Dos. İ.K.Musayev

Avtomatlaşdırılmış informasiya emalı sistemlərinin (AVİNEMSİ-lərin) layihələşdirilməsi

50 saatlıq mühazirə

GİRİŞ. AVİNEMSİ-nin məqsədi və vəzifələri

Avtomatlaşdırılmış informasiya emalı sistemi (AVİNEMSİ) dedikdə, geniş mənada, hesablayıcı və alqoritmik vasitələrin, həmçinin rabitə vasitələrinin zamanda və məkanda qarşılıqlı fəaliyyət göstərdiyi sistem nəzərdə tutulur.

AVİNEMSİ fənni avtomatlaşdırılmış informasiya emalı sisteminin proqram təminatının yaradılması qaydaları, ardıcıllığı və mərhələlərini öyrənir.

Proqram təminatı (PT) üzrə məhsul istehsalı sənayesi XX əsrin 50-ci illərində yaranmağa başlasa da, 60-cı illərin sonlarınadək bu sahəyə ciddi diqqət yetirilməmişdir. Çünki o zaman kompüter biznesində PT-nin xüsusi çəkisi olduqca az idi. 1970-ci ildə ABŞ-da PT istehsal edən firmaların illik dövrüyyəsi 0,5 milyard dollar idi ki, bu da kompüter biznesinin ümumi dövrüyyəsinin cəmi 3,7%-ni təşkil etmişdi. PT istehsalı sahəsində ciddi artım XX əsrin 70-ci illərində başlamışdır. Bu gün PT satışı ildə 100 milyard dollardan artıqdır.

Hal-hazırda PT istehsalı dünya iqtisadiyyatının nəhəng sahəsinə çevrilmişdir. Bu işlə 3 milyondan artıq mütəxəssis məşğuldur. Bir-neçə milyon adam *Microsoft* və ya *IBM* kimi korporasiyaların uğurlarından birbaşa asılıdır.

PT sistemlərinin yaradılması təcrübəsi göstərir ki, bu iş yalnız yüksək ixtisaslı mütəxəssislər tərəfindən icra edilə bilən məntiqi cəhətdən mürəkkəb, böyük zəhmət və vaxt tələb edən işdir. Lakin indiyədək bu sistemlərin yaradılması, adətən, ekspert qiymətlərinə və praktiki təcrübəyə əsaslanan formallaşdırılmamış metodların tətbiqi ilə həyata keçirilmişdir. Bundan əlavə, PT-nin yaradılması və fəaliyyəti prosesində istifadəçilərin tələbləri daim dəyişilir və dəqiqləşdirilir ki, bu da həm sistemin yaradılmasını, həm də fəaliyyətini daha da mürəkkəbləşdirir. Bununla bağlı olaraq XX əsrin 80-90-cı illərində bir sıra tədqiqatçılar PT-nin hazırlanmasında böhran yaranması

barədə yazılar dərc etdirirdilər. Məsələn, 1995-ci ildə 364 amerika korporasiyası tərəfindən hazırlanmış PT üzrə 23 mindən çox layihənin 52,7%-i xeyli gecikdirilmiş, 31,1%-i başa çatmadan ləğv edilmiş, orta hesabla layihə dəyəri 89%, yerinə-yetirilmə müddəti 122% artmış, cəmi 16,2%-i vaxtında və bahalaşmadan icra edilmişdir. 1998-ci ildə gecikdirilmiş layihələr 46%, ləğv edilmiş layihələr 28%, vaxtında icra edilən layihələr 26% təşkil etmişdir ki, bu da vəziyyətin cüzi yaxşılaşmasından xəbər verirdi. Son illərdəki tədqiqatlar göstərir ki, bu sahədə baş verən müsbət dəyişikliklər hələ də çox ləng gedr. Olan müsbət dəyişikliklər də layihələşdirmənin keyfiyyətinin artması hesabına deyil, yalnız layihələşdirmə miqyaslarının daralması hesabınadır.

Mürəkkəb PT sistemlərinin yaradılması, layihələrin dəyərinin, keyfiyyətinin və müddətlərinin proqnozlaşdırılması proseslərinə nəzarət edilməsinə yaranan obyektiv tələbat ötən əsrin 60-cı illərində PT layihələrinin yaradılmasının kustar üsullarndan sənaye üsullarına keçirilməsi zərurətini ön plana çəkdi. Beləliklə, proqram mühəndisliyi1 deyilən anlayış meydana çıxdı. Bu anlayış ilk dəfə 1968-ci ildə *NATO*-nun Elmi komitəsində konfrans mövzusu kimi istifadə edilmişdir. 1975-ci ildə Vaşinqtonda proqram mühəndisliyinə həsr edilmiş ilk beynəlxalq konfrans keçirildi. Konfransın nəticələri ayrıca toplu[2] şəklində nəşr edildi. Həmin vaxtdan proqram mühəndisliyi, bir tərəfdən, PT yaradılması üzrə mühəndis metod və vasitələri çoxluğu, digər tərəfdən də PT yaradılmasına, istismarına və müşayiət edilməsinə ciddi kəmiyyət yanaşması tətbiqini öyrənən elmi istiqamət kimi təyin edilmişdir.

Proqram mühəndisliyinin təməlini təşkil edən formal qaydaya görə, PT layihələşdirməsi öyrənilə və təkmilləşdirilə bilən formal prosesdir. PT yaradılması metod və vasitələrinin mənimsənilməsi və düzgün tətbiqi PT layihələşdirilməsi prosesinin keyfiyyətinin yüksəldilməsinə, idarə olunanlığını təmin etməyə və ömür müddətini uzatmağa imkan verir.

Beləliklə, "AVİNEMSİ-nin layihələşdirilməsi" fənninin məqsədi avtomatlaşdırılmış informasiya emalı sistemləri üzrə layihələşdirmə prinsiplərinə yiyələnmiş geniş

---

[1] *Software engineering*
[2] *IEEE Transactions on Software Engineering.*

profilli layihələşdiricilər hazırlamaqdan ibarətdir.

Bu fənni öyrənən AVİNEMSİ-nin arxitekturası barədə biliklərə yiyələnmiş olmalı; müasir AVİNEMSİ siniflərinin funksionallığı, təhlükəsizliyi və etibarlılığı üzrə tələblərdən xəbərdar olmalı; təməl normativ sənədlər, layihələşdirmə üzrə reqlamentləşdirilmiş qaydalar və sınaq və tətbiq barədə məlumatlı olmalı; müasir *CASE*-vasitələrindən istifadə etməyi bacarmalıdır.

"AVİNEMSİ-nin layihələşdirilməsi" fənnində əsas diqqət layihələşdirməyə hazırlıq, tələblərin təhlili, arxitekturanın layihələşdirilməsi və təfərrüatlı layihələşdirmə məsələləri üzərində cəmləşdirilir.

"AVİNEMSİ-nin layihələşdirilməsi" proqramlaşdırma və informasiya texnologiyaları ilə sıx bağlı fənndir.

Bu fənnin iqtisad yönümlü Ali məktəblərdə tədrisi iqtisadiyyatın, o cümlədən, iqtisadi vahidlərin sistem təhlili barədə zəruri biliklərə də malik olmasını məqsədəuyğun edir.

1.Sistem təhlilinin elementləri

Bütövlükdə iqtisadiyyat kimi, müəssisənin də fəaliyyəti çoxsaylı amillərin təsiri altında baş verən proseslər məcmusudur. Müəssisənin modelləşdirilməsi daha əhəmiyyətli elementləri ön plana çəkməklə idarəetmənin sadələşdirilməsini nəzərdə tutur. Lakin iqtisadi fəaliyyətin modelləşdirilməsi zamanı əhəmiyyətsiz sayılaraq nəzərdən qaçırılan amillər hadisələrin gedişinə ciddi təsir edə bilər. Buna görə də müəssisənin modelləşdirilməsi zamanı sistem yanaşması prinsiplərinə xüsusi diqqət ayrılmalıdır. Sistem yanaşması həm daxili, həm də xarici qarşılıqlı əlaqələrin mövcudluğunu ön plana çəkir. Sistem yanaşması informasiyanın hərəkət etdiyi və dəyişdirildiyi bütün altsistemləri və modulları əhatə edir.

Sistem təhlilinin əsas anlayışları: sistem, əks-əlaqə, ətraf mühit və sinerqetik effektdir.

