin aşkar olunması ilə nəticələnən testin yerinə yetirilməsi, yuxarıda göstərildiyi kimi, məqsədəuyğun kapital qoyuluşu olduğundan, uğursuz hesab oluna bilməz. Buradan belə nəticəyə gəlmək olur ki, “uğurlu” və “uğursuz” sözlərinə ümumi qəbul olunanın əksi olan mənaları vermək zəruridir. Ona görə də

gələcəkdə testin yerinə yetirilməsi zamanı səhvlər aşkar olunarsa onu uğurlu, və əgər korrekt nəticə olarsa "uğursuz" adlandıracağıq.

Baxılan prosesi xəstənin həkimin qəbulunda olması ilə müqayisə edək. Əgər həkimin məsləhət gördüyü laboratoriya analizləri xəstəliyin səbəbini aşkar etməyibsə, onda biz belə müayinəni uğurlu hesab edə bilmərik – o uğursuzdur: xəstə axı pul ödəyib, lakin o əvvəlki kimi xəstədir. Əgər aparılan müayinə göstərsə ki, xəstənin mədəsində xora var, onda müayinə uğurlu hesab olunur və həkim müalicə kursu təyin edə bilir. Beləliklə, tibb işçiləri bu anlayışları bizə lazım olan mənada istifadə edirlər (burada oxşarlıq, əlbəttə ki, ondadır ki, test olunacaq proqram xəstə pasiyentə uyğundur).

Testləşdirmənin "Testləşdirmə səhvlərin olmadığını nümayiş etdirən prosesdir" tipli müəyyən olunması əlavə bir problem də doğurur: onlar qarşıya, heç bir, hətta çox sadə, proqram üçün əldə oluna bilməyən məqsəd qoyurlar. Psixoloji tədqiqatların nəticəsi göstərir ki, əgər insanın qarşısında həlli mümkün olmayan məsələ qoyulursa, o çox pis işləyir. Başqa sözlə desək , testləşdirmənin səhvlərin aşkar olunması prosesi kimi müəyyən olunması həlli olan məsələlər dərəcəsinə gətirir, və beləliklə psixoloji çətinliklər aradan qaldırılır.

Digər bir problem o vaxt meydana çıxır ki, testləşdirmə üçün aşağıdakı tərif istifadə olunsun:

"Testləşdirmə - proqramın öz təyinatını yerinə yetirdiyinə inanmağa imkan verən prosesdir", belə ki, bu tərifi ödəyən proqramda səhvlər ola bilər. Əgər proqram ondan tələb olunan işi yerinə yetirmirsə, aydındır ki, bu proqramda səhvlər vardır. Lakin proqramda səhvlər o halda da ola bilər ki, bu proqram ondan tələb olunmayan işləri də yerinə yetirir.

Yekun olaraq demək olar ki, testləşdirmə destruktiv proses olmaqla proqramda səhvlərin aşkar olunması cəhdidir (səhvlərin olduğu güman edilir). Səhvlərin tapılmasına imkan yaradan testlər toplusu uğurlu hesab olunur. Təbii ki, son nəticə olaraq hər kəs testin köməyi ilə əmin olmaq istəyir ki, onun proqramı öz təyinatına uyğundur və təyin olunmadığı funksiyanı yerinə yetirmir. Məqsədə çatmaq üçün ən yaxşı üsul səhvlərin bilavasitə axtarılmasıdır.

Testləşdirməyə verilən tərifləri nəzərə almaqla növbəti addımda proqramdakı bütün səhvləri aşkar edə bilən testin yaradılması imkanını nəzərdən keçirmək zərurəti ortaya çıxır. Asanlıqla göstərmək olar ki, hətta ən sadə proqram üçün bu mümkün deyildir. Bu isə öz növbəsində iqtisadi, həm də proqramlardakı səhvlərin aşkar edilməsi prosesində insanın funksiyaları, testlərin yaradılması üsulları problemlərini doğurur.

Proqramların ənənəvi testləşdirilməsi metodları iki qrupa bölünür: statik və dinamik.

Statik metodlar proqramların reallaşdırılması məsələsini nəzərə almadan onların analizinə əsaslanır. Qeyd edək ki, altı belə metod məlumdur.

Birinci metod verilənlər axınının analizi metodu adlanır. Bu metodda məqsəd proqramların informasiya yollarında verilənlərin uyuşan olmadığını aşkar etməkdir. Bu iş bütün dəyişənlərin şərhinin yoxlanılması və onların adlandırılmasının yaddaşın sinifləri ilə razılaşdırılması yolu ilə həyata keçirilir.

İkinci metod interfeysin analizi metodu adlanır. Bu metod parametrlərin müxtəlif proqram modulları arasında ötürülməsinə nəzarət məqsədini güdür. Eyni zamanda aşağıdakı yoxlamalar həyata keçirilir:

1) Parametrlərin və arqumentlərin ölçü vahidlərinin uyğunluğu;
2) Modullara ötürülən (verilən) arqumentlərin sayının onların parametrlərinin sayına uyğunluğu;
3) Parametrlərin və arqumentlərin atributlarının və onların gəlmə ardıcıllığının uyğunluğu;
4) Arqumentlərin, onların atributlarının sayının verilməsinin düzgünlüyü və standart funksiyaların çağırılması zamanı onların gəlmə ardıcıllığı;
5) Cari giriş nöqtəsi ilə əlaqədar olmayan parametrlərə müraciətin olması;
6) Yalnız giriş verilənləri rolunu oynayan arqumentlərin alt proqramlarda dəyişməz qaldığı;
7) Bütün modullarda qlobal dəyişənlərin təyin olunmasının razılaşdırıldığı;
8) Arqumentlərin sabit şəkildə verilməsi.

Üçüncü metod mümkün ola biləcək səhvlərin axtarılması metodudur. Belə səhvlər verilənlərə müraciət zamanı, hesabi yaxud məntiqi ifadələrdə, yaxud da idarənin ötürülməsi zamanə rast gəlinə bilər. Verilənlərə müraciət zamanə rast gəlinən səhvlər aşağıdakılardır:

1) Adlandırılmayan yaxud şərh olunmayan dəyişənlərə müraciət;
2) Massivin indeksinin icazə verilməyən qiymətə malik olması;
3) Göstərici yaxud dəyişən-istinadın köməyi ilə bölüşdürülməmiş (paylanmamış) yaddaşa müraciət;
4) Yaddaş oblastında müxtəlif atributlara malik bir neçə adı (psevdonimi) olan verilənlərin qiymətlərinin korrekt olmayan atributları;
5) Yaddaşın ünvanlaşdırıldığı ölçü vahidindən kiçik yaddaş vahidindən istifadə;
6) Bir neçə prosedurun yaxud alt proqramın müraciət etdiyi verilənlərin strukturu. Bu struktur müxtəlif prosedurlarda müxtəlif şəkillərdə müəyyən olunur.

Dəyişən-istinadların olması "asılı qalan müraciətlər" tipli səhvlərin yaranmasına səbəb ola bilər, belə ki, müraciət olunan yaddaşın müəyyən edildiyi vaxt keçdikdə bu yaddaşa müraciət səhvə gətirib çıxarır. Məsələn, prosedurun mətnində göstərici lokan dəyişəni müəyyən edir, göstəricinin qiyməti çıxış parametrinə yaxud qlobal dəyişənə mənimsədilir, prosedur sona çatır (bu ünvanlaşdırılan yaddaşın azad edilməsi ilə müşayət olunur) və sonra isə göstəricinin qiymətindən istifadə etməyə cəhd edilir. Bu isə yuxarıda göstərilən səhvə gətirib çıxarır.

Hesablama zamanı aşağıdakı tipik səhvlər baş verə bilir:

1) Dəyişənlərin icazə verilməyən (məsələn, hesabi olmayan) tipləri üzərində hesablama;
2) Müxtəlif tipli verilənlər üzərində hesablama;
3) Müxtəlif ölçülü (uzunluqlu), lakin eyni tipli verilənlər üzərində hesablama;
4) Sıfra bərabər bölənin olduğu zaman;
5) Tam ədədlər üzərində düzgün olmayan hesablamalar aparılan zaman.

Münasibət əməlləri yerinə yetirilən zaman aşağıdakı hallarda səhvlər baş verə bilir:

1) Uyuşan olmayan operandların olduğu zaman;
2) Müxtəlif tipli yaxud müxtəlif uzunluqlu operandların olduğu halda;
3) Bul ifadələrdə qeyri bul operandların olduğu halda və bul ifadələrin və münasibətlərin birləşdirilməsində səhvlər olduğu halda;
4) Bul ifadələrin konkret kompilyator tərəfindən yerinə yetirilməsi üsulunun olmadığı halda.

Dördüncü metod çarpaz istinadların analizi metodu adlanır. Bu analiz ondan ibarətdir ki, proqramların mətnində istinad edilməyən dəyişənlər çarpaz istinadlar cədvəlində aşkar olunurlar.

Beşinci metod bir birinə zidd olmamanın yoxlanılması metodudur. Belə yoxlama "ölü kodların" (yəni heç zaman yerinə yetməyən əmrlərin) aşkar edilməsi üçün proqramın məntiqinin yoxlanılması və proqramın məntiqində ziddiyyətlərin aşkar olunması zamanı aparılır. Əksər hallarda bu metod, aralarında simvol şəklində verilmiş verilənlərin olduğu giriş verilənləri üzərində proqramın simvolik yerinə yetməsinin idarəsi axınının analizi adlanır. Proqramın simvolik yerinə yetməsinin nəticəsini ilkin verilənlər üzərində düsturlarla ifadə olunmuş dəyişənlərin simvil qiymətləri təşkil edir.

İki növ simvolik yerinə yetmə məqsədini bir birindən fərqləndirirlər:

1) Proqramın verilmiş yolu üçün simvolik yerinə yetmə zamanı bu yolun spesifikasiyasını müəyyənləşdirən testləşdirici verilənlər toplusunun yaradılması; belə verilənlər toplusu olmadıqda verilmiş yolun reallaşdırıla bilən olmadığı (yəni ziddiyyət olduğu) qənaətinə gəlinir;
2) Spesifikasiyalarla ifadə olunmuş və ilkin obyektlərin uyğun sərhəd qiymətlərini müəyyənləşdirən ilkin verilənlər üçün verilmiş məhdudiyyətlər daxilində proqramda keçiləcək yolu müəyyən etməli və dəyişənlərin simvolik qiymətləri yolun sonunda verilir, başqa sözlə desək, proqramla reallaşdırılan müəyyən funksiya qurmaq tələb olunur (tələb

olunmur ki, bu funksiya mütləq ilkin verilənləri arzu olunan çıxış verilənlərinə inikas edəcəkdir).

Tutaq ki, P(x,y) - $S = (S_1, S_2, \ldots, S_n)$ ; $S \in X$, müəyyən ilkin verilənləri olmaqla simvolik yerinə yetən proqramdır. Burada X çoxluğunun elementləri P(x,y) proqramının ilkin verilənləri çoxluğudur. Proqramın operatorlarını nömrələyək. Proqramın yerinə yetmə vəziyyəti $< N, pc, Z >$ üçlüyü ilə müəyyən olunur; burada N – cari operatorun nömrəsi, pc – proqramda yolun seçilməsi şərtidir (ilk əvvəl bu şərt doğruya bərabər götürülür) və S çoluğunun elementlərindən ibarət məntiqi ifadədir; Z isə aşağıdakı cütlüklə müəyyən olunur:

$$\{ < z_i , e_i > | \ z_i \in X \cup Y\},$$

burada $z_i$ – proqramın dəyişənləri (obyektləri), $e_i$ bu dəyişənlərin qiymətləridir. Y isə aralıq yaxud nəticə verilənləridir.

Proqramın simvolik yerinə yetməsi semantikası simvol qiymətlərlə əməliyyat qaydaları ilə verilir, bu qaydalara əsasən hesabi hesablamalar cəbri hesablamalarla əvəz olunur. Simvolik yerinə yetmənin semantikasını proqramlaşdərma dillərinin baza konstruksiyaları vasitəsi ilə verək. Paralelizm vasitələrinin olmadığı əmr formalı proqramlaşdırma dilləri üçün baza konstruksiyaları mənimsətmə, keçid və şərti operatorlar hesab olunur.

Z = e(x,y) mənimsətmə operatorunda x və y dəyişənlərinin simvollu qiymətləri e(x,y) ifadəsində yerinə qoyulur və nəticədə Z ( $z \in X \cup Y$) dəyişəninin qiyməti olacaq e(S) ifadəsi alınır. e(S) ifadəsinə Y çoxluğundan müəyyən qiymətlərin daxil olması onu göstərir ki, onların qiymətləri müəyyən olunmamışlar və Z operatorunun qiyməti hesablanmamışdır.

Testləşdirici verilənlər toplusunun qurulmasına yönəldilən (birinci məqsəd) simvolik yerinə yetmə aşağıdakı alqoritmlə reallaşdırılır:

1) Qəbul edirik ki, pc doğrudur;
2) Operatorların semantikasına uyğun olaraq verilmiş yol üçün ***pc*** yolunun seçilmə şərtini yaradırıq; bu zaman şərti operatorun semantikası pc – nin sadələşdirilmiş forması kimi başa düşülür.

İlkin verilənlər üzərində qurulmuş şərtlər daxilində keçiləcək yolların müəyyən olunmasına yönəldilən (ikinci məqsəd) simvolik yerinə yetmə aşağıdakı kimi reallaşdırılır:

1) pc = β(s), burada β(s) giriş spesifikasiyalarıdır, yaxud onun olmamasıdır; pc = doğru;

2) proqramın simvollu yerinə yetməsini operatorların semantikasına uyğun olaraq yerinə yetiririk. Əgər budaqlanmaya rast gəlinirsə, onda həmin nöqtədə vəziyyəti dəyişirik və budaqlardan birini seçirik, yaxud yeni fərziyyə irəli sürməklə verilmiş şərti operatoru yenidən yerinə yetiririk. Keçmə şərti proqramın vəziyyətinin (pc) ikinci komponenti tərəfindən qeydə alınır.

3) yolun sonunda çıxış dəyişənlərinin vəziyyətləri toplusu proqramla realaşdırılan funksiyanı şərh edir, pc şərtini ödəyən ixtiyari verilənlər toplusu, verilmiş yolu örtən, yəni əhatə edən test olur.

4) Bütün aralıq $\delta_i(x,y)$ və çıxış spesifikasiyaları $\gamma(x,y)$ üçün verilmiş yolda aşağıdakı məntiqi düsturların yerinə yetməsini isbat etmək olar:

$$pc \longrightarrow \delta_i(x,y),$$

$$pc \longrightarrow \gamma(x,y),$$

burada pc – proqramın baxılan nöqtəsində yolun şərtinin cari qiymətidir. Bu düsturların isbat edilməsi verilmiş yolda proqramın verifikasiyası hesab oluna bilər.

Misal. Tutaq ki, aşağıdakı proqram verilmişdir:

```
PO:  PROC(X,Y);                (1)
     Z = 1;                    (2)
     J = 1;                    (3)
AB:  IF  Y >= J  THEN          (4)
     DO;  Z = Z/X;             (5)
     J = J + 1;                (6)
     GO TO  AB;  END;          (7)
     RETURN (X);               (8)
     END  PO;                  (9)
```

(1) - (8) yolunu nəzərdən keçirək. Əgər X və Y parametrlərinin qiymətlərini $S_1$ və $S_2$ ilə işarə etsək, onda simvolun yerinə yetməsi nəticəsində yolun sonunda vəziyyət aşağıdakı şəkildə olacaqdır:

$$N = 8;\ S_2 \geq 1;\ S_2 < i + 1;\ X = S_1;\ Y = S_2;\ Z = İ * S_1;\ J = J + 1;$$

Müəyyən olunmuşdur ki, yol $1 \leq Y < 2$ şərti daxilində reallaşır, bununla əlaqədar olaraq proqramın simvollu yerinə yetməsi sona çatır və izlənilən birinci məqsədə çatırıq.

İkinci məqsədin izlənməsini həyata keçirən simvollu yerinə yetməni elə həmin PO proseduru misalında nümayiş etdirək. Tutaq ki, ilkin verilən olan $S_2$ üçün $S_2 < 1$ məhdudiyyət şərti verilmişdir. Asanlıqla göstərmək mümkündür ki, (1) – (3) operatorlarından sonra proqramın yerinə yetmə vəziyyəti aşağıdakı şəkildə olacaqdır:

$$N = 3;\ S_2 < 1;\ X = S_1;\ Y = S_2;\ J = 1;\ Z = 1;$$

Bu vəziyyət $Y \geq J$ şərtinin yalan olduğunu müəyyən etməyə imkan verir (belə ki, $S_2 \geq 1$ ola bilməz), və uyğun olaraq, növbəti yerinə yetən operator (8) olacaqdır. Prosedurun işinin nəticəsi Z = -1 qiymətidir.

Beləliklə, belə nəticəyə gəlmək olur ki, əgər X parametrinin dərəcə ölçüsü vahiddən kiçikdirsə (PO-nun parametrinin ikinci qiyməti), onda $Z = 1/X^Y$ qiymətinin hesablanması səhv proqramlaşdırılmışdır. Həm də Y = 0 başlanğıc qiyməti ilə təkcə mətn özü , hətta bu qiymət $S_2 < 1$ oblastından olsa belə, səhvləri tapa bilməzdi.

Görmək çətin deyildir ki, proqramların testləşdirilməsinin statik metodları, simvollu yerinə yetmədən başqa, yüksək səviyyəli prosedur proqramlaşdırma dillərinin translyatorlarının vasitəsi ilə tətbiq olunurlar. Qeyd edək ki, bu metodlar hələ keçən əsrin 60-cı illərində təklif olunmuş və bir çox kompilyatorların reallaşdırılması zamanı sınaqdan keçirilmişdir. Bu metodlar hal-hazırda da geniş istifadə olunurlar.

Lakin qeyd etmək lazımdır ki, translyatorlar vasitəsi ilə proqramlar diaqnozlaşdırılarkən çoxlu sayda qüsurların olduğu aşkar olunmuşdur. Odur ki, son illərdə translyatorların attestasiyası üçün proqramların diaqnostlaşdırılması

üsullarının sistemləşdirilməsi sahəsində tədqiqatlar aparılır və bir çox proqramlaşdırma dillərinin standartları qəbul olunur.

Proqramların testləşdirilməsi problemlərinin həlli dinamik testləşdirmə metodlarından da istifadə olunur. Bu metodlardan biri də qara qutu strategiyası, verilənlərə görə idarəetmə ilə testləşdirmə yaxud giriş-çıxışa görə testləşdirmə adlanan testləşdirmə strategiyasıdır. Bu strategiyadan istifadə zamanı proqrama qara qutu kimi baxılır. Başqa sözlə desək, belə testləşdirmənin məqsədi proqramın "özünü aparması" ilə onun spesifikasiyalarının uyğunsuzluğunun aşkar olunmasıdır.