Sistem dedikdə, müəyyən məqsədə nail olmaq üçün birgə fəaliyyət göstərən qarşılıqlı əlaqəli elementlər məcmusu başa düşülür. Elementlər qarşılıqlı əlaqəli olduqlarına görə bir elementin dəyişməsi digərlərini də dəyişdirə bilir. Bu xassə

qarşılıqlı asılılıq adlanır. Sistem elementlərinin bir-birindən asılılığı müxtəlif dərəcəli ola biləndir. Bəzi elementlər sıx, başqaları zəif asılılığa malik olsalar da bəzi elementlər arasında heç bir əlaqə olmaya da bilir. Əlaqəli sistemlər arasında intensiv informasiya və maddi məhsul mübadiləsi baş verir. Bu əlaqənin pozulması sistemin işini poza bilir.

Əks-əlaqə sistemin çıxışı ilə girişi arasındakı əlaqədir. Çıxışın arzu edilən nəticədən fərqləndiyi müşahidə edildikdə, girişə korreksiya edici informasiya verilir.

Ətraf mühit sistemi əhatə edən mühitdir. Sistem tam izolyasiya şəraitində fəaliyyət göstərə bilmir. O, həmişə digər sistemlə təmasda olur. Sistem ətraf mühitdən öz sərhədləri ilə ayrılır. Sistemin nəzarəti altında olanlar onun təşkilediciləri, qalanlar isə ətraf mühitə aiddir. Sistemin sərhədləri sabit deyil, dəyişə biləndir. Ətraf mühitlə əlaqə intensivliyi müxtəlif ola biləndir. Bu əlamətə görə sistemləri açıq və qapalı sistemlərə bölürlər.

Sinerqetik effekt – sistem effekti olub, tamın öz tərkib hissələrinin cəmindən böyük olduğunu ifadə edir. Belə ki, hissələrin birləşdirilməsi elə bir yeni ümumi nəticə yaradır ki, bu nəticəyə ayrı-ayrılıqda heç bir hissə nail ola bilmir.

Sistemin davranışı müxtəlif cəhətləri xarakterizə edən müxtəlif üsullarla təsvir edilə bilir. Sistem yanaşması mövqeyindən, davranışın modelləşdirilməsi zamanı aşağıdakı anlayışlar nəzərə alınmalıdır:

1. *Çoxvariantlılıq*. Bu, bir nəticənin bir-neçə üsulla alına bilməsini nəzərdə tutur;
2. *Sistem parametrlərinin çevikliyi*. Bu, sistemin çevikliyini təmin edən əsas tələblərdən biridir. Sistem mövcud olduğu müddət ərzində öz girişində güclü kənarlaşmalara məruz qala bilər. Dayanıqlı davranış üçün sistem bu kənarlaşmalara və dəyişkən şəraitə reaksiya verməyi və buna uyğunlaşmağı bacarmalıdır.

2.Sistemlər və sistem modelləri

Sistemlər nəzəriyyəsinin, kibernetikanın, sistem yanaşmasının, bütövlükdə sistemologiyanın mərkəzi konsepsiyası *sistem* anlayışıdır. Buna görə də sistemə çoxlu təriflər verilmişdir. Məsələn, Uemov A.İ. sistemə verilmiş təriflərin 35-ni toplamışdır. Biz burada sistemə verilən təriflərin təhilini deyil, təkamülünü izləməyi qarşımıza

məqsəd qoymuşuq. Odur ki, sistemin *ilk tərifi* onun problemli situasiya ilə bağlılığından törəmişdir. Bu mövqedən sistem—məqsədəçatma vasitəsi, başqa sözlə isə, problemin həlli vasitəsidir. Yəni problem olmayan yerdə sistem yoxdur. Digər tərəfdən isə, sistemin ilk tərifi göstərir ki, sistem—məqsədin mühitdə kölgəsidir. Göründüyü kimi, bu tərifdə sistemin konstruksiyası arxa plandadır. Bu cəhəti ön plana çəkmək üçün isə sistemin vizual ekvivalentinə keçmək zəruridir. Bunun üçün sistemin "qara qutu" modelindən başlamaq lazım gəlir. Lakin "qara qutu" sistemin yalnız davranışını izləməyə imkan verdiyindən, təbii olaraq növbəti addım kimi, "sistemin tərkibi" modelinə keçid zərurəti yaranır. Bu modeldə sistem hissələrin məcmusu kimi təsvir edilir. Sistemin bölünməz tərkib hissələri *element* adlanır. Elementdən böyük hissəyə *altsistem* deyilir. Lakin "sistemin tərkibi" modelini qurmaq da elə asan məsələ deyildir. Çünki bu model də məqsədlə bağlıdır. Məqsəd müxtəlifliyi eyni obyekt üçün müxtəlif tərkibli modellərin qurulmasına səbəb olur.

Digər tərəfdən, bir sıra suallar var ki, onlara cavab vermək üçün obyektin təşkilediciləri arasındakı *münasibətlər* aydınlaşdırılmalıdır. Lakin müxtəlif elementlər arasında praktiki olaraq sonsuz sayda münasibət ola bilər. Odur ki, hər dəfə yalnız konkret məqsədə çatmağı təmin etməyə kifayət edən elementlərarası münasibətlər ön plana çəkilir ki, belə münasibətlər çoxluğuna da sistemin *quruluşu* deyilir.

Münasibətlə xassə arasında sıx bağlılıq var. Lakin münasibətdə azı iki element iştirak edirsə, xassə bir obyektin müəyyən bir atributu kimi çıxış edir. Bu mənada, xassə "bir yerli" (unar) münasibət, ən sadə münasibət isə "iki yerli" (binar) münasibətdir. Bununla belə, xassə münasibətin törəməsidir.

Göründüyü kimi, xassə əslində, obyektin atributu deyil, hər hansı münasibətlər çoxluğunun mücərrədləşdirilməsinin məhsuludur. Bu mənada, xassə - qarşılıqlı fəaliyyətin sıxılmış nəticəsidir.

Bu deyilənlərin yekunu olaraq sistemin ikinci tərifini belə formalaşdırmaq olar ki, sistem - mühitdən ayrılmış qarşılıqlı əlaqəli elementlərin elə məcmusudur ki, özünü mühitlə qarşılıqlı fəaliyyətdə olan tam kimi aparır. Göründüyü kimi, bu tərif sistemin həm "qara qutu", həm "tərkib" və həm də "quruluş" modellərini özündə birləşdirir.

Odur ki, bu modellərin sintezi kimi yeni bir model - “sistemin quruluş sxemi” modeli alınır. Bunu bəzən “şəffaf qutu” da adlandırırlar. Lakin bütün bu dediklərimiz yalnız statik momentləri əhatə edir. Halbuki daim hərəkətdə olanı hərəkətsiz vasitələrlə izah etmək mümkün deyildir. Odur ki, sistemin dinamik momentlərini də əks etdirən modellər qurmaq zərurəti yaranır. Müxtəlif obyektlərdə baş verən dəyişiklikləri təsvir etmək üçün müxtəlif saylı dinamik modellər qurmaq lazım gəlir. Lakin dinamik modellər də statik modellərə münasib və mütənasib qurulur. Yəni həm “qara qutu”ya, həm “tərkib”ə, həm də “quruluş”a münasib və mütənasib dinamik modellər qurmaq lazım gəlir. Belə ki, hələ “qara qutu” mərhələsində sistem dinamikasının iki tipi: sistemin *fəaliyyəti və inkişafı* fərqləndirilir. Sistemin fəaliyyəti sabit məqsədin reallaşdırılması ilə əlaqədar olaraq sistemdə baş verən dəyişikliklərdirsə, inkişafı da məqsədin dəyişilməsi ilə bağlı olan proseslərdir. İnkişaf çox zaman sistemin quruluş dəyişmələrində təzahür edir. Çünki adətən köhnə quruluş yeni məqsədə uyğun gəlmir. Göstərək ki, fəaliyyətlə inkişaf həm ardıcıl, həm də paralel proseslər kimi mövcud ola bilər.

3.Süni və təbii sistemlər

Sistemin əsas əlamətlərindən biri onun *qurulmuşluğu* və elementləri arasında *məqsədəuyğun* əlaqələrin olmasıdır. Məhz bu məqamdan məqsəd anlayışındakı qeyri-müəyyənlik ön plana keçir. Odur ki, subyektin məqsədi ilə obyektin məqsədini fərqləndirmək zərurəti meydana çıxır. Bunu etmədikdə isə məntiqi ziddiyyət alınır. Mütləq Yaradan ideyası məhz həmin ziddiyyətin nəticəsidir! Halbuki məqsəd— *özünüqoruma effekti* olaraq bütün sistemlərə xas cəhətdir! Odur ki, məqsəd fərqləri mübarizə, məqsəd uyarlıqları həmrəylik törədir. Birinin özünü qoruması üçün başqası məhv edilməlidirsə, konflikt yaranır. Birinin özünü qoruması başqasını məhv etmirsə, yanaşıyaşama (yarış) alınır. İkisinin özünü qoruması üçün birləşmək zərurəti varsa, həmrəylik meydana çıxır. Odur ki, dünyada hər şey, canlı-cansız, makro-mikro nə varsa, hamısı *yoğrulma*da mövcuddur. Bu yoğrulma mahiyyətcə konflikt, yanaşıyaşama və həmrəylik kimi üç momentin sintezidir. Yoğrulmanın müxtəlif məqamları və nahiyələri müxtəlif intensivlik səviyyələri ilə xarakterizə olunurlar.

Sistem məqsədəçatma vasitəsidirsə, məqsəd də sistem elementlərini öz ətrafında birləşdirən nöqtədir. Gördüyümüz kimi, məqsəd—bütün aspektlərdə sistemin yaradıcısıdır. Odur ki, "sistem—məqsədin mühitdə kölgəsidir"-deyirlər.

Lakin məqsədi və quruluşu məlum olmayan obyektin də sistem olduğunu fərz etmək metodoloji cəhətdən faydalıdır. Çünki yalnız bu halda elmi tədqiqata ehtiyac yaranır. Bununla belə, süni və təbii sistemləri fərqləndirmək lazımdır. Bu isə məqsəd anlayışını genişləndirməyi tələb edir. Belə ki, süni sistemin məqsədi subyektiv məqsəd olub, arzu edilən vəziyyəti ifadə edirsə, təbii sistemin məqsədi də obyektiv məqsəd olmaqla, gələcək vəziyəti əks etdirir. Dünyaya bu yöndən baxdıqda aydın olur ki, Dünya başdan-başa sistemdir və sistem olmayan heç nə yoxdur. Lakin subyektiv və obyektiv məqsədlərin etiraf edilməsi bir tərəfdən də düşüncənin sistemliliyi ilə təbiətin sistemliliyi arasındakı ümumiliklə fərqliliyin bariz təzahürüdür.

Göründüyü kimi, sistemlər həm ümumi, həm də fərqli cəhətlərə malikdir. Ümumi cəhət bütün sistemləri birləşdirirsə, fərqli cəhətlər də onları bir-birindən ayırır. Məsələn, mənşəyinə görə sistemləri *süni, təbii və qarışıq* sistemlərə bölürlər. Bunların içərisində *idarəetmə* sistemləri xüsusi maraq doğurur. Çünki belə sistemlər daha çox praktik əhəmiyyət kəsb edir. Həm də bu sistemlər adətən "böyük" və "mürəkkəb" sistemlərdir. Ölçüsünün böyüklüyünə görə modelləşdirilməsi çətin olan sistemlər "böyük sistem" hesab edilir. Səmərəli idarə edilməsi üçün informasiya yetərsizliyi olan sistemlərə isə "mürəkkəb sistem" deyilir. Yəni "mürəkkəb sistem"i yalnız informasiya ilə "doydurmaq"la sadələşdirmək mümkündür. Məhz buna görə də U. R. Eşbi sistem nəzəriyyəçisini "sadələşdirmə eksperti" adlandırır. Odur ki, sistemə verilən təriflərdən birində deyilir ki: "sistem—mürəkkəbdə sadəlik tapmaq üsuludur". S.Bir isə beyni, iqtisadiyyatı, firmanı "çox mürəkkəb sistem" kimi tədqiq etməyi təklif etmişdir. Çünki "çox mürəkkəb sistem"i informasiya ilə axıradək "doydurmaq" mümkün deyildir.

4.Proqram sistemlərinin hazırlanmasına sistem yanaşması

İnformasiya sistemlərinin (İNSİ-lərin) layihələşdirilməsinin metodoloji əsasını sistem yanaşması təşkil edir. Bu yanaşmaya görə, ixtiyari sistem ümumi məqsədə nail

olmaq üçün birgə fəaliyyət göstərən qarşılıqlı əlaqəli elementlər yığımı kimi təsəvvür edilir. Sistem daxilindəki elementlər zaman keçdikcə, öz aralarında və ətraf mühitlə qarşılıqlı fəaliyyət nəticəsində vəziyyətlərini dəyişirlər. Bu baxımdan, sistemlər aşağıdakı prinsiplərə əsasən fəaliyyət göstərir:

-*emercentlik*, yəni ümumi strukura əsaslanan sistem bütövlüyü. Bu halda, ayrı-ayrı elementlərin davranışına bütün sistemin fəaliyyəti mövqeyindən qiymət verilir;

-*gemostazis*, yəni ümumi sistem qarşısında duran məqsədə nailolma prosesində dayanıqlı fəaliyyət göstərmək;

-xarici mühitin dəyişilməsinə *uyğunlaşabilənlik* və sistemdaxili elementlərə təsir etməklə *idarəolunanlıq*;

-sistem qarşısında duran məqsədin dəyişilməsinə uyğun olaraq sistem strukturunun dəyişdirilməsi yolu ilə *öyrənəbilənlik*.

İNSİ – qarşıya qoyulmuş məqsədə çatmaqdan ötrü informasiyanın saxlanması, emalı və təqdim edilməsi üçün istifadə edilən vasitələr, metodlar və işçi heyətinin qarşılıqlı əlaqəli yığımıdır.

Sistemin idarə edilməsi prosesinə xarici mühiti, idarəetmə obyektini və idarəetmə sistemini əlaqələndirən informasiya prosesi kimi baxılır. Bu baxımda xarici mühit və idarəetmə obyekti idarəetmə sisteminə öz vəziyyətləri barədə informasiya verir, idarəetmə sistemi həmin informasiyanı təhlil edib idarəedici qərar qəbul edərək idarəetmə obyektinə göndərir və xarici mühitin təsirinə cavab verir. Lazım gəldikdə, xarici mühitin dəyişikliyinə müvafiq olaraq məqsədini və strukturunu modifikasiya edir.

İNSİ-də idarəetmə obyekti öz alt sistemlərinin qarşılıqlı əlaqəli sxemi şəklində təsvir edilir.

İdarəetmə sistemi (İS) dedikdə, planlaşdırma, uçot, nəzarət, operativ idarəetmə və təhlil kimi idarəetmə funksiyalarını icra edən İNSİ-nin qarşılıqlı fəaliyyət göstərən struktur bölmələri yığımı nəzərdə tutulur.

Planlaşdırma – İNSİ-nin müxtəlif zaman kəsikləri üçün fəaliyyət məqsədlərini təyin edən idarəetmə funksiyasıdır.

Uçot – təsərrüfat proseslərinin icrasının nəticəsi ilə bağlı olaraq idarəetmə obyek-

tinin vəziyyətini əks etdirən idarəetmə funksiyasıdır;

Nəzarət – uçot məlumatlarının plan və normativlərdən kənarlaşmalarını təyin edən idarəetmə funksiyasıdır;

Operativ idarəetmə - uçot məlumatlarının plandan kənarlaşmalarını aradan qaldıran təsərrüfat proseslərinin gedişinin tənzimləyicisi kimi çıxış edən idarəetmə funksiyasıdır;

Təhlil – İNSİ-nin işindəki meylləri təyin edən idarəetmə funksiyasıdır ki, bunun da icrası nəticəsində alınmış informasiya növbəti dövr üçün plan tərtibi zamanı nəzərə alınır.

5. AVİNEMSİ-nin layihələşdirilməsi nəzəriyyəsinin əsas anlayışları. AVİNEMSİ-nin tərkibi və strukturu

İNSİ layihəsi - konkret proqram-texnika mühitində yaradılan və istismar edilən İNSİ-yə aid layihə həllərinin təsvir edildiyi layihə-konstruktor və texnoloji sənədlər məcmusudur.

İNSİ layihəsi obyektləri - funksional və təminat hissələrinin ayrı-ayrı elementləri və ya onların kompleksləridir. *Funksional element* dedikdə, ənənəvi kompozisiyaya uyğun olaraq məsələ, məsələ kompleksi və idarəetmə funksiyası nəzərdə tutulur. İNSİ-nin təminat hissəsinin tərkibində layihələşdirmə obyektləri – sistemin informasiya, proqram və texniki təminatlarının elementləri və ya onların kompleksləridir.

İNSİ layihəsi subyektləri – ixtisaslaşdırılmış təşkilat çərçivəsində fəaliyyət göstərən layihələşdirici mütəxəssis kollektivləri və sifarişçi təşkilatdır.