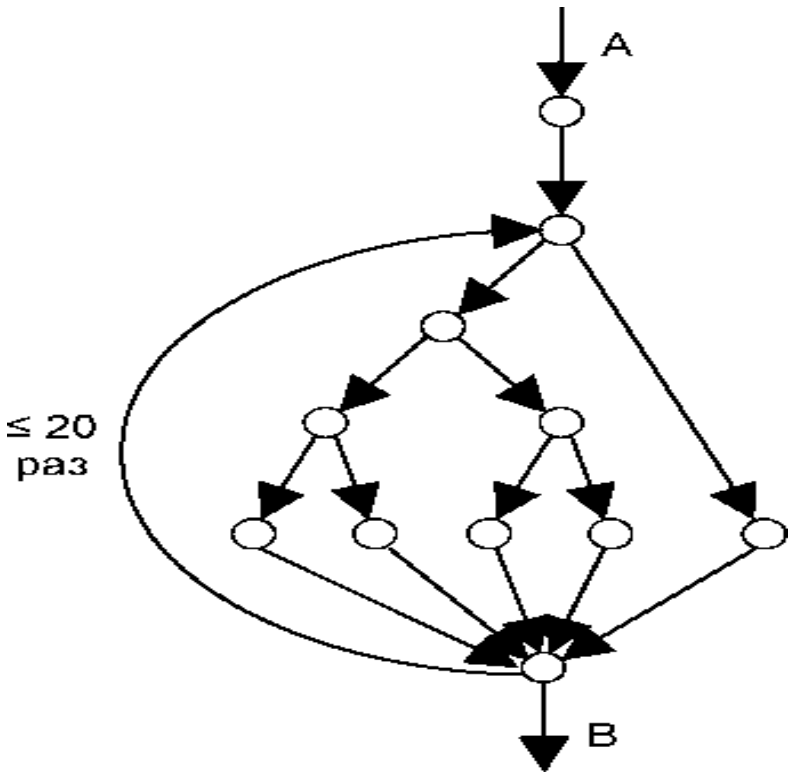
Belə yanaşma zamanı proqramdakı bütün səhvlərin aşkar olunması *mükəmməl giriş testləşdirməsi* kriteriyasıdır. Əgər test topluları kimi bütün mümkün giriç verilənləri topluları istifadə olunarsa, onda mükəmməl giriş testləşdirməsi əldə oluna bilər. Ona görə də hətta çox da böyük olmayan proqramın testləşdirilməsi üçün sonsuz sayda testlər tələb olunur. Misal kimi üçbucaq haqqında məsələni göstərə bilərik. Üç - A, B və C ədədləri verilmişdir. Müəyyən etmək tələb olunur ki, verilmiş bu üç ədəd üçbucağın tərəflərinin uzunluğu ola bilərmi, əgər ola bilərsə, onda bu üçbucaq düzbucaqlı, bərabəryanlı yaxud bərabərtərəfli üçbucaqdır. Aydındır ki, əgər A, B və C ədədləri həqiqi ədədlər olarsa, onda bu ədədlərin qiymətlərinin sonsuz sayda kombinasiyası mümkündür.

Əgər belə sınaq çox çətindirsə, onda böyük proqram üçün ətraflı testin yaradılması daha çətindir.

Yuxarıdakı şərhlərdən belə nəticəyə gəlmək olur ki, ətraflə giriş testlərinin yaradılması mümkün deyildir. Bu iki arqumentin vasitəsi ilə təsdiq oluna bilər: birincisi, səhvlərin olmadığına zəmanət verən test yaratmaq mümükün deyildir; ikincisi isə, belə testlərin yaradılması iqtisadi tələblərə ziddir. Ətraflı testləşdirmə mümkün olmadığından, məqsədimiz testləşdirməyə kapital qoyuluşunun sımərəliliyini maksimallaşdırmaq olmalıdır (başqa sözlə desək, bir test vasitəsi ilə aşkar olunan səhvlərin sayının maksimallaşdırılmasıdır). Bunun üşün biz proqramın daxili strukturunu nəzərdən keçirməliyik.

Ağ qutu strategiyası, yaxud proqram məntiqi ilə idarə olunan testləşdirmə strategiyası, proqramın daxili strukturunu tədqiq etməyə imkan verir. Bu halda testi icra edən şəxs test verilənlərini proqramın məntiqini analiz etmək yolu ilə əldə edir (təəssüflər olsun ki, çox vaxt burada proqramın spesifikasiyalarından istifadə olunmur).

Baxılan strategiyada testlərin yaradılması üsulunu qara qutu mükəmməl giriş testləşdirməsi ilə müqayisə edək. Testləşdirmə haqqında məlumatı olmayana elə gələ bilər ki, hər bir operatoru heç olmasa bir dəfə yerinə yetirilə bilən testlər toplusu qurmaq kifayətdir; bunun doğru olmadığını göstərmək şətin deyildir. Xırdalıqlara varmadan göstərmək olar ki, mükəmməl giriş testləşdirilməsinə uyğun olaraq mükəmməl marşrutların testləşdirilməsini qarşı qoymaq olar. Nəzərdə tutulur ki, idarəetmə qrafı üzrə bütün mümkün marşrutlardan keçməklə və testlərin köməyi ilə bu nproqramın yerinə yetirilməsini həyata keçirmək mümkündür. Bu fikrin iki zəif nöqtəsi vardır. Bunlardan biri ondan ibarətdir ki, proqramda bir birlərini təkrarlamayan marşrutların sayı – astronomikdir. Buna əmin olmaq üçün sadə proqramda idarənin ötürülməsi qrafını nəzərdən keçirmək olar (şəkil 1.7).




Şəkil 1.7 Böyük olmayan proqramın idarənin ötürülməsi qrafı.

Qrafın hər bir təpə nöqtəsi xətti operarorlar ardıcıllığından ibarət budaqlanma operatoru ilə qurtara bilən proqram sahəsini göstərir. İstiqamətlənmiş tillər isə idarənin ötürülməsinə uyğundur. Şəkildən göründüyü kimi qraf, DO dövr operatoru da daxil olmaqla 10-20 operatordan ibarət və 20 dəfədən az olmayaraq yerinə yetirilən proqramı şərh edir. Dövrün daxilində bir neçə İF operatoru vardır. Proqram yerinə yetərkən təkrarlanmayan marşrutların sayını müəyyən etmək üçün A təpə nöqtəsindən B təpə nöqtəsinə gedən təkrarlanmayan marşrutların sayını hesablayaq. Bu ədəd $5^{20} + 5^{19} + \dots + 5^{1} = 100$ trillion cəmi kimi hesablanır. Burada 5 – dövrün daxilindəki yolların sayıdır. Belə bir misala baxaq: fərz etsək ki, hər bir testin tərtibi üçün biz 5 dəqiqə sərf edirik, , onda testlər toplusunun tərtibi üçün bizə bir milyard il lazım olacaq.

İkinci zəif nöqtə ondan ibarətdir ki, marşrutların mükəmməl testləşdirilməsi tam test olsa da, bundan başqa proqramın hər bir marşrutu yoxlanılmış olsa da, proqramın özqndə səhvlər ola bilər. Bu hal aşağıdakı kimi izah olunur. Əvvəla marşrutların mükəmməl testləşdirilməsi proqramın öz şərhinə uyğun olduğuna zəmanət verə bilməz. Məsələn, ola bilər ki, tələb olunan artmaya görə testləşdirmə proqramı əvəzinə təsadüfən azalmaya görə testləşdirmə proqramı yazılsın. Belə olan halda marştutların testləşdirilməsinin qiymətliliyi o qədər də böyük deyildir, çünki, testləşdirmə sona yetdikdən sonra proqramda bir səhv olacaqdır, başqa sözlə proqram doğru deyildir. İkincisi isə, proqram ona görə doğru olmaya bilər ki, bu zaman bəzi marşrutlar nəzərə alınmamışdır. Marşrutların mükəmməl testləşdirilməsi onların olmadığını aşkar edə bilmir. Üçüncüsü isə, marşrutların mükəmməl testləşdirilməsi, verilənlərin emalından asılı olaraq əmələ gələn səhvləri aşkar edə bilmir. Belə səhvlərə çoxsaylı misallar göstərmək olar. Onlardan birinə baxaq. Tutaq ki, proqramda iki ədədin yığılmaya görə müqayisəsini yerinə yetirmək lazımdır, başqa sözlə desək, müəyyən etmək lazımdır ki, iki ədəd arasındakı fərq əvvəlcədən verilmiş müəyyən ədəddən kiçikdir. Bunun üçün aşağıdakı ifadəni yaza bilərik.

İF ((A – B) < EPSILON) …..

Aşkardır ki, bu ifadədə səhv vardır, çünki, əslində mütləq qiymətlərin müayisəsi yerinə yetirilməlidir. Lakin bu səhvin tapılması A və B üçün istifadə olunan qiymətlərdən asılıdır və səhv mütləq deyil ki, sadəcə olaraq hər bir marşrutun yerinə yetirilməsi yolu ilə aşkar olunsun.

Belə qərara gəlmək olur ki, mükəmməl giriş testləşdirməsi marşrutların mükəmməl testləşdirilməsinə nisbətən daha üstünlük verilə biləndir. Lakin onların heç biri reallaşdırıla bilən olmadıqlarından "xeyirli", yəni daha yaxşı strategiya sayıla bilməz. Ona görə də mümkündür ki, "mütləq yaxşı" strategiya olmasa da proqramların testləşdirilməsi üçün həm ağ, həm də qara qutu strategiyalarından birlikdə istifadə olunsun.

Proqramların testləşdirilməsi zamanı əsas diqqət layihələndirməyə yaxud da effektiv testlərin yaradılmasına yönəldilir. Bu əsasən proqramların "tam" testləşdirilməsinin qeyri mümkünlüyü ilə əlaqədardır, yəni, ixtiyari proqram üçün test qeyri tam olacaqdır (başqa sözlə, testləşdirmə bütün səhvlərin olmadığına zəmanət verə bilməz). Layıhələndirmə strategiyasının əsas məqsədi ondan ibarətdir ki, testləşdirmənin "qeyri tamlığının" mümkün qədər azaldılmasına cəhd edilsin.

Əgər müəyyən parametrlərə, məsələn, vaxta, qiymətə, maşın vaxtına və sairəyə məhdudiyyətlər qoyulsa, onda testləşdirmənin əsas sualı aşağıdakı kimi olacaqdır:

Bütün mümkün testlərin hansı alt çoxluğunun proqramda ən çox sayda səhvlərin aşkar edə bilməsi ehtimalı ən böyükdür?

Testlərin layihələəndirilməsi metodologiyasının öyrənilməsi bu suala cavab verir.

Görünür ki, bütün metodologiyalardan ən zəifi (pisi) təsadüfi giriş qiymətləri ilə testləşdirmədir (stoxastik) – proqramın testləşdirilməsi prosesi bütün mümkün giriş kəmiyyətlərinin hər hansı təsadüfi alt çoxluğunun seçilməsi yolu ilə həyata keçirilir. Ehtimal nəzəriyyəsinin anlayışlarına görə təsdüfi seçilmiş testlər toplusu ilə çoxlu sayda səhvlərin tapılmasının optimal olması ehtimalı çox kiçikdir. Test verilənlərinin məntiqlə seçilməsinə bir neçə yanaşma mümkündür. Yuxarıda göstərdik ki, qara yaxud ağ qutu prinsipi ilə mükəmməl testləşdirmə ümumi halda

mümkün deyildir. Lakin həm də qeyd etmişdik ki, testləşdirmə strategiyasında hər iki yanaşmanın elementlərindən istifadə oluna bilər. Qara qutu strategiyasına əsaslanaraq, müəyyən layihələndirmə metogologiyasından istifadə etməklə kifayət qədər tam test yaratmaq mümkündür və sonra isə onu proqramın məntiqinin yoxlanılması ilə tamamlamaq olar (yəni, ağ qutu metodunu cəlb etməklə).

Aşağıdakı metodologiyaları nəzərdən keçirmək olar:

| Qara qutu | Ağ qutu |
|---|---|
| Ekvivalent bölünmə | Operatorların örtülməsi |
| Sərhəd qiymətlərinin analizi | Həllərin örtülməsi |
| Funksional diaqramların tətbiqi | Şərtlərin örtülməsi |
| Səhvlər haqqında mülahizələr | Şərtlərin kombinasiyalı örtülməsi |

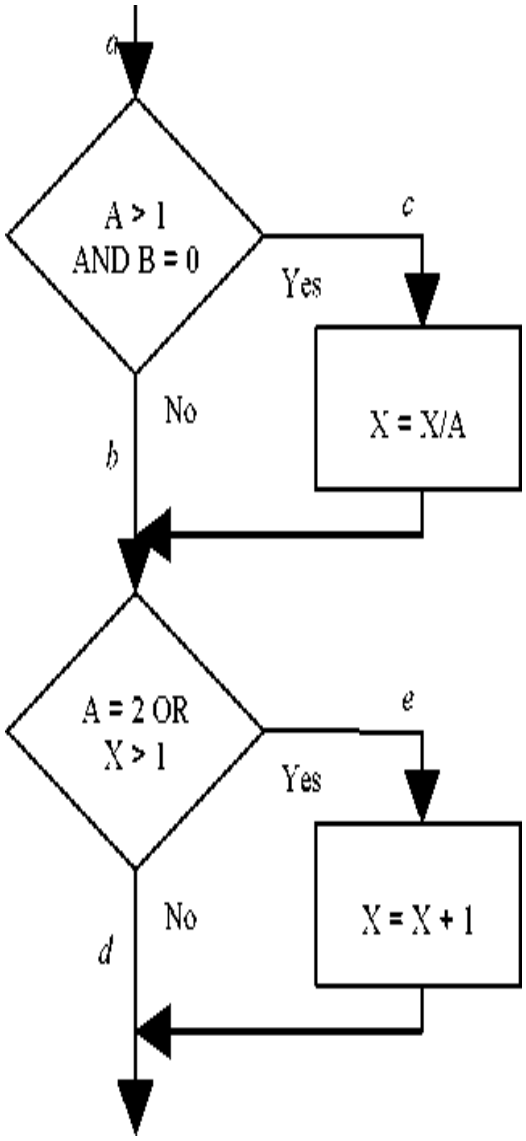
Effektiv testin layihələndirilməsi zamanı sadalanan metodologiyalardan hamısının istifadə olunması məsləhət görülməsə də, onlardan əksəriyyətinin istifadə olunması məsləhət görülür, belə ki, bu metodologiyaların hər birinin öz üstün və çatışmayan cəhətləri vardır (məsələn, müxtəlif tipli səhvlərin aşkar olunması və buraxılması). Bu metodların çox zəhmət tələb edən olduğunun doğru olmasına baxmayaraq, tətbiq olunmaları məsləhət görülür. Çünki testləşdirmə prosesi həddindən artıq mürəkkəb prosesdir. Bunu təsdiq etməkdən ötrü aşağıdakı aforizmi yada salmaq yerinə düşərdi: "Əgər siz fikirləşirsinizsə ki, proqramın yaradılması və kodlaşdırılması – çətin işdir, onda siz heç nə görməmisiniz".

Məsləhət görülən prosedur belədir ki, qara qutu metodunu tətbiq etməklə testləri yaratmaqdan ötrü, əlavə olaraq ağ qutu metodunu tətbiq etməklə testlər yaradılmalıdır.

Ağ qutu prinsipi ilə testləşdirmə hansı testlərdə proqramın (ilkin mətninin) məntiqinin örtülməsi yaxud yerinə yetirilməsi dərəcəsi ilə xarakterizə olunur. Ağ qutu prinsipi ilə mükəmməl testləşdirmə proqramda hər bir yolun yerinə yetməsini nəzərdə tutur; Lakin dövr saxlayan proqramlarda bütün yolların yerinə yetirilməsi mümkün olmadığından bütün yolların testləşdirilməsi nəzərdə tutulmur.

**Operatorların örtülməsi.** Əgər bütün yolların testləşdirilməsindən təmamilə etiraz olunarsa, onda demək olar ki, örtmə kriteriyası proqramın hər bir

operatorunun heç olmasa bir dəfə yerinə yetirilməsi hesab olunur. Təəssüf ki, bu çox zəif kriteriyadır, belə ki, hər bir operatorun heç olmasa bir dəfə yerinə yetirilməsi ağ qutu prinsipi ilə münasib testləşdirmə üçün zəruri şərt olsa da kafi şərt deyildir (şəkil 1.8).




Şəkil 1.8 Testləşdirilən proqramın alqoritmi

Fərz edək ki, şəkil 1.2 –də göstərilən çox da böyük olmayan proqramın testləşdirilməlidir. PL/1 proqramlaşdırma dilində yazılmış ekvivaletn proqram aşağıdakı şəkildədir:

```
M:   PROCEDURE (A, B, X);
     İF  ((A > 1)  &  (B = 0))  THEN  DO;
                    X=X / A;
     END;
     IF  ((A = 2)  |  (X > 1))  THEN  DO;
                    X=X + 1;
     END;
END;
```

***ace*** yolunu reallşdıran yeganə-bir test yazmaqla hər bir operatoru yerinə yetirmək olar. Başqa sözlə desək, əgər ***a*** nöqtəsində A=2, B=0 və X=3 qiymətləri qəbul olunarsa, onda hər bir operator bir dəfə yerinə yetirilər (həqiqətən burada X ixtiyari qiymət ala bilər.

Baxılan kriteriya, təəssüf ki, ilk baxışda göründüyündən pisdir. Məsələn, tutaq ki, ilk həll ***və*** kimi deyil ***yaxud*** kimi yazılmışdır. Bu kriteriyaya görə testləşdirmə zamanı bu səhv aşkar olunmayacaqdır. Tutaq ki, ikinci həll proqramda X > 0 şəklində yazılmışdır; bu səhv də aşkar olunmayacaqdır. Bundan başqa, proqramda elə yol vardır ki, X parametri dəyişmir (***abd*** yolu). Əgər burada da səhv vardırsa, o da aşkar olunmayacaqdır. Beləliklə, operatorların örtülməsi kriteriyası o qədər zəifdir ki, adətən bu kriteriyadan istifadə olunmur.

**Həllərin örtülməsi.** Proqram məntiqinin örtülməsinin daha güclü kriteriyası həllərin örtülməsi yaxud keçidlərin örtülməsi kimi məlumdur.

Bu kriteriyaya görə kifayət qədər elə testlər yazılmalıdır ki, həmin testlər toplusunda hər bir həllin nəticəsi heç olmasa bir dəfə ***həqiqi*** yaxud ***yalan*** qiymətlərini almalıdır. Keçid yaxud həll operatorlarına misal olaraq **DO** (yaxud Kobol dilində **PERFORM UNTİL**, Paskal dilində **REPEAT UNTİL**, **WHILE**, **Si** dilində **WHILE** və **DO WHILE**), **İF**, çox çıxışlara malik **GO TO** operatorlarını göstərmək olar.

Göstərmək olar ki, həllin örtülməsi kriteriyası adətən operatorların örtülməsi kriteriyasını qane edir. Hər bir operator müəyyən bir yolda, keçid operatorundan yaxud proqrama giriş nöqtəsindən başlayan yolda yerləşdiyindən, hər bir keçid

istiqaməti yerinə yetən zaman hər bir operator yerinə yetirilməlidir. Lakin ən azı iki istisna vardır. Birincisi – qeyri normal vəziyyət, proqramın həlli yoxdur. İkinci qeyri normal vəziyyət bir neçə giriş nöqtəsinə malik proqramlar yaxud alt proqramlarda rast gəlinir; baxılan operator yalnız o zaman yerinə yetirilə bilər ki, proqramın yerinə yetirilməsi uyğun giriş nöqtəsindən başlasın. Operatorların örtülməsi zəruri şərt sayılır. Daha güclü kriteriya sayılan həllərin örtülməsi operatorların örtülməsini daxil etməlidir. Beləliklə, həllərin örtülməsi tələb edir ki, hər bir həll nəticə etibarı ilə ***həqiqi*** yaxud ***yalan*** qiymətə malik olsun və bu zaman hər bir operator heç olmasa bir dəfə yerinə yetirilsin.

Yuxarıda şərh olunanların ikiqiymətli həllə yaxud keçidə malik olduğu nəzərdə tutulur və çoxqiymətli həll saxlayan proqramlar üçün modifikasiya olunmalıdır. Belə proqramlara misal olaraq aşağıdakı göstərmək olar:

- nişan-dəyişənin istifadə olunduğu **SELECT** (**CASE**) yaxud **GO TO** operatorları saxlayan **PL/ 1** dilindı proqramlar;
- hesablanan **GO TO** təlimata görə **GO TO** operatorları saxlayan **FORTRAN** dilində proqramlar;
- hesablanmaqla **ON – GOTO, ON – GOSUB** operatorları saxlayan **BASİC** dilində proqramlar;
- **ALTER**lə birlikdə **GO TO** yaxud **GO TO – DEPENDİNG – ON** operatorları saxlayan **KOBOL** dilində proqramlar;
- **CASE** operatorlarının istifadə olunduğu **PASCAL** dilində proqramlar:
- **SWITCH – CASE** operatorundan istifadə edən Si dilində proqramlar;
- **İF** hesabi operatorlarından istifadə edən ixtiyari proqramlar.