İNSİ-nin layihələşddirilməsi dedikdə, layihələşdirmə obyekti, metodları və təcrübəsi barədəki giriş informasiyanın dəyişdirilməsi prosesi başa düşülür. Bu baxımdan, İNSİ-nin layihələşdirilməsi tələblərin planlaşdırılması və təhlili, texniki və işçi layihələşdirmə, İNSİ-nin tətbiqi və istismarı kimi müxtəlif "həyat tsiklləri" mərhələlərində layihə həllərinin ardıcıl formalaşdırılmasından ibarətdir. İNSİ-nin layihələşdirilməsinin həyata keçirilməsi tərtib edilən layihənin miqyasına və xüsusiyyətlərinə müvafiq gələn müəyyən layihələşdirmə texnologiyasından istifadəyə əsaslanır.

İNSİ-nin layihələşdirilməsi texnologiyası – İNSİ-nin layihələşdirilməsi metodo-

logiyası və vasitələri, həmçinin layihələşdirmənin təşkili metodları və vasitələri məcmusudur. Layihələşdirmə texnologiyası layihələşdirici və sifarişçi tərəfindən reqlamentləşdirilən texnoloji əməliyyatların ardıcıllığı şəklində meydana çıxır.

Struktur təhlil vasitələri: verilənlər axınları diaqramları, "mahiyyət-əlaqə" diaqramları, vəziyyətlər keçidləri diaqramları

Ümumiyyətlə, sistemlərin modelləşdirilməsi, o cümlədən, struktur təhlil məqsədi ilə istifadə edilən vasitələr 3 qrupa bölünür:

-sistemin icra edəcəyi funksiyanı təsvir edən vasitələr;

-verilənlərarası münasibətləri təsvir edən vasitələr;

-sistemin davranış vaxtından asılılığı (real vaxt aspektlərini) təsvir edən vasitələr.

Verilmiş məsələlərin həlli vasitələrinin geniş müxtəlifliyinə baxmayaraq struktur təhlil metodologiyalarında aşağıdakılar daha tez-tez və səmərəli tətbiq edilir:

-*DFD* – verilənlərin axınları diaqramları (verilənlər lüğətləri və spesifikasiyaları[3] ilə birgə);

-*ERD* – "mahiyyət-əlaqə" diaqramları;

-*STD* – vəziyyətlərin keçidləri diaqramları.

DFD sistemə nəzərən kənar məlumat mənbələrini və verilənlərin ünvanlarını göstərir, məntiqi funksiyaları və bir funksiyanı digəri ilə əlaqələndirən verilən elementləri qrupunu, həmçinin müraciət edilən verilənlər saxlanclarını identifikasiya edir (adlandırır). Verilənlər axınlarının strukturu və bunun komponentlərinin təyini verilənlər lüğətində saxlanır və təhlil edilir. Xırdalama (detallaşdırma) faydalı olduğu hallarda məntiri funksiya *DFD*-nin köməyi ilə xırdalana bilər. Hər bir verilənlər saxlancının məzmunu da verilənlər lüğətində saxlanır və *ERD*-nin köməyi ilə açılır. Real vaxt tətbiqi zamanı *DFD STD*-nin köməyi ilə açılan əlavə təsvir vasitələri ilə zənginləşdirilir.

Verilənlər axınları diaqramları (*DFD*) layihələşdirilən sistemlərin funksional

[3] spesifikasiya (bir şeyin spesifik xüsusiyyətlərinin müəyyən edilməsi, bir şeyin dəqiqləşdirilmiş təsnifi).

tələblərinin modelləşdirilməsinin əsas vasitəsidir. Funksional tələblər *DFD*-nin köməyi ilə funksional komponentlərə parçalanaraq verilənlər axını ilə əlaqələndirilmiş şəbəkə şəklində təsvir edilir. Bu təsvir vasitələrinin əsas məqsədi hər bir prosesin öz girişini öz çıxışına necə çevirdiyini və proseslərarası münasibətləri təsvir etməkdən ibarətdir.

"Mahiyyət-əlaqə" diaqramları (*ERD*) verilənlər modelini işləyib hazırlamaq üçündür və verilənləri və verilənlərarası münasibətləri təyin etməyin standart üsulunu təmin edir. Faktiki olaraq layihələşdirilən sistemdə verilənlərin saxlancları və sistemdəki mahiyyətlər və onların arasındakı fəaliyyət üsulları *ERD* vasitəsilə detallaşdırılır.

Vəziyyətlərin keçidləri diaqramları (*STD*) sistemin əvvəlki və cari fəaliyyətinə əsasən sonrakı fəaliyyətini modelləşdirməyə imkan verir. Modelləşdirilən sistem ixtiyari vaxt momentində sonlu vəziyyətlər çoxluğuna daxil olan yalnız bir vəziyyətdə ola bilir. Zaman keçdikcə sistemin vəziyyəti dəyişə bilir və o bir vəziyyətdən digərinə keçid edir. Bu halda vəziyyətdən vəziyyətə keçid dəqiq təyin edilməlidir.

Göstərilən bu təsvir vasitələri həm mövcud olan, həm də layihələşdirilən sistemləri təsvir etmək üçün yararlıdır. Beləliklə, sistemin nə edəcəyini təsvir edən məntiqi funksiya qurulur. Bu, layihələşdiriciyə göstərir ki, son nəticə nədən ibarət olmalıdır.

6. Proqram vasitələrinin layihələşdirilməsi metodologiyası

AVİNEMSİ-nin layihələşdirilməsi texnologiyası dedikdə, layihələşdirmə prosesinin təşkili, yəni layihənin hazırlanması və modernləşdirilməsi proseslərinin idarə edilməsi metodologiyası, metodları və vasitələri məcmusu nəzərdə tutulur.

Layihələşdirmə metodologiyası – layihələşdirmə vasitələrinin köməyi ilə reallaşdırılan layihələşdirmə metodları yığımının tabe olduğu layihələşdirmə prinsipləri və layihələşdirmə konsepsiyasıdır.

Layihələşdirmə metodu (yanaşması) – AVİNEMSİ-nin yaradılmasının bu və ya digər mərhələsinin həyata keçirilməsi alqoritmidir.

Müxtəlif sistemlərin qurulmasının əsas prinsipi iyerarxik dekompozisiya prinsipidir ki, bu da iki qrup metodologiyadan ibarətdir:

1.Struktur-funksional metodologiyadır ki, burada da sistemin strukturu iyerarxiya-

ya daxil olan funksional elementlərin icra etdikləri funksiyaların terminlərində təsvir edilir.

2. Obyekt yönümlü metodologiyadır ki, burada da obyekt dekompozisiyasından istifadə edilir. Bu metodologiyada sistemin strukturu obyekt və obyektlərarası əlaqələrə aid terminlərlə, sistemin davranışı isə obyektlərarası məlumat mübadiləsi terminləri ilə təsvir edilir.

Struktur-funksional metodologiya.

Struktur təhlildə və layihələşdirmədə: sistemin funksional strukturunu, icra edilən əməllərin ardıcıllığını, funksional proseslər arasında informasiya ötürülməsini və verilənlərarası münasibətləri təsvir etmək üçün müxtəlif modellərdən istifadə edilir.

Bu modellərin reallaşdırılmasının ən geniş yayılmış nümunələri aşağıdakılardan ibarətdir:

1. *SADT*[4] adlı funksional modeli;
2. *IDEF3* modeli;
3. *DFD*[5] - verilənlər axınları diaqramları;
4. *ERD*[6]- "mahiyyət-əlaqə" diaqramı.

SADT metodu 1969-cu ildə Duqlas Ross tərəfindən[7] hazırlanmış və ABŞ-da hərbi, sənaye və kommersiya müəssisələrində geniş məsələlər dairəsinin həllində uğurla istifadə edilmişdir. *SADT* metodunu ABŞ Müdafiə Nazirliyi bəyənmişdir. Bu Nazirlik ABŞ Hərbi Hava Qüvvələrinin təşəbbüsü ilə keçirilən *ICAM* (istehsalın inteqrasiya edilmiş kompüterləşdirilməsi) proqramının əsas hissəsini təşkil edən *IDEF*[8] standartları ailəsinin hazırlanmasının da təşəbbüskarı olmuşdur.

SADT metodu ABŞ Standartlar və Texnologiyalar üzrə Milli İnstitut (*NIST*) tərəfindən 1993-cü ilin dekabrında son redaksiya ilə buraxılmış *IDEF0* standartlar ailəsi

---

[4] *Structured Analysis and Design Technique*
[5] *Data Flow Diagrams*
[6] *Entity-Relationship Diagram*
[7] *SoftTech, Inc.*
[8] *Icam DEFinition*

çərçivəsində reallaşdırılmışdır. 2000-ci ildə Rusiya Dövlət Standartının qərarı ilə РД[9] *IDEF0-2000* sənədi qəbul edildi ki, bu da təşkilati-iqtisadi və istehsal-texniki sistemlərin funksional modellərinin qurulması üçün əsas metodoloji məlumatları və qrafik dil elementlərini əhatə edirdi. Bundan əlavə, РД *IDEF0-2000* sənədi iyerarxik diaqramlar komplektinin rəsmiləşdirilmə (tərtibat) qaydalarını da müəyyən edir. *IDEF0* sənədinin metodoloji təməlini 4 əsas anlayış təşkil edir ki, bunlar da: funksional blok, interfeys qövsü, dekompozisiya və qlossari[10] anlayışlarıdır.

Funksional blok[11] dedikdə, baxılan sistem çərçivəsində müəyyən konkret funksiya başa düşülür. Diaqramda funksional blok düzbucaqlı şəkildə təsvir edilir. Funksional bolkun 4 tərəfinin hər birinin müəyyən mənası (rolu) vardır. Belə ki, düzbucaqlının üst tərəfi "İdarəetmə" (*Control*), sol tərəfi "Giriş" (*Input*), sağ tərəfi "Çıxış" (*Output*), alt tərəfi "Mexanizm" (*Mechanism*) mənası daşıyır. Diaqqramda 5-7-ci bloklar hakim mövqeyə malik bloklardır. Aidi (kontekst) diaqrama aid izahedici mətndə diaqramın qurulması məqsədi (*Purpose*) qısa şərh formasında hökmən göstərilməli və nöqteyi-nəzər (*Viewpoint*) qeyd edilməlidir.

IDEF standartları ailəsinin bir hissəsi olan *IDEF3* modelləşdirmə metodu 1980-ci illərin sonunda işlənib hazırlanmışdır. Bu metod elə modellər üçün nəzərdə tutulmuşdur ki, həmin modellərdəki proseslərdə əməllər ardıcıllığını və əməllər arasındakı qarşılıqlı asılılıqları anlamaq vacibdir. İş vahidləri - *Unit of Work (UOW)* (işlər) modelin mərkəzi komponentləridir. *IDEF3* modelində işlər düzbucaqlı şəklində təsvir edilir və əməliyyat prosesini və əmənin nömrəsini (identifikatorunu) ifadə edən ada malik olur. *IDEF3*-dəki əlaqələr işlərin qarşılıqlı münasibətlərini göstərir. *IDEF3*-dəki bütün əlaqələr biristi-qamətlidir və bu əlaqələrin *zamanda qabaqlama, obyektiv axın və qeyri-səlis münasibət* kimi tipləri vardır. Kəsişmələr (*Junction*) qovuşma və şaxələnmə zamanı oxların qarşılıqlı fəaliyyəti məntiqini və ya növbəti hadisədən əvvəl başa çatmalı olan hadisələr çoxluğunu təsvir etmək üçün istifadə edilir.

DEF – verilənlər axınları diaqramları verilənlər axınları ilə əlaqələndirilmiş funk-

---

[9] РД – руководящий документ
[10] м filol. qlossari (qədim sözlər lüğəti).
[11] *Activity Box*

sional proseslərin iyerarxiyasıdır. Bu təsvirin məqsədi hər bir prosesin öz girişini öz çıxışına necə dəyişdiyini və bu proseslər aarasındakı münasibətləri aşkarlamaqdan ibarətdir. Bu metoda uyğun olaraq sistem modeli informasiyanın sistemə daxil edilmə-sindən istehlakçıya təqdim edilməsinədək olan asinxron dəyişdirmə proseslərini təsvir edən verilənlər axınları diaqramlarının iyerarxiyası kimi təyin edilir.

İnformasiya mənbələri (xarici mahiyyətlər) alt sistemlərə və ya proseslərə informasiya ötürən informasiya (verilənlər) axınları törədirlər. Bu alt sistemlər və ya proseslər də, öz növbəsində, informasiyanı dəyişdirərək yeni axınlar yaradır ki, bunlar da digər alt sistemlərə və ya proseslərə, o cümlədən, verilənlər saxlanclarına (yaddaşa) və ya xarici mahiyyətlərə - istehlakçılara ötürülür.

Verilənlər axınları diaqramları yaradılarkən: *verilənlər axınları, giriş axınların çıxış axınlarına çevrilməsi prosesləri (işləri), xarici mahiyyətlər, verilənlər yığıcıları (saxlanclar)* kimi 4 əsas anlayışdan istifadə edilir.

SADT bütövlükdə sistemin modelləşdirilməsi vasitəsi kimi yaradılmışdır. *DFD* isə İNSİ-nin layihələşdirilməsi üçün nəzərdə tutulan vasitədir. Buna görə də *DFD* istifadə üçün daha perspektivlidir. Çünki *DFD* həm də *ERD* ilə uyğunlaşdırılmışdır. *DFD*-də verilənlərin struktur təsviri də mövcuddur. Bu isə *ERD* qurmaq üçün istifadə edilir.

Verilənlərin modelləşdirilməsi avtomatlaşdırılmış informasiya sistemi (AVİNSİ) layihələşdiricisini bir konseptual verilənlər bazası (VB) modeli ilə və ya VB-ni təsvir edən bir-neçə lokal modellərlə təmin etmək üçün lazımdır. Verilənlərin modelləşdiril-məsi üçün ən geniş istifadə edilən vasitə *ERD* - "mahiyyət-əlaqə" diaqramıdır ki, bu da 1976-cı ildə Piter Çen tərəfindən təklif edilmişdir.

ERD-nin təməl anlayışlarına: (*mahiyyətlər*) və onların xassələri (*atributları*) və mahiyyətlərarası münasibətlər (*əlaqələr*) aiddir. Bundan əlavə, *ERD*-də: əlaqənin tipi və gücü, ilkin və xarici açarlar, indekslər, diaqramların normallaşdırılması və s. kimi anlayışlardan da istifadə edilir.

Obyekt yönümlü metodologiya.

Sistemlərin layihələşdirilməsinə obyekt yönümlü yanaşma 1990-cı illərdən istifadə

edilməyə başlamışdır. Bu yanaşma imkanların genişləndirilməsi kimi iterativ prosesin nəticəsi olaraq meydana gəlmişdir. Vahid model təhlil, layihələşdirmə və reallaşdırma mərhələlərində konkretləşdirilir və uğurlu iterasiyaların nəticəsində yeni detallarla (elementlərlə) zənginləşdirilir. Zəruri olan hallarda dəyişikliklər və təkmilləşdirmələr edilir.

Obyekt yönümlü yanaşma aşağıdakı üstünlüklərə malikdir:

-obyekt yanaşmasından istifadə edilərkən layihələrin unifikasiya səviyyəsi yüksəlir ki, bu da təkrar istifadəyə imkan verir və nəticədə modelləri müəyyən tərkib hissələrdən yığmaq imkanı yaranır.

-obyekt dekompozisiyası mürəkkəb modellər yaratmaq zərurətini aradan qaldırır, çünki bu yanaşma nisbbətən kiçik alt sistemlər bazasında modelin inkişafının təkamül yolunu nəzərdə tutur.

-obyekt modeli təbbidir, çünki bu, dünyanın insani dərkinə yönəldilmişdir.

Obyekt yönümlü yanaşmanın nöqsanları aşağıdakılardan ibarətdir:

-ilkin xərclər yüksəkdir. Bu yanaşmada cəld fondverimi almaq mümkün deyildir. Bunun tətbiqindən alınan effekt bundan sonra gələn 2-3 layihənin hazırlanmasından sonra və təkrar istifadə edilə bilən komponentlərin hesabına əldə edilir.

-layihənin idarə edilməsini həyata keçirmək mürəkkəbdir. Menecerlər işlərin dəqiq dekompozisiyası vasitəsilə əsas mərhələlər və komplektləşdirici elementlər göndərmələrinin hərəkət səviyyəsini ölçə bilirlər. Obyekt layihələşdirməsində "xırdalaşdırma-detallaşdırma" nəticəsində mərhələlər arasında dəqiq sərhədlər aradan qalxır və layihə sənədləşməsi davamlı surətdə kəsilmədən inkişaf edir. Buna görə də layihəni xırda modullara bölmək məqsədəuyğun sayılır. Çünki həmin modulların yaradılmış variantını tətbiq etməklə işi xeyli sürətləndirmək olur.