Şəkil 1.2 – də təqdim olunan proqramda həllin örtülməsi iki testin, ya ***ace*** və ***abd***, yaxud da ***acd*** və ***abe*** yollarını örtən testlərin köməyi ilə yerinə yetirilə bilər. Əgər biz sonuncu alternativ örtməni seçsək, onda iki testin girişləri A = 3; B = 0, X = 3 və A = 2, B = 1, X = 1 olacaqdır.

Həllərin örtülməsi, operatorların örtülməsinə nisbətən daha güclü kriteriyadır, lakin onun da öz çatışmayan cəhətləri vardır. Məsələn, X parametrinin qiymətinin

dəyişmədiyi yol 50% ehtimalla yoxlanılacaqdır. Əgər ikinci həlldə səhv varsa (məsələn, X > 1 əvəzinə X < 1), onda əvvəlki misaldakı iki testin vasitəsi ilə səhv aşkar olunmayacaqdır.

**Şərtlərin örtülməsi.** Həllərin ötürülməsi ilə müqayisədə şərtlərin ötürülməsi daha yaxşı kriteriya hesab olunur. Bu halda, həlldə olan hər bir şərtin bütün mümkün nəticələrinin heç olmasa bir dəfə yerinə yetirilməsi kafi olan testlərin sayını yazırlar. Çünki, həllərin örtülməsi halında olduğu kimi bu örtmə də həmişə hər bir operatorun yerinə yetirilməsinə gəriri çıxarmır. Kriteriyaya əlavə tələb olunur. Bu tələb ondan ibarətdir ki, proqrama yaxud alt proqrama, həmçinin ON-vahidlərin hər bir giriş nöqtələrində çağırılan zaman heç olmasa bir dəfə idarə ötürülməlidir. Məsələn, Fortran dilində dövr operatoru

DO K = 0 TO 50 WHILE (J + K < QUEST);

yaxud Si dilində bu operatorun analoqu

for (K = 0; K < = 50 && J + K < QUEST; K++) iki şərt saxlayır:
K < = 50 və J + K < QUEST.

Beləliklə, a nöqtəsində, A >1, A <= 1, B = 0 və b nöqtəsində A = 2, A ≠ 2, X > 1 olan halda vəziyyəti reallaşdırmaq üçün kifayət qədər test tələb olunur. Şərtlərin örtülməsi kriteriyasını ödəyən və onlara uyğun yollar aşağıdakı kimidir:

1. A = 2, B = 0, X = 4 – ace.
2. A = 1, B = 1, X = 1 – abd.

Qeyd edək ki, bu misal üçün kifayət qədər testlər yaradılmışdır, şərtlərin örtülməsi adətən həllərin örtülməsindən yaxşı hesab olunur, çünki bu kriteriya şərtlərdə olan həllərin yerinə yetirilməsini çağıra bilir (amma həmişə yox). Məsələn, yuxarıda nəzərdən keçirilən operanorlar ikiqiymətli keçidlidirlər ( ya dövrün mətni yerinə yetirilir, yaxud da dövrdən çıxış).

Baxmayaraq ki, ilk baxışdan elə görünür ki, şərtlərin örtülməsi kriteriyası həllərin örtülməsi kriteriyasını təmin edir, lakin bu həmişə belə deyildir. Əgər İF (A & B) həlli testləşdirilirsə, onda şərtlərin örtülməsi kriteriyası ilə testləşdirmədə iki test tələb olunardı – A *həqiqidir*, B *yalandır* və A *yalandır*, B *həqiqidir*. Lakin bu halda İF operatorunun THEN < operator > budağı yerinə yetməyəcəkdir.

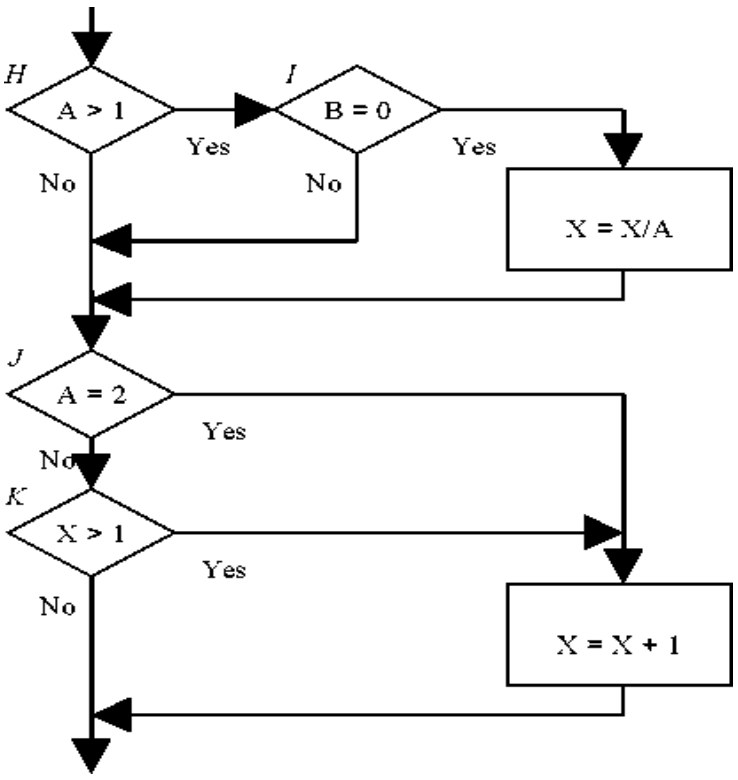
Əvvəldə baxılan misal üçün şərtlərin örtülməsi kriteriyasının testləri bütün həllərin nəticələrini örtür, lakin bu təsadüfən üst-üstə düşmədir. Məsələn, iki alternariv

1. A = 1, B = 0, X = 3
2. A = 2, B = 1, X = 1

testləri bütün şərtlərin nəticələrini örtür.

Bu dilemmalardan alınan nəticə həllərin-şərtlərin örtülməsi adlanan kriteriyadır. Həllərin-şərtlərin örtülməsi kriteriyası elə kifayət qədər testlər toplusu tələb edir ki, həlldə hər bir şərtin bütün mümkün nəticələri heç olçasa bir dəfə yerinə yetirilsin, hər bir həllin bütün nəticələri heç olçasa bir dəfə yerinə yetirilsin və proqramın hər bir giriş nöqtəsinə heç olçasa bir dəfə idarə ötürülsün.

Həllərin-şərtlərin örtülməsi kriteriyasının çatışmayan cəhəti ondan ibarətdir ki, onu bütün şərtlərin bütün nəticələrinin yerinə yetirilməsi üçün tətbiq etmək mümkün deyildir; çox vaxt belə yerinə yetmələr müəyyən şərtlərin digər şərtlər tərəfindən "gizlədilməsi" nəticəsində meydana çıxır. İlkin proqramın çoxşərtli həlli burada ayrı ayrı həllərə bölünür, çünki əksər maşınların çox çıxışa malik proqramları yerinə yetirmək üçün əmrləri yoxdur. Bu halda daha tam testlər vasitəsi ilə örtmə hər bir sadə mümkün həllin yerinə yetirilməsi ilə həyata keçirilir.




Şəkil 1.9 Testləşdirilən proqramın maşın kodu

# Fəsil II Proqram məhsulunun layihələndirilməsinin idarə olunması metodları

## 2.1 Proqram məhsulunun layihələndirilməsinin idarə olunmasının təşkili

Proqram məhsullarının yaradılması zamanı yardıcının fəaliyyəti, yəni görülən işlər, bir çox parametrləri ilə adı texniki məhsulun yaradılması zamanı görülən işlərlə üst-üstə düşür.

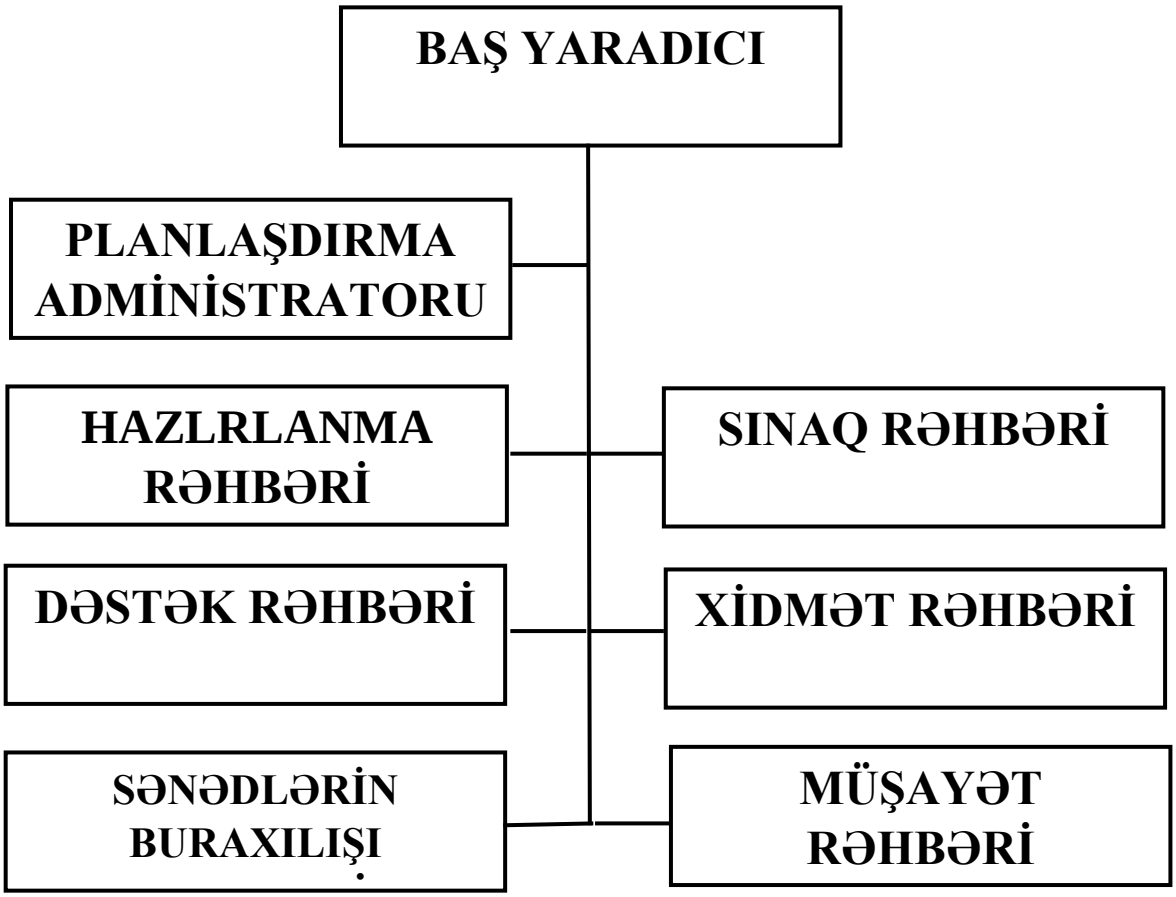
| Məhsul | Proqram |
|---|---|
| 1. Bazarın öyrənilməsi<br>2. Planlaşdırma<br>3. Xərclərin ödənilməsinin analizi<br>4. Hazırlama (istehsal etmə)<br>5. Konfiqurasiyalı idarəetmə<br>6. Keyfiyyətə zəmanətin təmini<br>7. Hazırlanma<br>8. Keyfiyyətə nəzarət<br>9. İstehlakçıya göndərmə<br>10. İşə salma<br>11. Zəmanətli təmir<br>12. Təkmilləşdirmə | 1. Bazarın öyrənilməsi<br>2. Planlaşdırma<br>3. Tələblər haqqında razılıqların nəzərdən keçirilməsi<br>4. Hazırlama (istehsal etmə)<br>5. Konfiqurasiyalı idarəetmə<br>6. Testləşdirmə (avtonom)<br>7. Surətini çıxarma (köçürmə)<br>8. Testləşdirmə (sistem)<br>9. İstifadəçiyə təhvil vermə<br>10. Işəsalma<br>11. Müşayət<br>12. Funksional imkanların genişləndirilməsi |

Cədvəldən göründüyü kimi, texniki məmulatların yaradılması zamanı görülən işlərin analoqu elə proqram məhsulunun hazırlanmasl zamanı görülən işlərin sırasında da vardır. Odur ki, proqram məhsullarını yaradarkən texniki məhsulların yaradılmasının idarə olunması təcrübasindən istifadə etmək olar.

Proqram məhsullarının layıhələndirilməsinin idarə olunması aşağıdakı funksiyaları özündə təcəssüm etdirir:

- planlaşdırma;
- hazırlanma;
- xidmət;
- sənədlərin buraxılışı (çap edilməsi);
- sınaq;
- kömək (himayə etmək);
- müşayət;

Proqram məhsulunun hazırlanmasının idarə olunmasının iyerarxik dekompozisiyası, yəni hissələrə necə ayrıldığı, aşağıdakı kimi göstərilə bilər.




Şəkil 2.1 İdarəetmənin dekompozisiyası

Görüləcək işlərin belə ideallaşdırılmış şəkildə təşkili layihələndirmə ilə əlaqədar olan proseslərin tam ayrılıqda nəzərdən keçirilməsini tələb edir. Layihələndirmə digər növ fəaliyyət addımlarında ayrılıqda nəzərdən keçirilir və onun əsas funksiyaları müəyyənləşdirilir. Aydındır ki, bunu praktikada reallaşdırmaq o qədər də asan deyildir.

Hər bir təşkilatın öz administratoru (direktoru) olmalıdır və bilavasitə bu şəxs hazırlanan məhsulun keyfiyyətli yaxud keyfiyyətsiz olmasına tam cavabdehdir. Strukturda bu və ya digər mühüm, o cümlədən, hazırlanma funksiyası da daxil olmaqla, fəaliyyət istiqamətləri də olmalıdır. Bu istiqaməti idarə edən şəxs təşkilatın istehsal etdiyi məhsulun hazırlanmasının bütün aspektlərinə məsuliyyət daşıyır. Proqram məhsulunun hazırlanma prosesinin əlaqələndirilməsindən ötrü, bu şəxs məhsulun administratorlarını, layihələrin rəhbərlərini təyin etmək və onların qarşılıqlı əlaqəsini təşkil etmək hüququna malikdir.

Qarşılıqlı əlaqələr lazım olan səviyyədə müəyyən edildikdə bu əlaqələrin idarə olunması da uyğun səviyyədə təşkil olunur. Əgər qarşılıqlı əlaqə pis təşkil olunubsa, onda hətta ciddi xətti tabeçilik strukturu olduqda belə son məmulatın yaradılması çətin olacaqdır.

Məlumdur ki, ixtiyari növ idarəetmənin əsas prinsipi tamı, yəni hər hansı bir obuekti yaxud sistemi, hissələrə bölməkdən ibarətdir, və bir çox metod və vasitələr bu prinsipə əsaslanır.

Bir biri ilə qarşılıqlı fəaliyyətdə olan iki funksional qrupdan ibarət "nöqtələr toplusu" ***təşkilati sərhədlər*** adlanır. Bəzən funksiya yerdə qalan bütün digər funksiyalarla yeganə qarşılıqlı əlaqə kanalına malik olur, lakin çox güman ki, o bir neçə toxunma (qonşuluq) sərhəddinə malikdir. Funksiyanın sərhədləri, onun funksional vəzifələrini müəyyən edən, yəni yerinə yetirəcəyi funksiyaları müəfyyən edən, qeydə alınmış və qeydə alınmamış planlar, strategiyalar, prosedurlar çoxluğu ilə müəyyən olunur. Yazılı şəkildə nə qədər çox məlumat qeydə alınarsa, bu yaxşı hal hesab olunur, çünki bu ikimənalılığı, yəni məlumatların müxtəlif mənalarda başa düşülməsini azaldır. ***Təşkilati sərhədlərin*** əsas xüsusiyyəti cavabdehliyin dürüst müəyyən edilməsidir (kim və nə edir, necə

edir, nəyə görə edir və sair). Müəyyənliyin tam olmaması, ikimənalılıq, və mürəkkəblik qarşılıqlı fəakiyyətin düzgün şərh olunmasına və bu səbəbdən də onun təbiətini başa düşməsinə imkan vermir.

Əgər funksional qruplar arasında münasibət yoxdursa, onda nə sənədlərin özləri, nə də kollektiv müzakirə aktiv qarşılıqlı əlaqəni təmin edə bilməz.

Ümumi təşkilati vəzifələr bölmənin məqsəd planları və vəzifə təlimatlarının köməyi ilə müəyyən oluna bilər. Konkret vəzifələr məhsulun buraxılış planı ilə müəyyənləşdirilir. Məsuliyyətin proporsional, yəni mütənasib bölünməsi uyğun idarəetmə strategiyası ilə müəyyən olunur. Bu idarəetmə strategiyasında mütləq götürülmüş öhdəliklərin yerinə yetirilməməsinin mümkünlüyü nəzərə alınmalı, xüsusi hallarda düzəlişlərin ediləcəyinin qaçılmaz olduğu nəzərdən qaçmamalıdır ki, planlar və prosedurlar fəaliyyət göstərə bilsinlər.

Məqsədləri qarşıya qoymaq və bu məqsədlərə çatmaq prosesində birinci addım zəruri personalın (işçi heyətin) seçilməsidir. Təşkilatda dəyişiklik baş verən zaman, aydındır ki, bu təşkilatda çalışan personal da dəyişir. Bəzi dəyişmələr periodik olaraq baş verir. Belə dəyişmələr müəyyən əlaqələrin sayının eksponensial yüksəlişi ilə xarakterizə olunur. Baş verən dəyişikliklərə uyğunlaşmaq üçün adaptasiyaya qadir olan rəhbərlər olmalıdır. Yalnız bir fəaliyyət sahəsində məhsuldar olan rəhbərləri dəyişmək lazımdır. Proqramlaşdırma - yüksək ixtisas tələb edən sahədir, o özünə müxtəlif tərkibli, qeyri adi insanları cəlb edir. Belə adamlar proqram məhsulu yaratmağa istiqamətlənmiş təşkilatın sturukturunu çox təsadüfən başa düşürlər. Çox vaxt yaradıcılıq fəaliyyətinin məhdudiyyəti şərtləri daxilində işləməkdən imtina edirlər. Çünki onlar öz yaratdıqları məhsulu müdafiə etməli, yəni yazdıqları proqramların keyfiyyətli olduğunu, sifarişçinin tələblərinə cavab verdiyini sübut etməli və sənədləşmə işlərinə xeyli vaxt sərf etmək macburiyyətindədirlər. Lakin bu “qara işdə” onların iştirakı zəruridir və onları işə yönəltmək üçün əffektivlik mülahizələrini rəhbər tutaraq stst cədvəlini komplektləşdirmək lazımdır. Bu isə qoyulmuş ideal qaydalardan kənarlaşmaq, işə gəlib getməkdə tam azad rejim təklif etmək, belə adamlara onların öz nəticələrini dəqiq sənədləşdirməyi bacaran köməkçilər

ayırmaq deməkdir. Lakin bu zaman həvəsləndirmə ilə bağlı olan "üstün" hesab edilən işçilərin birbaşa xərclərini, həmçinin əməkdaşların mənəvi cəhətdən vəziyyətlərinin pisləşməsi ilə əlaqədar olan qeyri aşkar xərcləri müqayisə etmək lazımdır.

Hazırlanmanın idarə olunmasının əsas metodiki prinsipi ***məqsədli idarəetmə***dir.

***Məqsədli idarəetmə*** dedikdə rəhbərin uyğun məqsədləri özünün bilavasitə daha yüksək ranqlı rəhbəri vasitəsi ilə müəyən və bu müəyyən etmədə sonuncu iştirak etdiyi planlaşdırma və idarəetmə konsepsiyası başa düşülür; bu zaman onun fəaliyyəti konkret müzakirə və sənədlərlə təstiqlənmiş rəylər əsasında qiymətləndirilir.