-layihələrin mürəkkəbliyinin artması PT-nin müşayiət üçün uyuşanlığına və miqyaslılığına təsir göstərir. Obyekt yönümlü yanaşmanın konseptual təməlini obyekt modeli təşkil edir.

Bu model aşağıdakı prinsiplər nəzərə alınmaqla qurulur:

-*abstraktlaşdırma*. Bu, müəyyən obyektin digər obyektlərdən fərqini ifadə edən

daha vacib xarakteristikalarının ayrılmasıdır. Bu, az vacib elementlərin təhlilə cəlb edilməməsi üçündür.

-*inkapsulyasiya*. Bu, ümuməlçatan interfeyslə gizlədilən xassələrin fiziki lokallaşdırılmasıdır.

-*modulluq*. Bu, dekompozisiya prosesində sistemin sıx əlaqəli hissələrinin zəif əlaqəli hissələrindən ayrılmasıdır. Belə ki, sıx əlaqəli hissələrin hər biri bir alt sistem kimi modullaşdırılır. Seyrək əlaqələr alt sistemlər arası əlaqələrdir.

-*iyerarxiya*. Bu, abstraksiyalar sisteminin səviyyələr üzrə düzülməsi və ya çeşidlənməsidir. Obyekt yönümlü yanaşmanın əsas anlayışları obyekt və ssinifdir. Obyekt dedikdə, dəqiq təyin edilmiş davranışı və vəziyyəti, həmçinin davranış fərdiliyi olan predmet və ya hadisə nəzərdə tutulur. Davranış və struktur oxşarlığı olan obyektlər bir sinif təşkil edir. Yəni sinif dedikdə, strukturu və davranışı oxşar olan obyektlər çoxluğu başa düşülür. Obyekt yönümlü yanaşmnın digər mühüm anlayışları varislik və polimorfizmdir. Polimorfizm anlayışı sinfin iki və daha çox tipə aid olmanı ifadə edir. Varislik anlayışı yeni sinfin mövcud siniflərə əlavələr edilməsi və ya verilənlərin və ya metodların yenidən mənalandırılması yolu ilə qurula bilməsini ifadə edir. Bir sinfə aid olan obyektlər eyni əməliyyatlara və eyni atributlara malik olurlar, lakin hər bir obyektə aid eyniadlı atributlar fərqli qiymətlərə malik olurlar. Obyektin vəziyyəti onun atributlarının qiymətləri ilə təyin edilir. Obyektin davranışı isə onun əməliyyatları ilə təyin edilir. Obyektin atributlarının qiymətlərini dəyişə bilən əməliyyatların icrası obyekti bir vəziyyətdən digər vəziyyətə keçirə bilir. Sistemin funksiyası həyata keçirilərkən obyektlər arasında məlumat mübadiləsinə imkan yaradan əlaqələr qurulur.

7. Unifikasiya edilmiş modelləşdirmə dili (*UML*).

Unifikasiya edilmiş obyekt yönümlü modelləşdirmə dili olan *UML*[12] yuxarıda bəhs edilən yanaşmlar arasında kompromis yaradılması zərurətindən meydana çıxmışdır. İNSİ-lərin həyat tsikllərini *UML*-in köməyi ilə dəstəkləyən kifayət qədər çoxsaylı instrumental vasitələr mövcuddur. *UML* həm də müxtəlif layihələşdirici qruplarının

[12] *UML - Unified Modeling Language*

(komandalarının) fəaliyyətini dəstəkləyən və sazlayan kifayət qədər çevik dildir. İNTEX-lərin bu istiqamətinin inkişafına güclü təkan verən obyekt yönümlü proqramlaşdırma dilləri 1980-ci illərin sonu, 1990-cı illərin əvvəllərində geniş yayılmağa başlamışdır. İstifadəçilər istəyirdilər ki, obyekt yönümlü yanaşmanın bütün gücünü özündə birləşdirən vahid modelləşdirmə dilinə malik olsunlar. Çünki bu dil həmin yanaşmanın bütün tərəflərini əks etdirən dəqiq sistem modeli kimi təsəvvür edilirdi. 90-cı illərin ortalarında bu sahədə görkəmli liderlər olaraq *Booch*[13], *OMT-2*[14] və *OOSE - Object-Oriented Software Engineering*[15] metodlarını nümunə gətirmək olar. Lakin bu metodların hər üçü müəyyən üstün və nöqsan cəhətlərə malik idi. Belə ki, *OOSE* problemin təhlili mərhələsində sistemə qarşı qoyulmuş tələblərin təhlili üçün ən yaxşı metoddursa, *OMT-2* İNSİ-nin layihələşdirilməsi mərhələsinin təhlilində ən yaxşı metoddur. *Booch* isə layihələşdirmə və dizayn üçün daha münasib metoddur. Vahid modelləşdirmə dili *UML* məhz bu metodların üstün cəhətlərinin birəşdirilməsi nəticəsi kimi müydana çıxmışdır. *UML* dili 1994-cü ilin oktyabrında yaradılmağa başladı və 1995-ci ilin sonunda hazır oldu.

Obyekt yönümlü modelləşdirmə dili olan *UML* aşağıdakı əsas xarakteristikalara malikdir:

-vizual modelləşdirmə dilidir. İNSİ layihələşdiricisi ilə sifarişçi arasında, həmçinin layihələşdirici qruplar arasında interfeys yaradan model kimi işləyir.

-dilin baza konsepsiyalarını genişləndirmək və ixtisaslaşdırmaq imkanı vardır.

UML proqram sistemlərinin vizual modelləşddirilməsi üçün standart notasiya[16] dilidir. Bu standart 1997-ci ilin payızında qəbul edilmiş və bu gün də *CASE*-məhsullarda rəhbər tutulur.

UML dilinin istifadəçilərinə aşağıdakı imkanlar təklif edilir:

-əksər tipik proqram əlavələrinin genişləndirmə mexanizmlərindən istifadə etməməklə dilin nüvəsində (daxilində) olan vasitələrə əsasən model qurmaq;

---

[13] *Grady Booch*
[14] *Jim Rumbaugh*
[15] *Ivar Jacobson*
[16] şərti yazılı işarələr sistemi

-lazım gəldikdə nüvəyə yeni elementlər daxil etmək.

UML dilinin əsas obyektlərinin semantikası.

UML – mürəkkəb proqramlar sistemlərinin modelini yaratmağa imkan verən diaqramlar yığımıdır. *UML* diaqramları 2 qrupa bölünür:

1) Struktur diaqramları:

- *Implementation Diagram* (reallaşdırma diaqramları):
    - o *Deployment diagram* (topologiya diaqramları);
    - o *Component diagram* (component diaqramları).
- *Class diagram* (siniflər diaqramları);

2) Davranış diaqramları:

- *Use case diagram* (istifadə variantları diaqramları);
- *Statechart diagram* (vəziyyətlər diaqramları);
    - o *Activity diagram* (fəallıq diaqramları);
- *Interaction diagram* (qarşılıqlı fəaliyyət diaqramları);
    - o *Sequence diagram* (ardıcıllıq diaqramları);
    - o *Collaboration diagram* (əməkdaşlıq diaqramları və ya korporativ diaqramlar);

8. İstifadə variantları (nümunələr) diaqramları (*Use Case Diagram*)

İstifadə variantı, ilk növbədə, fəaliyyətdə olan şəxs adlanan istifadəçinin proqram əlavəsinin adına və tipinə görə müəyyyən edilir.

İstifadə variantı fəaliyyətdə olan şəxslə proqram əlavəsi arasında mövcud olan tipik qarşılıqlı təsirdən ibarətdir. Məsələn, “faylı aç” əmri fəaliyyətdə olan şəxs qismində götürülən istifadəçi ilə mətn redaktoru arasında tipik istifadə variantı hesab edilir. Eyni bir adam və ya xarici sistem sistemdən müxtəlif rollarda çıxış edən fəaliyyətdə olan şəxs (aktyor) qismində, müxtəlif üsullarla istifadə edə bilər.

Aktyorlar yalnız rol oynayırlar. Onlar konkret insanı və ya mahiyyəti təmsil

etmirlər. Nümunə bir və ya bir-neçə aktyor üçün əhəmiyyətli olan nəticənin alınmasını nümayiş etdirən davranışı təsvir edir. Nümunəni xırdalamaq üçün spesifikasiya[17] tərtib edilir. İstifadə variantı diaqramları nümunələr arasındakı ümumiləşdirmə, daxiletmə və genişləndirmə kimi spesifik qarşılıqlı münasibətləri göstərə bilir. Nümunələrin ümumiləşdirilməsi dedikdə, bir və daha çox nümunə üçün ümumi olan davranışı valideyn nümunə kimi qəbul etmək başa düşülür.

İstifadə variantı diaqramları qurularkən əvvəla, iki istifadə variantını ardıcıl birləşdirmək olmaz, çünki bu diaqram variantların icra ardıcıllığını[18] deyil, variantın özünü göstərmək üçündür. İkincisi, hər bir istifadə variantı fəaliyyətdə olan şəxs tərəfindən təşəbbüsləndirilməlidir. Lakin bəzən diaqramlarda nümunələrin ardıcıl birləşdirilməsi halları rast gəlir. Bu halda aşağıdakı xüsusi əlaqə tiplərindən istifadə edilir:

1) nümunələr arasında "*include*" münasibəti bir nümunənin davranışını digər nümunənin axarına qoşur. Məsələn, "UMM-dəki İndeksli Kartoçkalara baxış" adlı nümunə icra edilərkən hökmən "UMM axtarışı" nümunəsi icra edilməlidir. Bu münasibəti *UML* dilində işarələmək üçün punktirli oxdan istifadə edilir. Bu halda oxun istiqaməti mənbə nümunədən hədəf nümunəyə, yəni, "UMM-dəki İndeksli Kartoçkalara baxış" adlı nümunədən "UMM axtarışı" adlı nümunəyə yönəlməlidir.

2) "*extend*" münasibəti mövcud nümunəyə yeni davranış daxil etmək üsuludur. "*extend*" münasibətindən istifadə edilərkən baza nümunə üzərində əməl aparılacaq nümunə (məsələn, ləğvetmə) üçün model karkası rolunda çıxış edir. "*extend*" münasibəti *UML* dilində punktirli oxla işarə edilir. Oxun istiqaməti hədəf nümunədən mənbə nümunəyə yönəlmiş olur. Bu halda mənbə nümunə (istifadə variantı) hər cür dəyişiklik (genişləndirmə) tətbiqinə imkan verən sərbəst nümunə olmalıdır. "Daxiletmə", "genişləndirmə" və "ümumiləşdirmə" kimi əlavə imkanlar sözü gedən diaqramı sadələşdirib sifarişçi üçün anlaşıqlı etməkdən ötrüdür.

İstifadə variantlarını təhlil etməklə layihələşdirici aşağıdakıları həyata keçirir:

-istifadə variantı hadisələri axınlarının reallaşmasında iştirak edən sinifləri identi-

[17] spesifikasiya (bir şeyin spesifik xüsusiyyətlərinin müəyyən edilməsi, bir şeyin dəqiqləşdirilmiş təsnifi).
[18] Variantların icra ardıcıllığını fəaliyyət diaqramları əks etdirir.

fikasiya edir;

-siniflər arasında reallaşdırılan istifadə variantları ilə həyata keçirilən davranışı paylayır, başqa sözlə, siniflərin vəzifələrini təyin edir;

-təhlil edilən sinifləri unifikasiya edir.

Nümunələr diaqramları aktyorların sistemlə niyə (hansı məqsədlə) qarşılıqlı təsirdə olduğunu təyin edir. Nümunələr və onların təsviri tələbatların təhlili mərhələsində ən mühüm faktdır. Əgər yalnız "Sistemlə qarşılıqlı təsir nə üçün baş verir?" sualına cavab verilməli olsaydı nümunələr diaqramı ilə kifayətlənmək olardı. Lakin burada həm də "Qarşılıqlı təsir necə baş verir?" sualına cavab vermək lazım gəlir. Buna görə də qarşılıqlı təsir (*sequence və collaboration*) və fəaliyyət (*activity*) diaqramlarına da diqqət yetirilməlidir.

Qarşılıqlı təsir diaqramları *(Interaction Diagram)*

Aktyorların bütövlükdə sistemlə və ya sistemin ayrı-ayrı modullarının (siniflərin) öz aralarında qarşılıqlı təsirdə olması xəbər mübadiləsi və ya sorğular vasitəsilə baş verir. Bu cür qarşılıqlı təsirləri modelləşdirmək üçün *UML*-in tərkibində iki diaqram[19] mövcuddur ki, bunlar da yaradılan sistemdəki obyektlərin qarşılıqlı təsirdə olmasına müxtəlif nöqteyi-nəzərdən baxmaq imkanı verir. Bir qayda olaraq, qarşılıqlı təsir diaqramları bir konkret istifadə variantı çərçivəsində obyektlərin davranışlarını əhatə edir.

Ardıcıllıqlar diaqramları (*Sequence Diagram*)

Ardıcıllıqlar diaqramlarını qurmaq üçün aşağıdakılar zəruridir:

-hər bir fəaliyyətdə olan şəxsi (obyekti) identifikasia etmək (adlandırmaq) və onun üçün yaşam xətti yaratmaq (təsvir etmək)[20].

-istifadə variantının təsvirinə əsasən sistem xəbərləri çoxluğunu və onların ardıcıl-

[19] Əməliyyatlar ardıcıllığı diaqramı (*Sequence Diagram*) və Əməkdaşlıq diaqramı (*Collaboration Diagram*)

[20] Diaqramın sol kənarında qarşılıqlı təsirin təşəbbüskarı olan obyekt təsvir edilir. Nisbətən sağda təşəbbüskar obyektlə qarşılıqlı təsirdə olan obyekt təsvir edilir. Bu sıra bu qayda ilə davam etdirilir.

lığını təyin etmək[21].

Ardıcıllıq diaqramında yalnız qarşılıqlı təsirdə olan obyektlər göstərilir. Bu diaqramlarda digər obyektlərlə mümkün olan statistik assosiasiyalar göstərilmir.

Obyektin yaşam xətti (*object lifeline*) obyektin sistemdə mövcud olduğu müddəti göstərmək üçündür. Buna görə də bu xətt obyektin bütün qarşılıqlı fəaliyyətlərində potensial surətdə iştirak edə bilər.

Diaqramda yeganə obyektlə assosialanmış yaşam xətti şaquli punktir xətlə göstərilir. Əgər obyekt sistemdə daimi mövcuddursa, onda onun yaşam xətti diaqramın bütün müstəvisi boyunca ən yuxarıdan ən aşağıyadək davam edir.

Əgər sistemin resurslarını azad etmək üçün hansısa momentdə obyektlər dağıdılırsa, onda yaşam xətləri məhvetmə momentində qırılır. Bu moment "X" kimi işarə edilir.

Ardıcıllıq diaqramında obyektlər iki vəziyyətdə ola bilirlər: aktiv və passiv. Aktiv vəziyyətdə obyekt hər hansı əməli icra edir, passiv vəziyyətdə isə digər obyektlərdən xəbər gözləyir. Obyektlərin aktivliyini xüsusi ayrıntı şəklində göstərmək üçün idarəetmə fokusu (*focus of control*) tətbiq edilir. İdarəetmə fokusu dartılmış dar düzbucaqlı formasında təsvir edilir. Bu düzbucaqlının yuxarı tərəfi obyektin idarəetmə fokusu almasını (aktivləşmənin başlanmasını), aşağı tərəfi fokusun sona çatdığını (aktivliyin bitdiyini) göstərir. Obyektlər arasındakı hər biq qarşılıqlı təsir onların bir-biri ilə mübadilə etdikləri xəbərlər çoxluğu ilə təsvir edilir. Bu xəbərlər bir obyektin təsiri ilə digər obyektin hərəkətə gətirilməsi ilə nəticələnir.