Məqsədlər planda ən müxtəlif səviyyələrdə ola bilirlər: məqsədli planlarda, büdcədə, məhsul buraxılışı planında, sənədləşmədə və sair. Proqram məhsulu üçün əsas plan – tələblər haqqında sazişdir (razılaşmalardır). Bu planda ifadə olunmuş məqsədlər fərdi iş planına daxil edilməlidir. Bu işçi planlar sistemdə məqsədli idarəetmənin icraçılar və onların rəhbərləri arasında əldə olunan razılaşmalar mexanizmi rolunu oynayır.

Məqsədə çatmaya fərdi işçi planlarina məmulatların hazırlanmasının ümumi məqsədinin daxil edilməsi və onların yerinə yetirilməsinə cari nəzarətin təşkili ilə zəmanət verilir.

Tələblər haqqında razılaşmalar, dəstək planı, büdcənin bölüşdürülməsi və digər vasitələr, əks əlaqələrin istifadə olunmasını tələb etməyən sərhədləri müəyyən edirlər.

Bu və ya digər fəaliyyətin effektivliyinin qiymətləndirilməsinin üç növ əsas kriteriyası vardır:

- konkret xassələr;
- sərf olunan vaxt;
- dəyər.

Bu kriteriyalardan hər birinin müəyyən fəaliyyət sərhədləri vardır. Həmin sərhədlərə çatdıqda nütləq şəkildə rəhbərliyə alınan nəticələr haqqında hesabat təqdim olunur.

Lakin nəzərə almaq lazımdır ki, proqram məmulatlarının hazırlanmasının idarə olunması, qeyri müəyyənlik şəraitində idarəetməyə ən yaxşı misaldır. Belə idarəetmənin keyfiyyəti rəhbərin çətinlikləri əvvəlcədən görə bilməsi, təsadüfi faktorları nəzərə almaqla hazırlamanı planlaşdırması, bu nöz planlaşdırmanın rəhbərliyin tənqidlərindən müdafiə etməsi bacarığından asılıdır.

Proqram vasitələrinin hazırlanması kifayət qədər mürəkkəb prosedur olduğundan, onun reallaşdırılması üçün yüksək ixtisaslı mütəxəsislər lazəmdır, başqa sözlə desək, proqram təminatının layihələndirilməsi üçün işin bütün sahələri üçün yüksək istisaslı mütəxəsislərin ayrılması zıruridir. Elə admlar axtarmaq lazımdır ki, belə funksiyaları kifayət qədər yaxşı yerinə yetirsin, yaxud da nisbətən az çətinlikli funksiyaları kim daha savadlı şəkildə yerinə yetirə bilər.

Layihələrin rəhbərliyini yalnız böyük təcrübəyə malik mütəxəsislərə tapşırmaq olar. Bu zaman baş proqramçının briqadası konsepsiyası çox yaxşı iş görə bilir.

Axı insanlar haradasa işləməyə başlamalıdırlar. Çünki hazırda universitetlərdə hazırlıq kifayət qədər ciddidir (struktur, obyekt-yönümlü proqramlaşdırma və sair), onda problem proqramçıların yerini universitetlərin məzunları tutmalıdırlar.

Çox böyük ehtimalla ixtisaslaşmış personalı universal proqram təminatı təchizatçıları arasında tapmaq olar. Namizədin malik olmalı olduğu əsas əlamət – nizam intizama tabe olmaq qabiliyyətidir. O, proqram məhsulunun yaradılmasında ***enən yanaşmanın***, yəni ***aşağı düşən yanaşmanın*** və hətta kodlaşdırmaya qədər proqramların sənədləşdirilməsinin vacibliyini başa düşməlidir.

Ona diqqət yetirmək lazımdır ki, ixtiyari proqramçının işinin nəticəsi digərlərinə də aydın olmalı və bu nəticələrdən istifadə etmək mümkün olmalıdır. Lazım olan işi yüksək ixtisasa, bilik və bacarığa malik olmayan mütəxəsislə yerinə yetirmək cəhdi əlbəttə ki, uğursuzluqla nəticələnir.

İşlərin müvəffəqiyyətlə yerinə yetirilməsi üçün mühüm faktor xidmət üzrə irəliləyişin təmin olunmasıdır. Ştatda olan vəzifə instansiyalarından (pillələrindən) maksimum dərəcədə istifadə etmək məqsədəuyğundur. Məsuliyyətin yüksəldilməsini müəyyən edən uyğun formal vəzifə təlimatlarını tərtib etmək, bir səviyyədə olan vəzifələr üçün eyni ixtisas dərəcəsinə malik olan işçilərin seçilməsi mütləqdir.

Obrazlı desək, xidməti pilləkənlə şaquli irəliləməkdən başqa , üfiqi səviyyə ilə də irəliləyişi təmin etmək lazımdır. İxtiyari səviyyəyə malik mütəxəsisi işə qəbul edən zaman, onu əlavə olaraq professsional tədrisə cəlb etməklə, ona ixtisasını artırmaqda köməklik göstərmək, bilmədiklərini daha savadlı mütəxəsislərdən öyrənə bilməsinə şərait yaratmaq və beləliklə onun xidməti pilləkənlə yüksələ bilməsinə şərait yaratmaq zəruridir. Yerinə yetiriləcək hər bir funksiyaya görə seminarlar təşkil olunmalıdır. Bu seminarlara müxtəlif ixtisas səviyyəsinə malik dinləyicilərin cəlb olunması vacibdir. Seminarın işini elə təşkil etmək lazımdır ki, bu seminarlarda layıhələndirmə üçün böyük əhəmiyyət kəsb edən çox unikal hesab edilən vasitələr müzakirə olunsun. Bu seminarlarda iştiraka marağı stimullaşdırmaq üçün müəyyən vəzifəyə təyinat zamanı həmin şəxsin bu seminarda iştirakı vacib şərt hesab olunmalıdır.

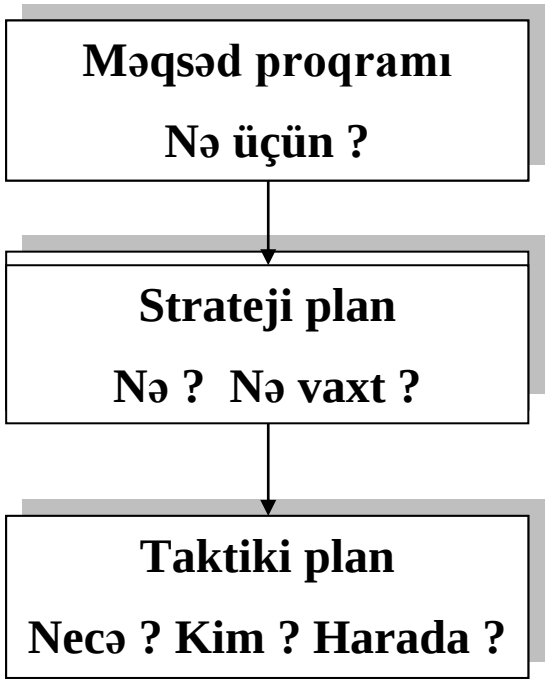
Təhsilin uyğun təşkilatdan kənarda aparılması da az əhəmiyyət kəsb etmir. Qabaqcıl proqramlaşdırma məktəblərinin təşkil etdiyi seminarlarda ixtisasa uyğun olaraq layihədə işdtirak edən işçilərin iştirakı zəruri hesab olunur. Təbiidir ki, bu seminarlarda iştirak etmək böyül məbləğdə xərc tələb edir. Lakin təşkilar əmin olmalıdır ki, işçisinin təhsili üçün çəkdiyi xərc hədər getməyəcəkdir və təşkilata daha çox kapital gətirməklə iş effektivliyini xeyli yüksəldəcəkdir.

## 2.2 Proqram məhsulunun yaradılma prosesinin planlaşdırılmasının təşkili

Proqram məhsulunun yaradılması planı, yaradılma, sənədləşdirmə, sınaq, istifadəçilərin öyrədilməsi, müşayət mərhələlərini əhatə etməlidir. Planların olmaması proqramların dəyişdirilməsinin, yəni yenidən işlənməsinin əsas səbəbidir. Təbiidir ki, planlarda bütün kateqoriyalardan olan istifadəçiləri nəzərə almaq mümkün deyildir, lakin görünməz vəziyyətlərdən qorunmaq üçün ehtiyat tədbirləri görmək lazımdır.

Proqram məhsulu elə təkcə proqramın özü deyildir, buraya hazırlanan proqramla yanaşı ona aid sənədlər, proqramın keyfiyyətinə zəmanət, reklam materialları, təhsil (təlim), məhsulun yayılması və müşayət də daxildir. Beləliklə, proqram məhsulunun yaradılması üçün planlar toplusu tələb olunur. Bu planlar məhsulun hazırlanmasının bütün aspektlərini və onların xarici nühitlə əlaqələrini əhatə edir.

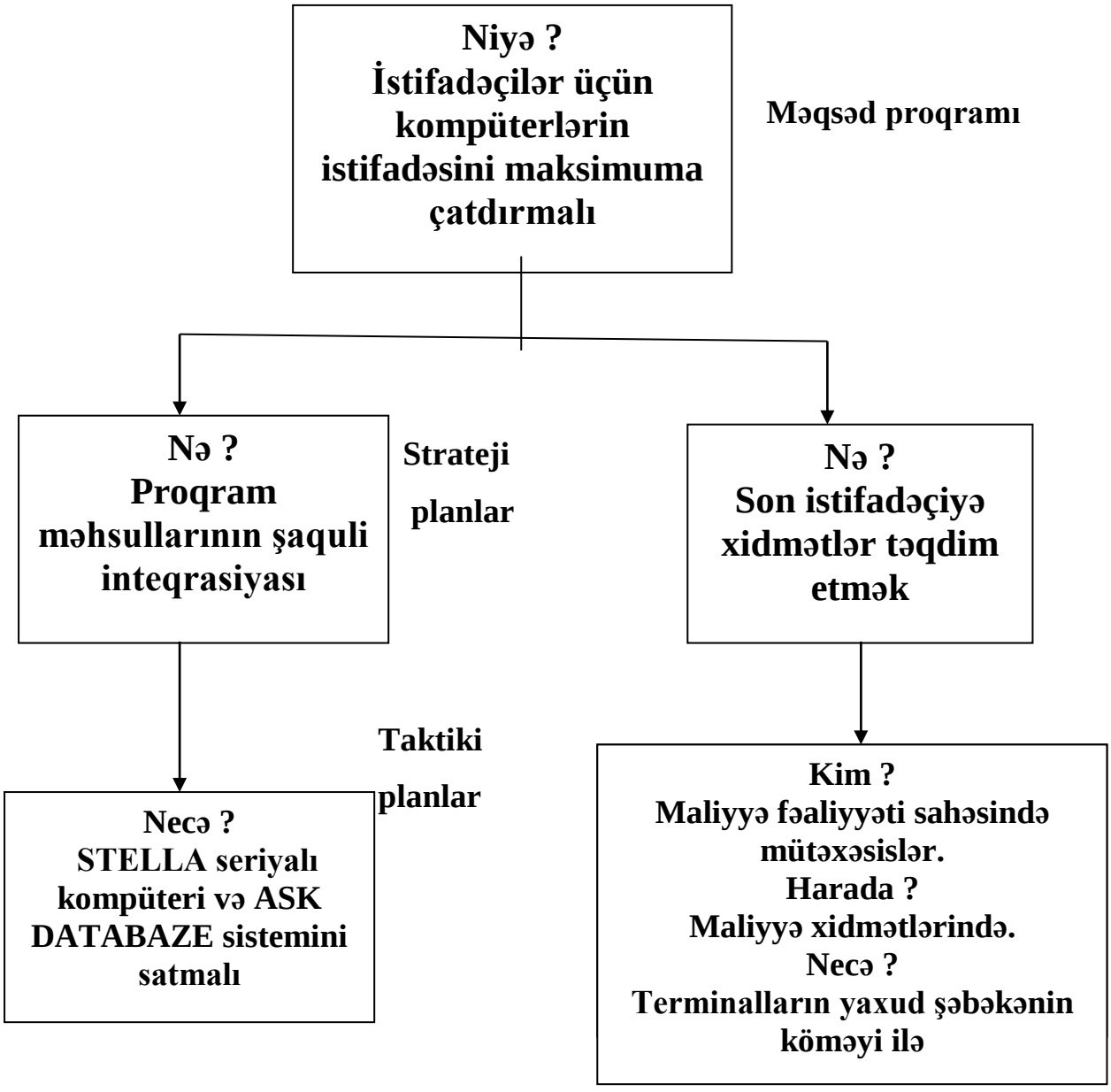
**Planların növləri.** Ən böyül əhatəyə malik plan məqsəd proqramıdır. Onun məqsədi – kapital qoyuluşundan gəliri maksimuma çatdırmaq. Məqsəd proqramı bu və ya digər fəaliyyət ***"nə aüçün təyin olunmuşdur ?"*** sualına cvab verir.




Şəkil 2.2 Planların iyerarxiyası

Məqsəd proqramında, məqsədə necə çatmaq lazım olduğu haqqında məlumat verilmir və çox təsadüfi hallarda realizasiya meddəti müəyyən olunur. Adətən məqsəd proqramları təsadüfi hallarda müstəqil mənaya, yəni əhəmiyyətə malik olurlar, əslində onlar ***"nə üçün"*** sualına cvab vermək üçün təyin olunmuşlar.

Planlaşdırmanın növbəti səviyyəsi – strateji plan, əsas planlardan biri hesab olunur. Bu plan ***"Nə ?"*** və ***"Nə vaxt ?"*** suallarına cavab verir. Bu zaman məqsəd nə qədər dəqiq olarsa, bu məqsədə çatmaq strategiyası da bir o qədər özünəməxsusluğu ilə seçiləcəkdir.




Şəkil 2.3 Strategiyanın işlənməsi

**Taktiki planlar nisbətən konkret hesab olunurlar. Bu planla**r ***kim ?*** və ***harada ?*** suallarına cavab verir, başqa sözlə qoyulmuş məqsədə çatma üsullarını dəqiqləşdirir.

Strateji plan qoyulmuş məqsədə və nə vaxt çatmaq üçün nə etməyin zəruri olduğunu formalaşdırır. Taktiki plan işin kim tərəfindən, necə və nə vaxt yerinə yetiriləcəyini göstərir.

*Misal.*

Firmanın məqsədi – çox da böyük sayda olmayan insanlar üçün kompüterin istifadəsini mümkün etməli. Proqram məhsullarının şaquli inteqrasiyasını həyata keçirilməsi strategiyalardan biridir. İstifadəçilərə (iqtisadçılara, fiziklərə və s.) bilavasitə xidmətlər təklif etmək isə başqa bir strategiyadır. Üçüncü strategiya isə müxtəlif şəkilli verilənlər ( iqtisadi situasiyalar, satış bazarı) üçün hazırlanmış proqram məhsulunun yaradıcının özü tərəfindən istifadə olunmasından ibarətdir.

Hər bir proqram məhsuluna nəzərdən keçirilən strategiyaların taktiki həlli kimi baxmaq olar (sək. 2.3).

Adətən məqsəd proqramı, strateji və taktiki planlar çox təsadüfən müstəqil element kimi fəaliyyət göstərirlər. Lakin planlar toplusu formalaşdırılan zaman,qoyulmuş məsələnin asan başa düşülməsini asanlaşdırmaq üçün, bu planların hər birini göstərilən üç kateqoriyaya görə təsnifləşdirmək zəruridir.

Bir neçə konkret plana baxaq.

Büdcə - bu elə plandır ki, ümumiyyətlə desək, burada “niyə”, “necə”, və “nə” sualları müəyyən olunmur. Büdcə planında “nə” sualı ilə bağlı mühüm kəmiyyət məhdudiyyətləri müəyyən olunur. Bu ən çox yayılmış plandır. Büdcənin hesabına real ətraf mühitin əks təsiri kifayət qədər tez və dəqiq olur.

Xüsusi növ başqa bir plan təqvim planıdır. Vaxt da pul kimi ən müxtəlif növ fəaliyyətin qiymətləndirilməsinin ümumi faktoru ola bilər.

İşlərin fərdi planları məqsədli idarəetmənin əsas aləti hesab olunur. Bu planlar konkret tərtib edilən məqsədlərin və işlərin sona çatdırılması vaxtı, nəzarəti yaxud da işlərin dayandırılmasını təmin edirlər. Yəni, fərdi planlar büdcəyə əsasən

məqsədli, strateji və taktiki planları özündə cəmləşdirir. Bu təsiredici vasitə proqram məhsullarının hazırlanmasının "ruhlandirma"sı hesab olunur və proqram məhsullarının hazırlanmasında konkret iştirakçıların rolunu müəyyən edir.

Şəbəkə qrafiki əsasən taktiki plan hesab olunur. Şəbəkə qrafiki, tapşırıq (iş) anlayışları ilə desək, məqsədə necə çatmaq olar və konkret işlərin görülməsinə kim cavabdehdir suallarına cvab verir. Bu qrafikdə də strateji planda olduğu kimi işlərin sona çarma tarixi müəyyənləşdirilir.

***Planların dekompozisiyası***. Planlaşdırmada (planlar tərtib olunan zaman) aşağıya enmə planlaşdırma sxemindən istifadə olunur. Bu planların dekompozisiyası adlanır:

- proqram məmulatları ailəsinin yaradılması planı;
- seriyaların yaradılması planı;
- məmulatlar toplusunun yaradılması planı;
- konkret məmulatın istehsalı planı.

Planın mükəmməlliyi o deməkdir ki, bütün zəruri suallar nəzərdən keçirilmişdir və eyni zamanda detallaşdırmanın zəruri səviyyəsi təmin olunmuşdur. İşə aidiyyəti olan bütün suallar detallaşmanın ən yüksək səviyyəsində və iyerarxiyanın ən aşağı səviyyəsində müzakirə olunmalıdır. Detallaşmanın lazım olan səviyyəsini müəyyən etmək, həmçinin bütün sualları əhatə etmək çox çətindir. Ən ciddi səhv mövcud olan səhvlərin buraxılmasıdır, başqa sözlə desək, nəzərə alınmamasıdır. İkinci böyük səhv – həddindən artıq detallaşdırmadır ki, bu da "içərisində batıb qalmağa" gətirib çıxarır. Belə səhvləri aradan qaldırmaq üçün əvvəlcədən tərtib olunmuş suallar siyahısından yaxud müəyyən formadan istifadə etmək lazımdır. Həddindən artıq detallaşdırma səhvindən qaçmaq üçün hər dəfə "detallaşdırmanı davam etdirməyə ehtiyac varmı ?" sualını vermək lazımdır.

Planların dekompozisiyası zamanı aşağıdakı prinsipə əsas prinsip kimi baxmaq lazımdır: "Planlaşdırmanın növbəti addımında hansı seçmə azadlığı arzuolunandır ?". Adətən planlaşdırmanın növbəti mərhələləri bütün daha aşağı səviyyələrdə yerinə yetirilir və yalnız onların səlahiyyət oblastını nəzərə almaq kifayətdir.

Bütün real iyerarxik planların strukturu mütləq üst – üstə düşməlidir. Planın hər bir bəndinin detallaşdırma dərəcəsi iyerarxik olaraq aşağı endikcə artmalıdır. Lakin mövcud olmayan detalların müzakirəsinə çox böyük vaxt planları qeyri real edir.

Dekompozisiyanın ikinci prinsipi – növbəti, daha aşağı səviyyə üçün, "Hər hansı proqramlaşdırma dilindən bir il ərzində translyator yarat və 300.000 $ -dan çox vəsait xərcləmə", məhdudiyyətləri müəyyən etməli. Burada üç məhdudiyyət toplusu vardır:

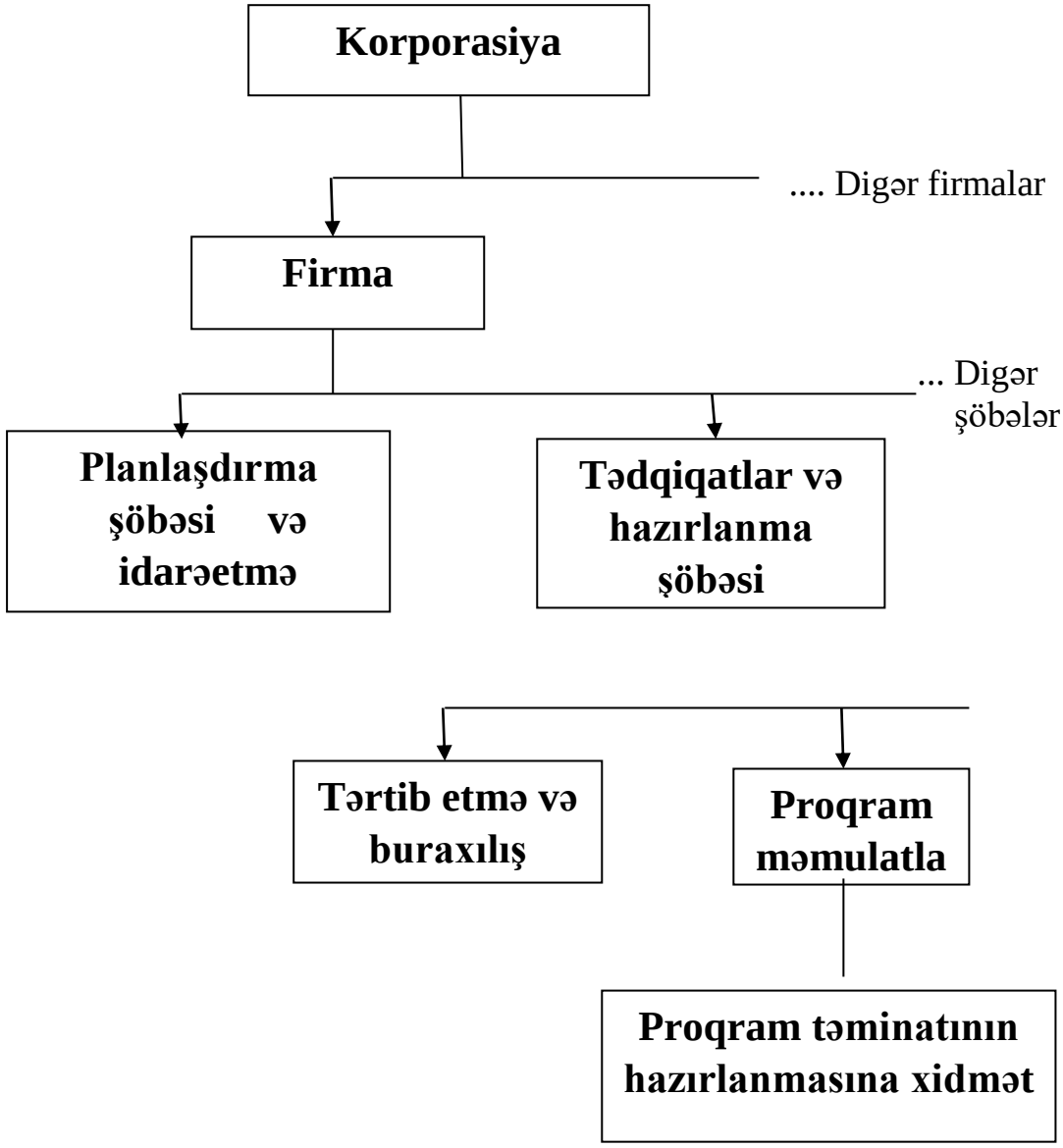
- translyator konkret dilin təlimatlarını emal etməlidir;
- yaradılma üçün bir ildən çox olmayaraq yaxt ayrılmışdır;
- büdcə - 300.000 $ - dan çox deyildir.

Heç bir özgə ehtiyat məhdudiyyəti yoxdur. Bu çərçivədə azad seçim mövcuddur. Əgər planın dekompozisiyasıdavam etdirilərsə, onda çoxsaylı məhdudiyyətlər əmələ gələ bilər. Beləliklə, ən aşağı səviyyədə planların mütləq detalizasiyası təmin olunacaqdır. Lakin bu zaman, əgər səlahiyyətlər düzgün bölüşdürülmüşdürsə, başlanğıc məhdudiyyətlərdən heç biri pozulmayacaqdır.

Təbiidir ki, müxtəlif növ planlar müxtəlif səviyyəli rəhbərliklərə uyğundur. İxtiyari yaradıcı, yəni məhsul hazırlayan təşkilatda xüsusi bölmə yaradılmalıdır. Bu bölmənin yeganə vəzifəsi təşkilatın bütövlükdə planlaşdırılması və idarə edilməsi olmalıdır. Həm də bu bölmə, bütün digər bölmələr üçün planların olduğuna, onların uyuşan olduğunu təmin edəcəyinə və qarşılıqlı əlaqədə olduqlarına zəmanət verməlidir. Bunun üçün təşkilatın strukturu və fəaliyyəat dairəsi dərindən başa düşülməlidir. Bu funksiyaları yerinə yetirən adamların eyni zamanda heç bir başqa vəzifəsi olmamalıdır. Planlar elə qanunlar kimidir. Onlar sözsüz yerinə yetirilməlidirlər. Yəni planlaşdırmanı və nəzarəti həyata keçirən bu adamlar, uyğun səlahiyyətlərə malik olmalıdırlar. Planın idarə olunması da elə planlaşdırmanın özü kimi dəqiq təşkil olunmalıdır.

Planların yerinə yetirlməsinə rəhbərlik edən məsul şəxslər, planlaşdırmanın administratorları adlanırlar. Lakin, unutmaq lazım deyildir ki, planlaşdırma qruplarında həddindən çox adamların olması bürokratiyanın əmələ gəlməsinə

səbəb olur, odur ki, planlaşdırma qrupu konkret-istehsal ixtisaslaşması prinsipinə görə təşkil olunmalıdır. Vəzifə təlimatlarında hər bir funksional qrupun planlaşdırma qrupu ilə qarşılıqlı əlaqədə olduğu yeganə - bir təmas nöqtəsi dəqiq müəyyən olunmalıdır (şək. 2.4).




Şəkil 2.4 Planlaşdırma qrupu ilə funksional bölmələrin qarşılıqlı fəaliyyət sxemi

Bir çox təşkilatlarda planlaşdırma administratoru rolunu məqsəd proqramının rəhbəri yerinə yetirir.

Planlaşdırma konfiqurasiyalı idarəetmə prinsiplərinin düzgün tətbiq olunduğuna verilən zəmanətə söykənməlidir. Bu məqsədlə, saxlanma prosedurları və layihə sənədlərinin yayılması da daxil olmaqla, sənədlərə nəzarət funksiyası daxil edilir.

Eyni zamanda planlaşdırma yalnız yuxarıya iyerarxiyaya görə mümkün deyildir. Ona görə də hər bir funksional qrupun öz xüsusi taktiki planlaşdırma heyəti olmalıdır. Onların qarşılıqlı fəaliyyəti aşağıdakı sxem üzrə qurulmalıdır:

- planlaşdırma bölməsi məqsədli və strateji planlaşdırma, büdcənin idarə olunması və bütün təşkilat üçün planlara cavabdehdir;
- funksional qrup taktiki planlaşdırmaya cavabdehdir.

Aşağı enən planlaşdırma istifadə olunduqda tədricən ümumidə xüsusiyə (məqsəd proqramından strateji və taktiki planlara) və qlobaldan lokala (məmulatlar ailəsi planından, seriyanın baraxılışı planı və məmulatlar toplusundan keçməklə konkret proqram məhsulu layihəsinə) keçid baş verir. Müxtəlif planların strukturlarının düzgün başa düşülməsini asanlaşdırmaq üçün "standart" tərifləar daxil edilir.

Proqram məhsulu – ayri-ayri proqram vasitələri toplusu, onların sənədləri, keyfiyyət zəmanəti (İSO9000), reklam materialları, istifadəçilərin təlim ölçüsü, proqram təminatının yayılması və müşayət olunması. Proqram təminatının tam yəni bütöv məhsul, yaxud da onun xüsusi modifikasiyasından asılı olmayaraq, adətən məhsul, bütün yuxarıda nəzərdən keçirilən elementlərə nisbətən ən kiçik obyekt hesab olunur.

Məhsullar toplusu – bir və ya bir neçə ümumi xarakteristikaya malik və müəyyən kombinasiyalarda birlikdə işləyən məhsullar qrupu ( əməliyyat sistemi və əməliyyat sistemi ilə idarə olunan kompilyatorlar, servis proqramları, diaqnostika vasitələri belə qrup təşkil edir).

Məhsullar seriyası – bir və ya bir neçə ümumi əlaqəyə malik və müəyyən kombinasiyalarda müstəqil sistem kimi fəaliyyət göstərən aparat və proqram məhsullarının birləşdirilmiş forması.

Məhsullar ailəsi – bir neçə əlaqəyə malik proqram məhsullarıdır. Məcburi deyildir ki, onlar mütləq ümumi interfeysə malik olsunlar və eybi bir aparat platformasında fəaliyyət göstərsinlər (məsələn, bəzi təchizatçılar tərəfindən müxtəlif EHM-lər üçün yaradılmış FORTRAN dilinin kompilyatorları).

Ən yüksək səviyyə plan məhsullar ailəsi yaxud məhsullar seriyası hesab olunur. Çünki, onların adında məhsul sözü vardır – bu strateji plandır. Bütünlükdə o, texniki vasitələr, proqram təminatı, personalın təlimi məsələləri anlayışları vasitəsi ilə formalaşdırılır. O strateji elementlərdən ibarətdir:

- bazara asanlıqla yol tapa bilən bir-biri ilə rəqabətdə olan məhsulların uyuşanlığını necə təmin etməli;
- məhsulun həyat fəaliyyətinin uzadılması məqsədi ilə təkmilləşdirilməsinin dövrülülüyü və sair.

Adətən strategiyanın elementləri uzunmüddətli intervalı əhatə edirlər (5-10 il).

Seriya planı bəyənilən kimi məmulatlar toplusunun buraxılması planı hazırlanır. Belə planların təqdim olunmasının ən yaxşı vasitəsi konfiqurator hesab olunur. Bu anlayış əməliyyat sistemləri arasındakı qarşılıqlı əlaqəni və ona tabe olan məmulatlar toplusu elementlərini qısa şəkildə xarakterizə edən cədvəli təyin etmək üçün istifadə olunur.

Konfiquratorun yığcam forması idarəetmənin yuxarı səviyyələrinə böyük miqdarda informasiya ötürməyə imkan verir.

Proqram məmulatlarının hazırlanması üzrə tədqiqatlar başlanan zaman, əvvəlcə görüləcək işlərin təxmini planı hazırlanır, ilkin vəsaitlərin ayrılması baş verir, şi sona çatdırmaq üçün əsas fondlar qeydə alınır. Bu zaman hazırlanan sənəd, büdcənin bölünməsindən başqa bir şey deyildir. Bu sənəd artan maliyyələşdirilməsi konsepsiyasını reallaşdırır, o planların yerinə yetirilməsinə nəzarəti təmin edir.

Cədvəl 1. Konfiqurator

| | **Proqram məmulatları toplusu** | | | | | | |
|---|---|---|---|---|---|---|---|
| | **VSOS2** | | | **VSOS3** | | **VSOS4** | |
| Proqram məhsulunun adı | sə hi fə | və zi yy ət | sə vi yə | və zi yy ət | sə vi yə | və zi yy ə | sə vi yə |
| VSOS2<br>VSOS3<br>VSOS4 | 4<br>4<br>5 | 7.0<br>///<br>/// | 3 | ///<br>7.07<br>/// | 2 | ///<br>///<br>7.0 | 1 |
| /// - məhsuldan istifadə mümkün deyil<br><br>tarix – ay və il, məhsuldan nə vaxt istifadə etmək olar | 1 – bildiriş vasitəsi ilə aşkar olunmuş qüsurlar haqqənda məlumat verilir, imkanların genişləndirilməsinə baxılır.<br>2 - bildiriş vasitəsi ilə aşkar olunmuş qüsurlar haqqənda məlumat verilir, imkanlarən genişləndirilməsi qəbul olunmur.<br>3 – yalnız qüsurlar haqqında bildirişlər emal olunur. | | | | | | |

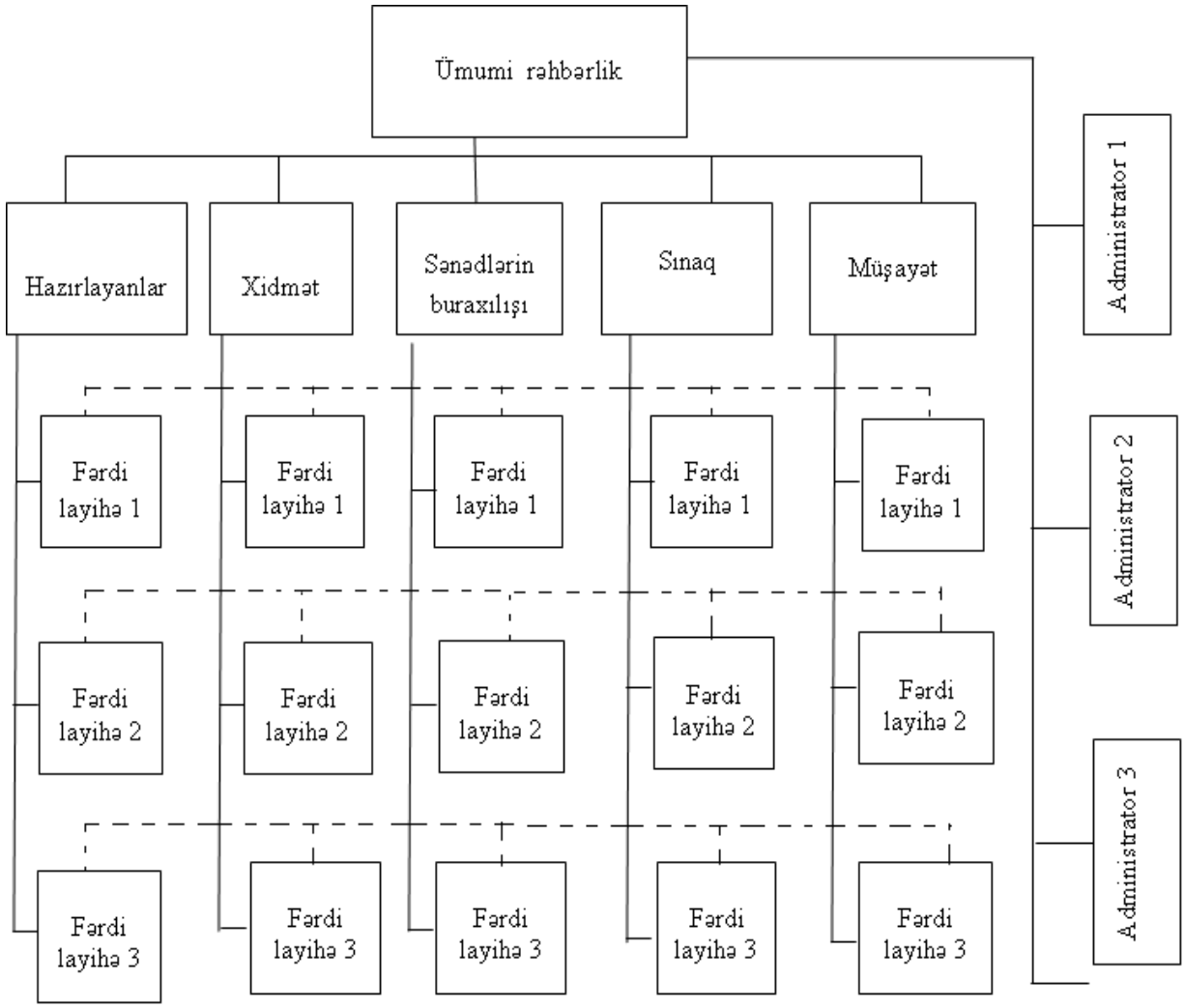
Problem ciddi şəkildə öyrənildikdən və müzakirə olunduqdan sonra məhsul buraxılışının yekun planı işlənib hazırlanır.

Birinci məsələ büdcə bölüşdürüldükdən sonra həll olunur, tələblər haqqında razılaşma adlanan plan formalaşdırılır. Bu sənədin hazırlanmasında yaradıcı ilə yanaşı bir çox funksional bölmələr, o cümlədən sınaq qrupu, sənədləri buraxan qrup, tətbiqlə, müşayətlə, satışla məşğul olan personal da iştirak edir.

Tələblər haqqında razılaşmaların nəzərdən keçirilməsi, təsdiqi və düzəlişlərin aparılması hazırlanmanın planlaşdırılması prosesinin ən vacib mərhələləridir. Bilavasitə tələblər haqqında razılaşmalara əsasən bütün maraqlı tərəflər, məhsulun hazırlanması ilə bağlı nələrin gözlənildiyini müəyyənləşdirməyə çalışırlar.

## 2.3 Proqram məhsulunun hazırlanmasının təşkili mərhələləri

Layihənin idarə olunması dedikdə biz burada proqram məhsuluna, onun funksiya və layihələrinin matris strukturundan istifadə etməklə, qoyulan tələblərіə çatmanın idarə olunmasını başa düşəcəyik. Bu matrisdə hər bir funksiyaya rəhbərlər qrupu təyin olunur. Bu rəhbərlər uyğun finksiyaların ən yaxşı şəkildə yerinə yetirilməsinə məsuliyyət daşıyırlar. Hər bir proqram məhsuluna, öz növbəsində, rəhbərlər qrupu cavabdeh olur ki, onlar fikirlərini yalnız proqram məhsulunun hazırlanmasına yönəltməlidirlər (şək. 2.5).




Şəkil 2.5 Layihənin matrisli idarəolunması sxemi

Layihənin administratoru (rəhbəri) bir – yeganə layihə ilə məşğuldur, belə ki, onun hər bir funksiyası bir neçə fərdi hazırlanmanı əhatə edir. Məmulatın administratoru hər bir funksional qrupun proqram məmulatının hazırlanmasında iştirakını tənzimləyir. Bu tənzimlənmənin uğuru, birinci növbədə, əvvəlcədən müəyyən olunmuş texniki tələblərə və məqsədlərə, həm də mümkün xərclərin sərhəddlərinə riayət olunması ilə tənzimlənir. Adətən məmulatın administratoru öz fəaliyyətini mümkünlük, yəni həyata keçirilə bilmə fazasından başlayaraq qiymətləndirmə fazasının sonuna qədər yerinə yetirir. Administrator – adətən, digər adamların və bölmələrin gördükləri işlərdən xəbərdar olan, hazırlanma şöbəsinin işçisi olur (çox vaxt o, səlahiyyəti çatmadan bütün işlər üçün cavabdehlik daşıyır).

Matris strukturda hər bir yaradıcının iki müdiri vardır. Lakin, bilavasitə matris struktur proqramların hazırlanması kimi əvvəlcədən söylənilə bilməyən proseslərin idarə olunması üçün kifayət qədər effektiv hesab olunur.

Proqram məmulatının hazırlanmasının yaradılma (layihələndirmə) mərhələsində təşkilinə baxaq. Layihələndirmənin əsas məqsədi proqram məmulatına qoyulan tələblərin hazırlanması və analizi hesab olunur. Layihənin dekompozisiyası prosesi tələblər haqqında razılaşmaların tərtibindən başlamaqla spesifikasiyaların hazırlanması və layihənin iki hissəyə - daxili və xarici layihələrə bölünməsi yolu ilə davam etdirilir. Xarici layihə - istifadəçinin görə bildiyi mroqram məhsulunun xarakteristikaları toplusudur. Daxili layihə - istifadəçidən gizlədilən proqram məhsulunun xarakteristikaları toplusudur. Bu ona görə belə edilir ki, istifadəçilər birinci növbədə, onlara aidiyyəti olan proqram məhsulunun xarakteristikalarına kritik yanaşa bilsinlər, lakin məmulatın daxili xarakteristikalarının kritikliynə baxılmır. Çünki, xarici layihə proqram məhsulunun nə etdiyini şərh edir, lakin daxili spesifikasiyalar isə xarici spesifikasiyalara çatmaq üçün məhsulun necə layihələndirildiyini göstərir. Xarici spesifikasiyalar açıq və geniş müzakirə üçün ötürülür, bu zaman istifadəçilərin təkliflərinə üstünlük verilir.

Daxili və xarici layihələrin arasında sərhəd çəkməkdən ötrü dekompozisiya sxemini nizamlayırlar. Dekompozisiya sxemi yaxşı nizamlanmış hesab olunur. Bir

funksiyanın digəri tərəfindən çağırılmasının hər anı, hər hansı abstraktlaşdırma səviyyəsi də daxil olmaqla, qeydə alınarsa, onda belə dekompozisiya sxemi yaxşı nizamlanmış adlanır. Bundan sonra hər bir funksional modula qara yeşik kimi baxılır. Belə funksional modulun xarici əlaqələrini müəyyən etmək mümkün olsa da, lakin onun daxili quruluşu haqqında heç nə demək mümkün deyildir. Qara qutunun xassələri onun kənardan görünən xarakteristikalarının tam şəkildə şərhi ilə müəyyən olunur. Bu şərhə həmçinin şərtlərin şərhi də daxildir. Proqram məhsulunun vahid bir tam kimi şərhinin tərtib olunmasında mühüm rol oynayan modulların atributları xarici layihəni təşkil edir. Modulların bütün digər, tam yaxud da qismən proqram məhsulunun daxilində "gizlədilən" parametrləri birlikdə daxili layihəni təşkil edirlər.

Xarici və daxili layihələr arasındakı dəqiq fərqi nəzərə almaqla, proqram məhsulunun xarici və daxili spesifikasiyalarını tərtib etmək mümkündür. Bu zaman müxtəlif sənədlərdə olan ümumi hissələri nəzərə almamaq, yəni kənarlaşdırmaq zəruridir. Adətən spesifikasiyaların forması ciddi standartlaşdırılmışdır (SADT, PDL və başqaları).

Xarici spesifikasiyalar nisbətən stabil xarakter aldıqdan sonra, hazırlanma funksiyası çərçivəsində onların funksional qruplara göndərişi baş verir ( "layihə" qeydiyyatı ilə). Bütün iradlar texniki komitə adlanan xüsusi orqan tərəfindən cəmlənir və analiz olunduqdan sonra hazırlayıcı qrupa (proqram məgsulunu hazırlayan qrupa) ötürülür. Bütün iradların maksimal şəkildə nəzərə alınması yaradıcı qrupun vəzifəsidir.

Xarici və daxili spesifikasiyalar yaradılan zaman digər funksional qruplar sənədlərin çıxışa hazırlanması, sınaqların planı və digər planların hazırlamaqla məşğul olurlar. Bu sənədlər layihələndirmə fazasının sonunda baxılmaq üçün göndərilir. Layihənin rəhbəri onu öyrənir və təsdiq edir. Layihələndirmə mərhələsi adətən xarici spesifikasiyaların təsdiqi ilə başa çatır.

Proqram məhsulunun hazırlanmasının proqramlaşdırma mərhələsində (fazasında) necə təşkil olunduğunu nəzərdən keçirək. Hazırlanmanın təşkilinin əsas məsələsi, bu funksiyanın reallaşdırılması ilə məşğul olan çoxsaylı işçilərin

qüvvələrinin koordinasiyasından, yəni əlaqələndirilməsindən, həm də müxtəlif funksional qruplar arasında qarşılıqlı əlaqənin təşkilindən ibarətdir. Metod – baxılan müəssisədə qəbul olunan proqramlaşdırma standartlarına riayət etməkdir.

Kodlaşdırmaya proqramlaşdırmanın ilkin mərhələsində başlanılıar. Bu zaman "dalğalı effekt"dən (cədvəl 2) istifadə olunur. Proqramın xarici və daxili spesifikasiyaların tərtibi, kodlaşdırma, sazlanma və onun ayri-ayrı elementlərdən yığılması eyni zamanda proqram mıhsulunun strukturunun müxtəlif səviyyələrində yerinə yetirilir.

Cədvəl 2 – məmulatın modullarının hazırlanmasında dalğa effekti

| | **Modullar** | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **İ** | **J** | **K** | **L** | **M** | **N** |
| Xarici spesifikasiyaların tərtib olunması sona çatmışdır. | x | x | - | - | x | x | x | - | - | - | x | x | x | x |
| Daxili spesifikasiyaların tərtib olunması sona çatmışdır. | x | x | x | x | - | x | x | x | x | - | - | x | x | x |
| Kodlaşdırma sona çatmışdır | x | x | x | - | - | x | x | x | x | - | - | x | - | x |
| Sazlanma bitmişdir | x | x | x | - | - | x | x | x | - | - | - | x | - | x |
| Ayri-ayri elementlərdən yığılma bitmişdirş | x | x | x | - | - | - | x | x | - | - | - | x | - | - |

Məsələn, hər hansı bir modulun proqramlaşdırılması fazasında proqram məhsulunun bütünlükdə xarici spesifikasiyaları artıq təsdiqlənə bilər, lakin daxili spesifikasiyalar hələ sona qədər tərtib olunmamışdır və əksinə, kodlaşdırma, sazlanma və toplanma mərhələlərinin bəzi modulları hazır olsa da, onların bütövlükdə tam proqram məhsulu çərçivəsində hazırlanması hələ sona çatmaya bilər.

Dalğa effektinin ciddi bir çatışmayan cəhətini yaddan çıxarmaq lazım deyildir: xarici spesifikasiyaların təsdiq olunması ilə bağlı yubanma bir çox funksiyaların yerinə yetməsinə çox pis təsir göstərə bilər. Çünki, sınaq spesifikasiyalarını təsdiq etmək olmaz, reklam materiallarını sona çatdırmaq mümkün deyildir və sair.

Hazırlayıcı qrupun fəaliyyəti kodlaşdırma, sazlama və toplanma ilə yanaşı həm də proqramlaşdırma fazasının sonunda hazır məhsulun nümayişindən və hazırlayıcı qrupla digər funksional qruplarla qarşılıqlı əlaqəsinin təşkilindən ibarətdir. Hazırlayıcı qrupda hər şeydən əvvəl, planda nəzərdə tutulan müddətlərin düzgün əsaslandırılmasına və proqram məhsulunun şərhinə aid olan dəstək qrupunun verdiyi təkliflərin düzgünlüyünə inam omalıdır. Reklam materialları vasitəsi ilə istifadəçilərə çatdırılan proqram məhsulunun xarakteristikalarının şərhindəki ixtiyari səhvlər qeyri müəyyənlik, yaxdu, ən pisi odur ki, sifariş yerinə yetirilmədiyi üçün cərimə halı yarada bilər.

Proqramlaşdırma fazasında sənədləri çıxışa hazırlayan qrup (buraxılış qrupu) bir neçə variantda sorğu materialları təqdim edir. Proqramlaşdırma fazasının ortalarında sınaq qrupu tərəfindən aparılacaq sınaqların qrafikin təqdim olunur. Yaradıcı qrup sınağın sənədlərinin bütün variantlarını və spesifikasiyalarını çox diqqətlə nəzərdən keçirirlər ki, heç bir səhv olmasın. Əgər yaradıcı qrup vaxtında korrekt, yəni səhvsiz xarici spesifikasiyalar hazırlamışsa, onda onların yoxlanılması böyük çətinliklər törətmir və çox vaxt aparmır. Lakin, əgər xarici spesifikasiyaların hər hansı vəziyyətləri buraxılıbsa (unudulubsa), yaxud kifayət qədər tam şərh olmayıbsa, onda onların yoxlanılması nəinki çox vaxt aparır, həm də böyük çətinliklər törədir. Belə olan halda xarici spesifikasiyaları dəyişmək lazım gəlir, bu isə layihələndirmənin təqvim planı ilə əlaqədar vaxt ehtiyatını yoxa endirir.

Dəstək qrupu təsdiqdən sonra reklam materiallarını hazırlayır və baxılmaq üçün göndərir. Yaradıcı qrup bu materialları, anlaşılmamazlıqlardan doğan və uyğun maliyyə sanksiyalarının yaratdığı texniki səhvlərə yol verməmək üçün, analiz edir. Adətən proqramlaşdırma fazasında proqram məhsulunu onun işi

zamanı, yəni fəaliyyətdə olan zaman nümayiş etdirmək daha məqsədə uyğundur, çünki, bu zaman proqram məhsulunun ən kritik istismar xarakteristikalarının uyğun tələblərə necə cavab verdiyini görmək, yaxud layihənin nə qədər "uzağa" getdiyini müəyyən etmək mümkündür. Yaradıcı qrup bu mərhələnin tez sona çatmasına çalışır ki, proqram məhsulunun nümayişi zamanı tutulan iradları, tənqidləri nəzərə ala bilsin.

Layihənin sona yetməsini şərtləndirən səbəblərdən asılı olmayaraq yaradıcı qrup, gələcək layihələrin yaradıcılıarı üçün təcrübə mənbəyi ola biləcək hesabat açıqlayır. Son hesabat layihənin bütün iştirakçıları tərəfindən birlikdə hazırlanır və minimum aşağıdakı informasiyaları özündə əks etdirir:

- layihənin hazırlanması zamanı rast gəlinən ən böyük çətinliklərin aradan qaldırılması təcrübəsi;
- sonrakı, yəni sonra yaradılacaq layihələr üzrə məsləhətlər (müxtəlif variantlar da daxil olmaqla);
- mərhələlərin yerinə yetirilməsinin plan və faktiki müddətləri haqqında məlumat;
- planda nəzərdə tutulan və faktiki xərclər haqqında məlumat;
- planda nəzərdə tutulan və faktiki əmək sərfi haqqında məlumat;
- planda nəzərdə tutulan və faktiki istifadə olunan maşın ehtiyatları haqqında məlumat;
- avadanlıqlarla iş zamanı çətinliklərin xronologiyası və gələcəkdə çətinliklərin aradan qaldırılması üçün məsləhətlər;
- funksional bölmələrin qarşılıqlı fəaliyyəti ilə bağlı olan çətinliklərin baş vermə xronologiyası;
- Qeyri müəyyənlik şəraitində planlaşdırma üzrə məsləhətlər;
- ən əhəmiyyətli hadisələrin xronoloji yazısı.

Əgər proqram məhsulunun hazırlanması tam başa çatmışdırsa, onda yekun hesabat müşayət spesifikasiyalarına daxil edilir.

# FƏSİL III Proqram vasitələrinin yaradilmasinin dialoqlu instrumental proqramlaşdirma sistemi

## 3.1 Dialoqlu sistemlərin layihələndirilməsi vasitələri

Dialoqlu instrumental proqramlaşdirma sisteminin (**DİPS**) hal-hazırda fəaliyyətdə olan bir neçə geniş yayılmış instrumental-texnoloji komplekslər sırasında yerini qiymətləndirməyə çalışaq. Bu zaman nəzərə alaq ki, dialoqlu mayak sistemlər (**DM-sistemlər**) unikal dialoqlu proqram vasitələridirlər və onların analoqları əsasən əllə yaradılır, yəni proqramçılar özləri bu proqram vasitələrini yaradırlar. Bir çoz instrumental komplekslərdə yalnız dövr saxlamayan tərbiq sahəsinin modelinə və A hesablama rejimli, birvariantlı ünsiyyət modelinə malik problem-oriyentasiyalı sistemlər avtomatlaşdırmaya məruz qalırlar. Belə problem-oriyentasiyalı sistemlərin nisbətən sadə olması bu sistemlərdə, dialoqlu mayak sistemlərdə tətbiq olunan hesablama prosesinin dinamik rejimindən fərqli olaraq hesablama prosesinin statik rejimdə həyata keçirilməsi ilə bağlıdır. İribloklu proqramlaşdırma şəraitində modulların kompleksləşdirilməsi (birləşdirilməsi) problemini nəzərdən keçirək.

İribloklu proqramlaşdırma ideyası (başqa sözlə desək, zəruri funksiya və prosedurların yaradılması üçün iriləşdirilmiş “bloklar”dan istifadə etmək ideyası) keçən əsrin 50-ci illərində rus akademiki L. V. Kantoroviç tərəfindən irəli sürülmüşdür. Sonralar bu ideya bir çox proqramlaşdırma və spesifikasiyalar dillərində, məsələ həlledicilərində, proqram generatorlarında, instrumental-texnoloji komplekslərdə və problem-oriyentasiyalı sistemlərdə həyata həyata keçirilmişdir. Bu ideyanın əsas aspektlərindən biri də hazır proqram vasitələrinin fondlaşdırılması, yəni uyğun ölkənin alqoritm və proqramlar fondlarında saxlanılması və müstəqil şəkildə istifadə oluna bilməsidir.

Modullararası interfeysin ən mütərəqqi mexanizmləri problem-oriyentasiyalı sistemlərdə və instrumental-texnoloji komplekslərdə sınaqdan keçirilmişdir. Bu əsasən verilmiş proqram vasitələrinin yaradıcılarının həll etdilkəri məsələlərin

ümumi olması və proqram məhsulunun layihələndirilməsi prosesinin kifayət qədər tam şəkildə avtomatlaşdırılmasına nail olmaq məqsədi ilə əlaqədardır. Məsələn, mobil (köçürülə bilən) problem-oriyentasiyalı sistemlərin yaradılması tendensiyası müşahidə olunur; onların maşından asılı və maşından asılı olmayan komponentləri bir birlərindən fərqləndirilir. Ən çox populyarlıq tətbiqi proqramlar paketləri və avtomatlaşdırılmış tətbiqi proqramlar paketləri yaradan sistemlər (tətbiqi proqramlar paketlərinin generatorları) qazanmışlar. Onlar inkişaf etmiş və geniş tərkibli kitabxanalarda, yəni xarici yaddaşda qorunan fayllarda saxlanılan çoxsaylı (dəfələrlə) istifadə oluna bilən modullara əsaslanaraq yaradılırlar.

Bir çox dünya ölkələrində tətbiq olunan məşhur **PRİZ** instrumental kompleksi hal-hazırda da inkişaf etdirilməkdədir. Bu kompleksin tərkibi yeni modullar əlavə olunmaqla zənginləşdirilir və imkanları, həm də tətbiq sahələri durmadan genişləndirilir. **PRİZ** instrumental kompleksinin əsasını təşkil edən ideyanın təsiri altında bir çox instrumental komplekslər yaradılmışdır (məsələn, **SPORA** instrumental kompleksi). **PRİZ** generatorunun instrumentariləri yaradılan zaman qarşıya aşağıdakı vasitələrin yaradılması məqsədi qoyulmuşdur:

- hesablama modelləri üzrə avtomatik sintez metodları əsasında hesablama prosesinin planlaşdırılması;
- generasiya olunan tətbiqi proqramlar paketlərinin dil səviyyəsində inteqrasiyasına zəmanət verən, **PRİZ** generatorunun **UTOPİST** giriş dilinin genişləndirilməsi kimi mühəndis-texniki giriş dillərinin layihələndirilməsi.;
- relyasiyalı verilənlər bazasının aparılması, vahid informasiya bazası əsasında inteqrasiya olunan tətbiqi proqramlar paketlərinin imkanlarının genişləndirilməsi.

**PRİZ** instrumental kompleksi inteqrasiya olunmuş proqramlaşdırma sistemi (**İOPS**) ilə uyuşandır. **İOPS** sistemi bazasında **MEMO** generatorlar ailəsi yaradılmışdır. Bu generatorun dörd variantı reallaşdırılmış və yayılmışdır: paket rejimində hesablama aparan **İOPS-2**; bir istifadəçili interaktiv rejimdə işləyən

**İOPSİ**; çox istifadəçili interaktiv rejimdə işləyən **İOPS-KP**; genişləndirilmiş giriş dilləri sinfi və paketli işləmə rejimli **İOPS-KP**.

Dialoqlu instrumental proqramlaşdirma sistemində olduğu kimi, **MEMO** generatorlar ailəsi də hesablama prosesinin təşkili stabil sxem üzrə təşkil olunan zaman məqsədəuyğundur. Operativ verilənlər bazası və dialoq rejimin geniş istifadə olunması **MEMO** və **DİPS** komplekslərini bir birinə yaxınlaşdırır.

Eyni tətbiq sahəsində istifadə olunan paketlər versiyalarının yaradılmasına istiqamətlənmiş tətbiqi proqramlar paketlərinin generasiyası texnologiyası **SİGMA-DELTA** generatorunda realizə olunmuşdur. Bu generatorun vasitəsi ilə verilənlərin statik işlənməsi və analizi məsələlərinin həlli üçün **D-sistemlər** yaradılır. Bu texnologiya, paketlərin və əməliyyat mühitinin birlikdə təkmilləşdirilməsini təmin edən, inkişaf etmiş modulluluq prinsipinə əsaslanır.

**VİLNÜS** generatoru vahid konseptuallıq əsasında dialoqlu tətbiqi proqramlar paketlərinin yaradılması istiqamətində inkişaf edir. Bu istiqamət layihələndirmə dillərini, paketlərin generatorlarını, instrumental və servis vasitələrini əhatə edir. İnformasiya xidmətinin inkişafı və unifikasiyası, tətbiqi proqramlar paketləri generatorlarında tətbiqi proqramlar paketləri və verilənlər bazasının idarə olunması sistemləri arasında informasiya mübadiləsini təşkil edən komponentin oması müşahidə olunur. Bəzi tətbiqi proqramlar paketlərinin generatorlarında informasiya yaxud sual-cavab sistemləri, komponentlər arasında informasiya əlaqələrini bərpa etmək, və həm də istifadəçilərin yaradılan tətbiqi proqramlar paketləri ilə qarşılıqlı fəaliyyətini təşkil etmək üçün istifadə olunur. İnkişaf etmiş tətbiqi proqramlar paketlərinin generatorları yaratma və verilənlər bankları ilə qarşılıqlı fəaliyyət vasitələrinə malikdirlər. Məsələn, **PRİZ** kompleksinin **DABU** VBİS verilənlər bamkları yaratmaq üçün istifadı oluna bilər. **VİLNÜS** generatorunda özünün xüsusi VBİS qurulmuşdur.

Bir çox tətbiqi proqramlar paketlərinin generatorlarında tətbiq sahəsi modelinin yaradılması altsistemi vardır. Bu alt sistemin yaradılması tətbiqi proqramlar paketləri yaradılarkən ən çox zəhmət və vaxt tələb edən prosesdir. Tətbiq sahəsi modelinin yaradıcıları süni intellekt, xüsusilə də dillər və

proqramların spesifikasiyalarının müəyyən olunması sahələrində əldə olunmuş naliyyətlərdən istifadə edirlər. Bundan başqa, tətbiq sahəsi modelində tətbiq sahəsinə nisbətən invariant olan komponent ayrılır.

Beləliklə, tətbiqi proqramlar paketlərinin generatorlarının yaradılması sistem və tətbiqi proqramlaşdırmanın ayrı-ayrı sahələrində yaradılan və tətbiq olunan, indiyə qədər bir-biri ilə zəif qarşılıqlı fəaliyyətdə olmaqla paralel inkişaf edən, müxtəlif vasitələrin və metodların bir-birinə yaxınlaşmasına və birlikdə istifadə olunmasına gətirib çıxardı. Əgər nəzərə alsaq ki, tətbiqi proqramlar paketlərində hesablama prosesinin planlaşdırılmasının təşkili zamanı hal-hazırda süni intellekt yaradılan zaman əldə olunan naliyyətlərdən istifadə olunur, onda belə qərara gəlmək olar ki, problem-oriyentasiyalı instrumental proqramlaşdlrma sistemlərinin yaradılması problematikası sistem və nəzəri proqramlaşdırmanın əsas istiqamətlərindən birini təşkil edir.

Həll olunacaq məsələlər üçün proqram modullarının ayrılmasına zəmin olan tətbiq sahəsinin modul analizinin aparılması metodlarını nəzərə almadan, tətbiqi proqramlar paketlərinin generasiyası prosesini nəzərdən keçirək.

Generator tətbiqi proqramlar paketini iki üsuldan biri ilə yaradır. Üsullardan birində generatorun özünügenişləndirmə vasitələrindən istifadə olunur. Nəticədə alınan paket generatorun özünün sistem komponentlərindən istifadə edir və yalnız bu generator vasitəsi ilə yaradılan əməliyyat mühitində fəaliyyət göstərə bilir. Belə tətbiqi proqramlar paketləri qurulmuş tətbiqi proqramlar paketi adlanırlar. Qurulmuş tətbiqi proqramlar paketinin generasiyası uyğun tapşırığın mətninin işlənməsindən və hazır modullardan paketin funksional modullarının formalaşdırılmasından ibarət olur.

**PACKAGE** generatorunda tətbiqi proqramlar paketini generasiya etmək üçün tapşırıq **KONSTRUKTOR** giriş dilində şərh olunur. Paketin şərhi dedikdə, onun komponentlərinin, əsasən də baza dilinin proqram vahidləri hesab olunan altproqram və funksiyaların, prosedur və prosedur-funksiyaların şərhi nəzərdə tutulur. Hər bir şərh, şərhin mətnindən və başlıqdan ibarət olur. Başlıqda ad, formal parametrlərin siyahısı və onların spesifikasiyaları göstərilir. Makrotəyinlər və

makroşərhlər də paketin komponentləri hesab olunurlar. Paketin şərhinin emalı (işlənməsi) **KONSTRUKTOR** altsistemi vasitəsi ilə yerinə yetirilir. Bu altsistem paketin şərhinin prosessoröncəsi işlənməsini yerinə yetirir, paketin emal olunmuş komponentlərini xarici yaddaşda yerləşdirir, paketin bütün komponentləri haqqında sorğu informasiyalarını, onların xarakteristikaları və xarici yaddaşda yerləşdirilməsi haqqında informasiya da daxil olmaqla, toplayır. **PACKAGE** sistemi vasitəsi ilə generasiya olunan paketləri birlikdə istifadə etmək olar, bu şərtlə ki, onların informasiya, proqram və funksional uyuşanlığı təmin olunsun.

**PRİZ** kompleksi vasitəsi ilə yaradılan hər bir paket UTOPİST dilinin genişlənməsi olan yüksək səviyyəli giriş dili ilə təchiz olunur, paketin funksional hissəsi (paketin mətni) modullar kitabxanasından və paketin tətbiq sahəsindən olan anlayışları şərh edən semantik modellər kitabxanasından ibarətdir. **PRİZ** kompleksi tətbiq sahəsinin modul analizi, proqramlaşdırma və paketin funksional hissəsinin sazlanması mərhələlərini dəstəkləyir. Bu kompleks mühitində yaradılan paketlərin sistem hissəsində (idarəedici proqramda) onun, yəni **PRİZ**in sistem proqramlarından istifadə olunur. Paketlərdən birlikdə istifadə etmək üçün onların proqram, informasiya və funksional uyuşanlığını təmin etmək lazımdır. Paketlərin proqram uyuşanlığı modulların adlarının, müxtəlif paketlərdə ümumi sahələrin üst-üstə düşməməsi ilə təmin olunur. Birlikdə tətbiq olunan paketlərin tətbiq sahələrinin modelləri eyni bir kitabxaanaya yerləşdirilməlidir.

Qurulmuş tətbiqi proqramlar paketləri POLE generatoru vasitəsi ilə, dəyişən strukturlu universal uyğunlaşdırılmış modullardan, həll olunacaq məsələlərin alqoritmlərinin giriş dilində şərhi vasitəsi ilə alınır.

Avtonom sistem hissəsinə malik paket qurulmuş paketə nisbətən müəyyən üstünlüklərə malikdir; o yalnız onun təyin olunduğu funksiyaları yerinə yetirir və bu paket konkret tətbiq sahəsinin mütəxəsisləri üçün yaradılmışdır. Yaradılan tətbiqi proqramlar paketlərinin funksional hissəsi kifayət qədər araşdırılmış və müsbət nəticələr əldə olunmuşdur.

Tətbiqi proqramlar paketlərinin generatorlarının yaradılması təcrübəsini analiz etməklə belə nəticəyə gəlmək olur ki, problem-oriyentasiyalı instrumental

proqramlaşdırma sistemlərinin effektiv şəkildə fəaliyyəti, generatordan asılı olmayaraq fəaliyyət göstərən avtonom sistem hissənin reallaşdırılması yolu ilə əldə olunur. Öz növbəsində, avtonom sistem hissəyə malik problem-oriyentasiyalı instrumental proqramlaşdırma sistemləri içərisində çox böyük maraq doğuran o paketlərdir ki, onlarda aşağıdakılar tətbiq olunur:

- sistem hissənin xarici modullaşması;
- inteqrallaşdırılmış informasiya bazasının avtomatik aparılması;
- tətbiq sahəsinin alqoritmləşdirilən obyektlərinin onların formal modellərinə inikası.

Modullu interfeys metodları bilavasitə bu istiqamətdə inkişaf etdirilir və öz əksini aşkar şəkildə spesifikasiyalar dillərində, proqramların şərh olunmasında tapır. Bu məqsədlərə çatmağa həm də **MODULA-2**, **ADA**, **KLU** və başqa yeni perspektivli prosedur dillərin yaradılması zamanı çalışmışlar.

## 3.2 Dialoqlu instrumental proqramlaşdirma sisteminin generasiya vasitələrinin tərkibi

PL – mayak sistemlər dialoqlu instrumental proqramlaşdırma sistemi vasitəsi ilə generasiya olunurlar. İkisəviyyəli tətbiq sahəsinin modeli PL – mayak sistemlərində (PL-MS sistemlərində) adətən yüklü qraf formasında verilə bilirlər. İlk səviyyədə həlli tələb olunan məsələnin alt məsələlərə bölünməsinin sxemi Z-şəbəkədə qeyd olunur, başqa sözlə desək paketlərin inteqrasiyası sxemi alınır. Sonrakı səviyyədə və mayak qrafında paketdə məsələnin həllinin bütün mümkün istiqamətləri çoxluğu verilir.

Dialoqlu instrumental proqramlaşdirma sistemində tətbiqi proqramlar paketləri birləşdirilərkən Z-şəbəkə qurulur:

- hər hansı bir paketə uyğun tətbiq sahəsinin altmodeli Z-şəbəkənin müəyyən təpə nöqtəsinə qarşı qoyulur;

- Z-şəbəkənin təpə nöqtələri, yəni düyünləri mümkündür ki, yüklənmiş tillərlə əlaqələndirilsinlər, belə ki, bu şəbəkənin düyünlərinin daxilolma və çıxma yarımdərəcələri ixtiyari ola bilərlər;
- paketlərin qoşulmasının hər hansı ardıcıllığı Z-şəbəkə üzərində mayak adlandırılır.

Mayak qrafı ilə Z-şəbəkə arasında təkcə bir fərq vardır: mayak qrafının bir kökü və yeganə son təpə noqtəsi olduğu halda, Z-şəbəkənin son və kök təpə nöqtələrinin sayı elə onun bütün təpə nöqtələrinin sayı qədərdir. Mayak sistemlərdə paket və paketüstü səviyyələrdə məsələlərin addımlarla həlli konsepsiyası bir birindən fərqlənir.

PL-MS sisteminin giriş dilləri cədvəl formasına yaxın bir formadadırlar. Məsələlərin verilmiş şərtlərində ( məsələlərin verilmiş şərtlərinin maketində) məsələnin parametrlərinin yerləşmə vəziyyəti sətirlər ardıcılığında qeyd olunur.

DİPS kompleksi dörd altsistemdən ibarətdir. Bu altsistemlərdən üçü PL-MS sisteminin generasiyası zamanı istifadə olunur. Dördüncü altsistem isə mayak sistemin nüvəsi hesab olunur və monitor adlandırılır. Monitor dialoq rejimində addımlarla hesablamanı təşkil edir. Baza vasitələri, qrafların translyatoru və Z-şəbəkə üzrə paketlərin inteqrasiyası pasportlarının formalaşdırılması proseduruna əsasən informasiya mühitinin sistem seqmentləri və MS-sistemin işə salınması proseduru yaradılır.

PL-MS sisteminin monitoru yalnız MS sistemin xüsusi fəaliyyət dövründə parametrləşdirilir. Bunun üçün informasiya mühitinin sistem hissəsinin uyğun fraqmentləri hesablanır. Hər bir həll olunan məsələyə görə əlavə olaraqistifadəçi hissənin seqmenti yaradılır. İnformasiya mühiti istifadəçinin birbaşa təsirindən qorunur (mühafizə edilir) və bununla da PL-MS sisteminin tamlığı əldə olunur.

PL-MS sistemini generasiya etmək elə informasiya mühitinin sistem fraqmentlərini almaq deməkdir. Belə ki, fraqmentlərin alınması qaydası DİPS-1 kompleksi vasitəsi ilə yönləndirilir və düzəlişlər edilir.

DİPS-1 instrumental proqramlaşdırma sistemində paketlərdə, mayak qrafının təpə nöqtələri ilə əks olunan hesablama addımları arasında verilənlərin ötürülməsi

xarici yaddaşda saxlanılan informasiya mühitinin xüsusi fraqmentləri vasitəsi ilə həyata keçirilir. Hesablama addımları soproqram şəklində formalaşdırılır, yəni yerinə yetməyə hazır olan modulların mətni xüsusi pasportlarla təmin olunur, onların məzmunu əsasən mayak qrafından götürülür və hesablama addımı yerinə yetriliən zaman informasiya mühitinin vəziyyəti nəzərə alınır. Belə ki, modullar özləri deyil, onların yalnız giriş və çıxışları pasportlaşdırılır. Hesablama addımı, elə tətbiqi proqramlar paketinin özü kimi bir girişə və bir neçə çıxışa malik olur, lakin yalnız çıxışlardan biri hesablamanın normal bitməsinə, yəni qəzasız sona çatmasına uyğun gəlir və idarəni mayak sistemin monitorunun çıxışına ötürür. Hesablama addımının qəzalı sona çatması aşağıdakı səbəblərdən yaranır:

- hesablama prosesi istifadəçi işə qarışdıqdan sonra iş davam etdirilərkən mayak qrafının üzərində dəyişiklik edilə bilən tilləri üzrə çıxışlar;
- hesablama addımı bitərkən, məsələnin tam qoyulmaması yaxud da səhv qoyulmas səbəbindən, yaxud da mayak qrafındakı səhvə görə artıq marşrutun qurulması prosesinin mümkünsüzlüyü aşkar olunur.

Əgər mayak qrafında paketin hesablama addımlarının idarə olunması axınları pasportlaşdırılırsa, onda onların informasiya uyuşanlığı, paketin spesifikasiyalarında parametrlərin xülasəsi adlanan elementində əks olunur. Belə qəbul olunub ki, məsələlərin həlli zamanı hesablama addımları informasiya mühitinin istifadəçi hissəsinin xüsusi fraqmentləri vasitəsi və parametrlərin qiymətlərinin aralıq faylının yadda saxlanılması yolu ilə "ixrac/idxal" olunur. Paketin kök addımı üçün məsələnin şərti giriş aralıq faylı, çıxış üçün isə ixrac olunan fayl hesab olunur. Uyğun olaraq məsələnin həllinin nəticəsi paketun kök addımından "ixrac" olunan aralıq fayldır.

Paketin spesifikasiyası dedikdə aralıq fayllarda parametrlərin şablonlarını və lokasiyasını əks etdirən paketlərin bütün parametrlərinin xülasəsi başa düşülür. Tam və tam olmayan (xülasənin hissəsi) xülasələri bir birindən fərqləndirirlər. Tam olmayan xülasə parametrlərin məsələnin şərtinə daxil olması haqqında məlumatları saxlayır, tam xülasə isə parametrlərin bütün aralıq fayllara hissə-hissə daxil olmasıdır.

Hesablama addımlarının və paketin bütün parametrlərinin tam xülasələrinin şərhi adlar **tek**ində saxlanılır. Adlar tekində obyektin açarı birbaşa müraciət oluna bilən fayı olmaqla, bu obyektin sistem adı olur.

Baza vasitələrin tərkibinə daxil olan paketlərin parametrləri haqqında məlumat teklərə mayak sistemin yaradıcısı tərəfindən COR-GENER proseduru vasitəsi ilə yazılır. Addımların yükləmə modulları DİPS-1 instrumental proqramlaşdırma sistemin baza vasitələrinin tərkibində olan SYS-HAG və SHA-GENER prosedurları vasitəsi ilə yaradılır.

Baza vasitələri on iki prosedur olmaqla kataloqlaşdırılmış prosedurlar şəklində tərtib olunmuşdur. Onlar mayak sistemin informasiya mühitinin sistem hissəsinin seqmentlərinin yaradılması üçün təyin olunmuşlar. CR-FİLE, COR-GENER, SİO-GENER və PROT-GEN prosedurları qraflar translyatoru vasitəsi ilə istifadə olunur. Bu prosedurların ilkin verilənlərini əllə kodlaşdırdıqda informasiya mühitinin bütün zəruri seqmentlərini qraflar translyatoru olmadan da almaq olar.

Generasiya mərhələsində istifadə olunma ardıcıllığı ilə bütün baza vasitələrini sadalayaq:

- CR-FİLE proseduru adlar tekinin strukturunu formalaşdırır və hesablama addımlarının sistem adlarını, paketlərin parametrlərini və aralıq faylları təyin erdir;
- SYS-HAG proseduru paketlərin hesablama addımları üçün yükləmə modullarının alınmasından ötrü tapşırığın mətnini formalaşdırır;
- SHA-GENER proseduru paketlərdə hesablama addımları kitabxanasını formalaşdırır;
- CR-FAN proseduru paketin parametrlərinin tam olmayan xülasəsini hazırlayır;
- COR-GED proseduru paketin parametrlərinin tam xülasəsini hazırlayır;
- CO-RAN proseduru parametrlərin tam xülasələrini sistem təqdimatın çevirir;
- COR-GENER proseduru paketlərin adlar tekini formalaşdırır;

- SİO-GENER proseduru addımlarla hesablamanın planlaşdırılması üçün mayak qrafını kodlaşdırır;
- PROT-GEN proseduru paketlərin diaqnostikalar cədvəllərini formalaşdırır. Bu cədvəllərə paketlərdə məsələlərin həlli prosesində displeyin ekrana verilən məlumatların mətnləri formalaşdırılır;
- DİAG-NOZ proseduru paketlərin məlumatları ilə səhvlərin baş verməsi səbəbləri arasında uyğunluq yaradır;
- RİS-GRAF proseduru paketlərin mayak qraflarının şəkillərini çapa vermək üçün yaradılır;
- PECH-TEKA proseduru paketlərin teklərinin adlarını çap edir.

Tətbiqi proqramlar paketlərinin inteqrasiyasının pasportlarını yaradan altsistem informasiya mühitinin uyğun seqmentlərini yaratmaqla yanaşı, həm də sistem hissənin digər seqmentlərinin də olduğunu yoxlayır, informasiya mühitinin bəzi istifadəçi seqmentinin adlarını müəyyən edir və mayak sistemin işə salınması prosedurunu yaradır.

## 3.3 Modullararasi interfeysin inkişaf perspektivləri və pl-ms sisteminin layihələndirilməsi

PL-MS mayak sistemləri üçün məsələlərin addımlarla həlli zamanı əsas obyektlər məsələlər, paketlər, paketlərin parametrləri, hesablama addımları, paketlərin mayak qrafları və paketlərin inteqrasiyası sxemi hesab olunurlar.

PL-MS sisteminin informasiya mühitinin istifadəçi hissəsi xüsusi alt sistemin – monitorun administratorunun idarəsi altında doldurulur və modifikasiya olunur.

Məsələlərin şərtlərinin istifadəçilər tərəfindən modifikasiyasına PL-MS sistemi yalnız müəyyən qədər nəzarət edə bilir. Çünki bu müəllif, yəni istifadəçi modifikasiyaları (baxılan halda PL-MS sistemi ilə nəzarət olunmayan) məsələnin həlli marşrutunun seçilməsinə təsir göstərir, onlar son nəticədə sistem vasitəsi ilə testləşdirilirlər.

İnformasiya mühitinin tamlığına nəzarət münasibətlər modeli (MO) üçün PL-MS sistemi ilə əlaqədə başlıca tələb hesab olunur. PL-MS sistemi bu mühitdə fəaliyyət göstərir və istifadəçilər kollektivinə məsələlərin həllinin nəticələrini bir-birlərinə göndərməyə imkan verir. PL-MS sistemi üçün, Z-şəbəkə heç olmasa yeganə ümumi paketi belə birləşdirirlərsə, informasiya bazasının sistem və istifadəçi hissələrini birləşdirmək daha məqsədəuyğundur. Onda bir PL-MS sistemi vasitəsi ilə həll olunmuş və A<P> , həm də LU<P> fraqmentlərində yerləşdirilmiş məsələnin nəticələri digər PL-MS sistemi tərəfindən istifadə oluna bilər. Təbiidir ki, PL-MS sistemləri arasında verilənlər mübadiləsinə yalnız paketüstü səviyyədə icazə verilir və paketdaxili səviyyədə bir paket vasitəsi ilə məsələnin həlli sona çatana qədər qadağan edilir.

İnformasiya mühitinin sistem hissəsinin tamlığı əsası PL-MS sisteminin layihələndirməsi mərhələsində balanslaşdırılmış baza prosedurları ilə birlikdə qoyulur. Sistem hissənin tamlığı həm də PL-MS sisteminin ayrı-ayrı komponentləri haqqında informasiyaların iterativ və addımlarla, yəni məsələnin həlli prosesində toplanmasını təmin edir. Buraya paket, paketin parametrləri, məsələlərin şərti, mayak qrafları və digər lazımlı informasiyalar daxildirlər. Bu zaman translyatorlar qrafı vasitəsi ilə emal olunan mayak qrafının şərhinə əsasən dəqiqləşdirmə və əlavə məlumatların toplanmasının dialoq xarakterli olması mühüm rol oynayır.

Mayak sistemin yaradıcısının informasiya mühitinin formalaşdırılması və məlumatların toplanması üçün apardığı sorğuların ardıcıllığı məlumdur. Sistemin yaradıcının qraflar translyatoru tərəfindən yaradılan sistem fraqmentlərinin gələcəkdə doldurulması üçün, yəni bu fraqmentlərə iş prosesində əmələ gələn yaxud kənardan daxil olan məlumatların doldurulması üçün, istifadə etməli olduğu baza prosedurların xarakteristikalarına baxaq.

Tətbiqi proqramlar paketlərinin modullaşdırılması prosesinin (modulyarizasiya prosesinin) asanlaşdırılması məqsədi ilə müəyyən qabaqlayıcı tədbirlər görülür. Modullaşdırılması prosesinin nəticəsi olaraq “paketin addımları kitabxanası” fraqmenti alınır. Hər bir paketin mayak qrafına əsasən SYS-HAG

proseduru vasitəsi ilə JCL dilində ***k*** addımdan ibarət tapşırıq yaradılır, burada ***k*** – hesablama addımlarının sayıdır, başqa sözlə desək, paketin funksional modulları ilə reallaşdırılan mayak qrafının təpə nöqtələrinin sayıdır. SHA-GENER proseduru ***k*** addımlarının hər birində çağırılır və bu modul uyğun hesablama addımının yüklənmə modulunu formalaşdırır.

PL-MS sisteminin layihələndirilməsi zamanı yaranan çətin proseslərdən biri də paketlərin parametrlərinin şərh olunması (annotasiya verilməsi) prosesidir, başqa sözlə təsvirlərin maketində (displeyin ekranında) həll olunan məsələlərin və aralıq faylların $P_1$, ... , $P_m$ parametrlərinin lokasiyasıdır. Maket dedikdə səksən simvoldan ibarət sətirlər ardıcıllığı başa düşülür. Bu sətirlər proqramlaşdırma dilinin tələblərinə uyğun olaraq yaradılırlar və həll olunacaq məsələnin parametrlərinin qiymətlərindən ibarət olurlar.Tapşırığın bütün parametrlərinin hamısı paketlərin parametrləri deyildir, onların maketə daxil edilməsi üçün PL-MS sisteminin xüsusi komponenti yaradılmışdır.

Dialoqlu instrumental proqramlaşdirma sistemində bütün parametrlərin şərhləri (annotasiyaları) tam və qismən kompüterin idarəsi altında yaradılır. CR-FAN baza proseduru vasitəsi ilə PL-MS mayak sistemin yaradıcısı ilə sorğu aparılır və sistemin köməyi ilə yaradılan və displeyin ekranına verilən paketin teklərin adı cədvəlində uyğun mövqelər doldurulur. Bu zaman $P_1$, ... , $P_m$ parametrlərinin hər birinin məsələnin həlli maketinə daxil olması göstərilir. Bundan başqa, hər bir parametrin aralıq fayllar ardıcıllığına daxil olması sayı göstərilir.

Cədvəlin bütün mövqeləri doldurulduqdan sonra annotasiyalar, yəni çərhlər, qismən düzəldilir və bu cədvəllər öz növbəsində COR-GED prosedurunda PL-MS sistemi tərəfindən digər cədvəllərin tərtib olunması üçün ilkin informasiya rolunu oynayır və hər bir parametrin aralıq fayllar ardıcıllığına daxil olması sayı göstərilir.

Aydındır ki, PL-MS mayak sisteminin yaradıcısı bir seans ərzində doldurulan cədvəllərin sayını idarə edə bilər, həm də ki, real PL-MS onlarla parametrə malikdir və onların bir çoxu parametr-massivlərdir. Bu səbəbdən də parametrlərin addımlarla, hissə-hissə annotasiyasına CO-RAN proseduru nəzarət edir.

PL-MS mayak sistemi layihələndirilərkən sistemin yaradıcısı izahedici funksiyaları təmin etmək üçün bütün diaqnostik məlumatların mətnlərini əvvəlcədən DİAG-NOZ proseduruna yazır. Paketdə addımlarla hesablama zamanı sistemin istifadəçisini məsələnin həlli gedişi qane etmədikdə o, məsələnin həlli gedişini dayandıra bilir və işin dayandırılma səbəbi DİAG-NOZ proseduru vasitəsi ilə əks olunur. Sistemin yaradıcısının bütün cəhdləri DİPS instrumental kompleksi vasitəsi ilə istiqamətlənir və ona nəzarət olunur. Kəsilmələrin tutarlı səbəbləri əsas götürülür, mayak qrafının şərhində istifadə olunan sadə əməliyyatların mətnləri, məsələnin həlli gedişinə elə də mənfi təsir göstərmədiklərindən, nəzərə alınmır. Bunun üçün elektron cədvəl formalaşdırılır. Bu cədvəldə mayak qrafının yükündə dəyişiklik edilə bilən tillərin çıxdığı təpə nöqtələrinin adları ilə yanaşı, xüsusi qeyd olunmuş sahə də vardır ki, mayak sistemin yaradıcısı kəsilmənin səbəbini bu sahəyə qeyd etsin. Qeyd edək ki, yükündə dəyişiklik edilə bilən tillərin yükü paketin parametrlərinin siyahısından, predikatdan, diaqnostik məlumatdan ibarətdir və bu informasiya da həmin cədvələ yazılır.

Tətbiqi proqram paketləri inteqrasiya olunan zaman pasportların formalaşdırılmasında dialoq rejimi istifadə olunur, və istifadəçiyə bir sıra blankları doldurmaq təklif olunur. Cədvələ yazılacaq informasiyaların çox böyük hissəsi kompüter tərəfindən, kompleksləşdirilən uyğun paketlərin informasiya mühitinin bütün sistem fraqmentlərinin analizinin nəticəsinə əsaslanaraq, hazırlanır. Əvvəlcə mayak sistemdə kompleksləşdirilən paketlərin adlarının sadalanması zəruridir, həm də paketlərin adlarının biri-biri ilə , həm də mayak sistemi ilə üst-üstə düşmədiyi yoxlanılır. Sonra isə Z-şəbəkədə paketlərin inteqrasiyası ardıcıllığı aydınlaşdırılır. Mayak sistemin yaradıcısına aşağıdakı matris-elektron cədvəlin doldurulması təklif olunur:

| | Ad-1, | Ad-2, | ..., | Ad-3 |
|---|---|---|---|---|
| Ad-1 | 0 | 0 | ..., | 0 |
| Ad-2 | 0 | 0 | ..., | 0 |
| Ad-3 | 0 | 0 | ..., | 0 |

Tətbiqi proqram paketlərinin **i** və **j** birləşdirilməsi vahidin matrisin lazım olan mövqeyinə yerləşdirilməsidir, burada i – paketin adından ibarət sətir, j – sütunun nömrəsidir. PL-MS sistemi ixrac/idxal verilənlərə malik razılaşdırılmaqla və razılaşdırılmamaqla tipli paketlərin birləşməsi olduğundan, doldurulan cədvəlin Z-şəbəkəsi ilə əlaqədar olduğu yoxlanılmır. Beləliklə belə nəticəyə gəlmək olur ki, Z-şəbəkə hər bir təpə nöqtəsi mayak qrafı olan yalnız əlaqəli qraf deyil, həm də bir-biri ilə əlaqəli olmayan mayak qrafları toplusu yaxud əlaqəsiz altqraflar toplusu ola bilər.

Z-şəbəkədə i təpə nöqtəsindən j təpə nöqtəsinə tilin olması məsələnin həlli zamanı avtomatik olaraq i adlı paketdən j adlı paketə keçmək imkanının olduğunu göstərir. Belə tilin olmaması isə i adlı paketdən j adlı paketə keçərkən əllə i adlı paketin işinin nəticələrinin j paketi üçün məsələnin şərtində yenidən təyin olunacağını göstərir.

Paketlərin inteqrasiyasının sonuncu addımında pasportlarının formalaşdırılması hər bir paketdə verilənlərin ixrac/idxal sxemi aydınlaşdırılır. Daxil edilmiş sxem üzrə mayak sisteminin işə salınması üçün JCL dilində prosedur yaradılır və sistem işə salınır. Bu çox mürəkkəb prosedurdur. Onun tərkibinə informasiya mühitinin xeyli sayda sistem və istifadəçi fraqmentləri daxildir və bunların hər biri üçün uyğun spesifikasiyalar tələb olunur.

Paketlərin inteqrasiyası pasportlarının formalaşdırılmasının son mərhələsində addımlarla və iterativ rejimdə yaradılan sistem fraqmentlərinin sona yetməsi haqqında sual qoyulmur. Addımların sayından və iterasiyadan asılı olmayaraq, bütün aralıq seqmentlər və bölmələr formalaşdırılmayana qədər, sistem fraqmenti uyğun adla yaradılmır. Layihələndirmənin ayrı-ayrı addımlarında informasiyalar aralıq seqmentlər və bölmələr çoxluğunda tədricən toplanır. Başqa sözlə desək, PL-MS mayak sistemin informasiya mühitinin fraqmentlərinin ikitərəfli adlandırılması mexanizmi layihələndirmənin ixtiyari mərhələsində bu sistem haqqında toplanmış məlumatların tamlığının birqiymətli olduğunu deməyə imkan verir.

Mahiyyətcə, dialoqlu instrumental proqramlaşdirma sistemi – DİPS, PL-1 universal proqramlaşdırma dilinin translyatoru üzərində qurulmuş makrogenerator, makroassembler və əməliyyat sisteminin JCL- tapşırıqların idarə olunması instrumentarilərindən ibarətdir. Makrogeneratorun tərkibinə informasiya mühitinin fraqmentlərinin adlarının PL-1 və Assembler dilləri səviyyəsində formalaşdırmaq üçün emalı zəruri olan makroslar daxildir. Əməliyyat sisteminin əlaqə redaktorunun idarəedci cümlələrini formalaşdırmaq və tapşırığı işə salmaq üçün makroslar realizə olunmuşdur.

# NƏTİCƏ VƏ TƏKLİFLƏR

İnteraktiv mayak sistemlər Dialoqlu Instrumental Proqramlaşdirma Sisteminin (DIPS – kompleksinin) vasitəsi ilə kompüterdə məsələlərin icrası texnologiyasının dörd məlum mərhələsini avtomatlaşdırmaqla yaradılırlar və bu sistemlər dialoq rejiminin tətbiq olunduğu unikal proqram vasitələridirlər.

İribloklu proqramlaşdırma ideyasının əsas aspektlərindən biri də hazır proqram vasitələrinin fondlaşdırılması, yəni uyğun ölkənin alqoritm və proqramlar fondlarında saxlanılması və müstəqil şəkildə istifadə oluna bilməsidir. Hal-hazırda ən çox populyarlıq, tətbiqi proqramlar paketləri və avtomatlaşdırılmış tətbiqi proqramlar paketləri generasiya edən sistemlər (tətbiqi proqramlar paketlərinin generatorları) qazanmışlar. Onlar inkişaf etmiş və geniş tərkibli fayllar sisteminə, yəni xarici yaddaşda qorunan fayllarda saxlanılan çoxsaylı (dəfələrlə) istifadə oluna bilən modullara əsaslanaraq yaradılırlar.

Sistem proqram təminatı sahəsində aparılan elmi araşdırmalar göstərir ki, hal hazırda da instrumental proqram təminatı elementlərinin yaradılması metodları durmadan inkişaf etdirilir. Çoxsaylı metodların hazırlanmasına baxmayaraq bu metodların yalnız bəziləri praktikada geniş tətbiqini tapır.

Proqramların yaradılması yaradıcılıq prosesi olsa da, bu prosesin sadələşdirilməsi üçün tətbiq oluna biləcək çoxsaylı alqoritmlər mövcuddur. Məsələlərin həlli üçün əsas alqoritm, qoyulmuş məsələnin müstəqil alt məsələlərə bölünməsi alqoritmidir.

Yekun olaraq demək olar ki, testləşdirmə destruktiv proses olmaqla proqramda səhvlərin aşkar olunması cəhdidir (səhvlərin olduğu güman edilir). Səhvlərin tapılmasına imkan yaradan testlər toplusu uğurlu hesab olunur. Təbii ki, son nəticə olaraq hər kəs testin köməyi ilə əmin olmaq istəyir ki, onun proqramı öz təyinatına uyğundur və təyin olunmadığı funksiyanı yerinə yetirmir. Məqsədə çatmaq üçün ən yaxşı üsul səhvlərin bilavasitə axtarılmasıdır.

İxtiyari səviyyəyə malik mütəxəsisi işə qəbul edən zaman, onu əlavə olaraq professsional tədrisə cəlb etməklə, ona ixtisasını artırmaqda köməklik göstərmək,

bilmədiklərini daha savadlı mütəxəsislərdən öyrənə bilməsinə şərait yaratmaq və beləliklə onun xidməti pilləkənlə yüksələ bilməsinə şərait yaratmaq zəruridir.

Tələblər haqqında razılaşmaların nəzərdən keçirilməsi, təsdiqi və düzəlişlərin aparılması hazırlanmanın planlaşdırılması prosesinin ən vacib mərhələləridir. Bilavasitə tələblər haqqında razılaşmalara əsasən bütün maraqlı tərəflər, məhsulun hazırlanması ilə bağlı nələrin gözlənildiyini müəyyənləşdirməyə çalışırlar.

Proqram məhsulunun hazırlanmasının təşkili zamanı dalğa effektinin ciddi bir çatışmayan cəhətini yaddan çıxarmaq lazım deyildir: xarici spesifikasiyaların təsdiq olunması ilə bağlı yubanma bir çox funksiyaların yerinə yetməsinə çox pis təsir göstərə bilər. Çünki, sınaq spesifikasiyalarını təsdiq etmək olmaz, reklam materiallarını sona çatdırmaq mümkün deyildir.

Belə nəticəyə gəlmək olur ki, tətbiqi proqram paketlərinin generatorlarının yaradılması sistem və tətbiqi proqramlaşdırmanın ayrı-ayrı sahələrində yaradılan və tətbiq olunan, indiyə qədər bir-biri ilə zəif qarşılıqlı fəaliyyətdə olmaqla paralel inkişaf edən, müxtəlif vasitələrin və metodların bir-birinə yaxınlaşmasına və birlikdə istifadə olunmasına gətirib çıxardı. Əgər nəzərə alsaq ki, tətbiqi proqramlar paketlərində hesablama prosesinin planlaşdırılmasının təşkili zamanı hal-hazırda süni intellekt yaradılan zaman əldə olunan naliyyətlərdən istifadə olunur, onda belə qərara gəlmək olar ki, problem-oriyentasiyalı instrumental proqramlaşdırma sistemlərinin yaradılması problematikası sistem və nəzəri proqramlaşdırmanın əsas istiqamətlərindən birini təşkil edir.

Modelləşdirmə əsasən kompüterlərdə müxtəlif fiziki proseslərin tədqiqi zamanı, xüsusilə də bu proseslərin riyazi şərhi məlum olduqda, istifadə olunur. Buna baxmayaraq, belə yanaşma böyük proqram kompleksləri yaradılan zaman proqramçıların diqqət mərkəzində olmalıdır.

Proqram məhsulunun yaradılmasında ***enən yanaşmanın***, yəni ***aşağı düşən yanaşmanın*** və hətta kodlaşdırmaya qədər proqramların sənədləşdirilməsinin vacibliyini başa düşməlidir.

Qurulmuş tətbiqi proqramlar paketləri POLE generatoru vasitəsi ilə, dəyişən strukturlu universal uyğunlaşdırılmış modullardan, həll olunacaq məsələlərin alqoritmlərinin giriş dilində şərhi vasitəsi ilə alınır.

Avtonom sistem hissəsinə malik paket qurulmuş paketə nisbətən müəyyən üstünlüklərə malikdir; o yalnız onun təyin olunduğu funksiyaları yerinə yetirir və bu paket konkret tətbiq sahəsinin mütəxəsisləri üçün yaradılmışdır. Yaradılan tətbiqi proqramlar paketlərinin funksional hissəsi kifayət qədər araşdırılmış və müsbət nəticələr əldə olunmuşdur.

Tətbiqi proqramlar paketlərinin generatorlarının yaradılması təcrübəsini analiz etməklə belə nəticəyə gəlmək olur ki, problem-oriyentasiyalı instrumental proqramlaşdırma sistemlərinin effektiv şəkildə fəaliyyəti, generatordan asılı olmayaraq fəaliyyət göstərən avtonom sistem hissənin reallaşdırılması yolu ilə əldə olunur.

PL-MS sistemi ixrac/idxal verilənlərə malik razılaşdırılmaqla və razılaşdırılmamaqla tipli paketlərin birləşməsi olduğundan, doldurulan cədvəlin Z-şəbəkəsi ilə əlaqədar olduğu yoxlanılmır. Beləliklə belə nəticəyə gəlmək olur ki, Z-şəbəkə hər bir təpə nöqtəsi mayak qrafı olan yalnız əlaqəli qraf deyil, həm də bir-biri ilə əlaqəli olmayan mayak qrafları toplusu yaxud əlaqəsiz altqraflar toplusu ola bilər.

# İSTİFADƏ EDİLMİŞ ƏDƏBİYYAT


1. Карминский А. М., Черников Б. В. Применение информационных систем в экономике. Учебное пособие, изд-во Форум Инфра-М, 2012. - 320 стр.
2. Информационные системы в экономике. Под ред. проф. Романова А. Н., Одинцов Б. Е. Учебное пособие, 2-е издание, изд-во Вузовский учебник, 2010, - 410 стр.
3. Терехов А. Н. Технология программирования. Учебное пособие, изд- во Бином, 2011. – 148 стр.
4. Анализ опыта реализации диалоговых систем. /Л.В. Кокарева, И.И. Малашинин, О.Л. Перевозчикова, Е.Л. Ющенко // Упр. системы и машины. —1987.—№4. -с. 63-69.
5. Аннотированный библиографический указатель по диалоговым системам. - Пущино: НИВЦ АН СССР, 1986.—220 с.
6. Бабаев И.О., Лавров С.С. Развитие автоматизированной системы решения задач СПОРА // Пакеты прикладных программ. Инструментальные системы. —М.: Наука, 2007.—с. 5-18.
7. Диалоговая информационная система управления пакетами прикладных программ ДИСУППП / О.Л. Перевозчикова, И.В. Криштопа, Г,Э. Микаилов, М.Н. Мусаев и др.; Ин-т кибернетики им. В.М.Глушкова АН УССР—Киев, 1985 —290с
8. Ещенко В.Г. О реализации на языке высокого уровня математического обеспечения пакета программ РАЗМЕЩЕНИЕ.— Программирование, 1983, № 2, с.12—16.
9. Каталог диалоговых систем: материалы по математическому обеспечению—Киев: Ин-т кибернетики АН УССР, 2006, —232 с.
10. Кахро М.М., Калья Л.П., Тыугу Э.Х. Инструментальная система программирования ЕС ЭВМ (ПРИЗ).-М.: Финансы и статистика, 1988.- 181 с.

11. Компьютерные технологии обработки информации: Учебное пособие/С.В. Назаров, В.И.Першков, В.А.Гафинцев и др.; Пол. ред. С.В. Назарова.—М.: Финансы и статистика, 1995—248 с.
12. Крижановский В.В., Мусаев М.Н., Перевозчикова О.Л. Создание схем вычислений маршрутных систем.—Киев, 1984.—30с.—(Препринт/АН УССР, Институт Кибернетики; 84-27).
13. Мусаев М.Н. Об одной проблеме проектирования проблемно-ориентированных систем. Труды конф. «Новые информационные технологии и проблемы прикладной математики», Баку, 1997.
14. Мусаев М.Н. Об одном способе построения моделей предметных областей.—Изв. АН Азербайджана, сер. Физ.-техн. и матем. Наук, XYI том, № 5-6, 1995.
15. Пакеты программ офисного назначения: Учеб. Пособие /С. В. Назаров. Л.П. Смольников, В.А. Тафинцев и др.;Под, ред. проф. С.В. Назарова.-М.: Финансы и статистика, 2007—320 с.
16. Перевозчикова О.Л. Модели общения при решении задач на ЭВМ. //Упр. системы и машины.—1987.—№5.—с.61-68.
17. Перевозчикова О.Л., Ющенко Е.Л. Диалоговые системы.—Киев: Наук. думка, 1990.—184 с.
18. Ездаков А. Л. Функциональное и логическое программирование. Учебное пособие. Изд-во Бином, 2011. – 119 стр.
19. Перевозчикова О.Л., Ющенко Е.Л. Системы диалогового решения задач на ЭВМ—Киев: Наук. думка, 1986.—264 с.
20. Чаплинскас А.А., Матулис. В.А. Система «Вильнюс»: конпепции, структура и технологии ее использования. — Вильнюс: Ин-т математики и кибернетики АН Лит. ССР, 1981 .—146 с.
21. Петрушин В.А. Экспертно-обучающие системы.-Киев: Наук думка, 1992.-с.196.

# РЕЗЮМЕ

Создание генераторов пакетов приложений привело к системному подходу и применению различных средств и методов, разработанных и применяемых в отдельных областях программирования и параллельной разработки, с пока слабым взаимодействием. Принимая во внимание тот факт, что при планировании вычислительного процесса в пакетах прикладного программного обеспечения, достижениях, которые в настоящее время используются для создания искусственного интеллекта, можно утверждать, что проблемно-ориентированное инструментальное программирование является одним из основных направлений системного и теоретического программирования.

Эффективное функционирование проблемно-ориентированных систем инструментального программирования достигается за счет реализации автономной части системы независимо от генератора.

## SUMMARY

Creation of application packages generators led to system approach and application of different means and methods, developed and applied in separate areas of programming and parallel development, with weak interactions so far. Considering the fact that when planning the computing process in packages of application software packages, the present achievements are used when the artificial intelligence is generated, then it can be argued that the problem-oriented instrumental programming system is one of the main directions of system and theoretical programming.

Effective functioning of problem-oriented instrumental programming systems is achieved through the implementation of the autonomous system, regardless of the generator.