Xəbər (message) bir obyektdən digərinə ötürülən bitkin informasiya fraqmentidir. Xəbər qəbul edən obyekt qoyulmuş məsələnin həllinə yönəldilmiş əməllər ardıcıllığı şəklində həmin xəbərə reaksiya verməlidir. Xəbərlərin sırası şaquli istiqamətdə göstərilir. Müxtəlif oxlarla təsvir edilən xəbər tipləri mövcuddur:

- *Simple (1)* – xəbərin sadə ötürülüşü;
- *Synchronous (2)* – bu əməliyyat o halda icra edilir ki, müştəri xəbər göndərir,

[21] Sistem xəbərləri oxlarla təsvir edilir. Oxlar fəaliyyətdə olan şəxslə sistemin yaşam xətləri arasında yerləşdirilir. Sonra xəbərlərin adları və ötürülən qiymətlərin siyahısı göstərilir.

server isə həmin xəbəri qəbul edə bilir;

- *Özünügöndərmə (3)* – obyekt yönümlü proqramlaşdırmada (ООП)[22] geniş yayılmış hadisədir ki, bu halda obyekt özünün bir qayda olaraq qorunan metodunu çağırır;
- *Balking (4)* – bu əməliyyat o halda icra edilir ki, server xəbəri cəld qəbul etməyə hazır olur, əgər server xəbəri qəbul etməyə hzır deyilsə, müştəri xəbər göndərmir;
- *Timeout (5)* - əgər müəyyən müddət ərzində server xəbəri qqəbul edə bilmirsə, müştəri xəbər göndərməkdən imtina edir;
- *Procedure Call (6)* – müştəri server prosedurunu çağırıb onu idarəetməyə ötürür;
- *Asynchronous (7)* – müştəri xəbəri ötürür və serverin cavabını gözləmədən özünün proqram kodunu icra edir;
- *Return (8)* – prosedurdan qayıdış zamanı nə baş verdiyini təyin edir.

Xəbər göndərişlərinin tezliyinin 2 variantı mümkündür:

Periodic – müştəridən daxil olan xəbərlər məlum dövrülüklə daxil olur;

Aperiodic – müştəridən daxil olan xəbərlər qeyri-müntəzəm daxil olur.

Xəbərin adı aşağıdakı tələblərə cavab verən olmalıdır:

-informativliyi artırmaq üçün ad *add* (əlavə et), *enter* (daxil et), *end* (sona çatır), *load* (yüklə) və s. feillərindən biri ilə başlamalıdır.

Təhlil mərhələsində əvvəlcə, qara qutu modeli tətbiq edilərək ardıcıllıq diaqramında sistemə bir obyekt kimi baxılır. Bu halda diaqramda təsvir edilən əməliyyatlar sistem əməliyyatları adlanır. Bu əməliyyatlar sistemin açıq interfeysini təşkil etməklə, sistemə giriş xəbərləri emal etmək üçündür. Bu xəbərləri qəbul edərkən sistem qara qutu vəziyyətində olur. Sonra sistem əməliyyatlarının təsvirini (onların ilkin və son şərtlərini) dəqiqləşdirmək üçün aktivliklər diaqramından istifadə edilir.

UML dilinin digər bütün diaqramları kimi, ardıcıllıqlar diaqramı da iki xırdalaşdırma (detallaşdırma) mərhələsindən keçir:

[22] ООП – объектно-ориентированное программирование

1) tələblərin yığılması və təhlili. Bu mərhələdə diaqramın eskiz versiyası hazırlanır ki, bu versiyada da yalnız əsas obyektlər və xəbərlər əks etdirilir. Əsas obyektlər və xəbərlər dedikdə, sifarişçinin istədiyi bu və ya digər nümunənin reallaşdırılması prosesində sistemdə baş verən proseslərlə sifarişçiləri tanış etməkdən ibarətdir;

2) layihələşdirmə. Bu mərhələdə bütün diaqramlar proqramçıların işini sadələşdirmək üçün bütün xırdalıqlarla təchiz edilir. Layihələşdirmə mərhələsində həm də ardıcıllıqlar diaqramında əməliyyatlar siqnatura ilə (ad və parametrlərlə) təchiz edilir. Diaqramın təsvirində ilkin və son şərtləri də əhatə edən spesifikasiyalar göstərilir.

Əməkdaşlıq diaqramı (*Collaboration Diagram*) qarşılıqlı fəaliyyət diaqramından onunla fərqlənir ki, əməkdaşlıq diaqramı diqqəti xəbərlərin ötürmə ardıcıllığına yönəldir. Bu, düyünləri obyektlərdən ibarət olan qraf formasındadır və diaqramın istənilən yerində yerləşdirilə bilir. Xəbər mübadiləsinin nümayişində zaman şkalası iştirak etmədiyindən, bu diaqram yığcam olur ki, bu da bütün obyektlərin qarşılıqlı fəaliyyətinin təhlilini asanlaşdırır. Lakin ardıcıllıqlar diaqramı xəbərlərin zaman sırasını dəqiq təsvir edir, buna görə də çox yer tutur. Əməkdaşlıq diaqramında xəbərlərin ötürülmə ardıcıllığı sıra nömrələri ilə təsvir edilir.

İstifadə edilən mümkün əlaqə tipləri aşağıdakılardır:

-*Link To Self* - obyektin özünün özü ilə əlaqəsini, yəni qısa əks-əlaqəni göstərir.

-*Link Message / Reverse Link Message* (xəbər ötürülməsi / əks ötürmələr) – bu əlaqə xəbərin hökmən düz (əks) istiqamətdə ötürülməsini ifadə edir.

-*Data Flow / Reverse Data Flow* (verilənlər axını) – bu əlaqə bir obyektdən digər obyektə düz və əks istiqamətdə verilənlər axınlarını əks etdirir.

Əməkdaşlıq diaqramının qarşılıqlı fəaliyyət diaqramından əsas fərqi daha geniş sintaksis konstruksiyalara malik olmasıdır.

Fəaliyyət diaqramları *(Activity Diagram)*

Bu diaqramlar sistem, istehsal və ya texnoloji prosesdə baş verən əməliyyatlar axınlarını təsvir etməyə imkan verir. Fəaliyyət diaqramları nümunələri aktyorla sistemin qarşılıqlı fəaliyyətini təsvir edən fəaliyyətlər ardıcıllığı (*activitiy*) şəklində modelləş-

dirmək üçün istifadə edilir. Bu diaqramlar əsas diqqəti icra edilən əməliyyat üçün məsuliyyət daşıyanın üzərində cəmləşdirir.

Fəaliyyətin 2 tipi vardır:

action – fəaliyyət, iş, hərəkət.

send event – hadisə göndərişi.

Fəaliyyət (*action*) öz növbəsində, ada (*name*) malikdir və girişdə (*on Entry*), çıxışda (*on Exit*), vaxtında (*Do*) və hadisə meydana çıxdıqda (*on Event*) icra edilir. Məsələn, "Diskin hazırlanması" işinin semantikasına girişdə "Formatlaşdırma" daxil olur. Hadisə üzrə (*on Event*) hərəkət adlandırılarkən hadisəyə ad verilir, arqumentlər və şərtlər daxil edilir. Məsələn, "Texnoloji proses" işinin semantikasına "*time*" arqumentinin "*time=0*" şərtində "Proses başa çatmışdır" hadisəsi baş verdikdə "zəng" hərəkəti daxildir.

Əgər fəaliyyət hadisə göndərişi (*send event*) tipindədrisə, onda həmin fəaliyyətin inisializə edilməsi[23] momentini (*on Entry, on Exit, Do, on Event*) göstərməklə yanaşı, həm də göndərilən arqumentlər (*Send arguments*) və növbəti fəaliyyət (*Send target*) göstərilə bilər. Məsələn, "Texnoloji proses" fəaliyyətinin "Zəng" hərəkəti "*time*" arqumentli və "*time=0*" şərtli "Proses başa çatmışdır" hadisəsi baş verərkən icra edilir və "status" və "tam vaxt" arqumentli "Prosesin başa çatması" fəaliyyəti barədə hadisə göndərişi ilə bitir.

Fəaliyyət diaqramı özünün fəaliyyətlərindən (*activitiy*) əlavə vəziyyət (*State*), sinxronlaşdırma (*Synchronizations*), həll düyünü (qovuşdurma - *decision*) və bölmələr (*swimlanes* – üzgüçülük yolları) kimi fəaliyyətləri də əhatə edə bilir.

Vəziyyət (*State*) – obyektin həyatında mövcud olan elə situasiyadır ki, mövcud olduğu müddət ərzində obyektin müəyyən işlər görməsinə şərait yaradır və ya müəyyən hadisələrin baş verməsini gözləməyi təmin edir. Fəaliyyət kimi, vəziyyət də 2 tipə malikdir:

-*action* – verilmiş işin icrası vəziyyəti;

-*send event* – bir vəziyyətdən digərinə keçid məqamında analoji sazlama hadisə-

[23] adlandırılması

lərinin ötürülməsi vəziyyəti.

Obyektin bir fəaliyyət növündən (vəziyyətdən) digərinə keçməsi (*transition*) ox vasitəsilə təsvir edilir.

Keçidin spesifikasiyası aşağıdakıları əhatə edir:

-hadisə (*event*) – bir fəaliyyət növündən digərinə keçidə səbəb olandır. Hadisənin adı vəziyyətlər arasındakı əlaqənin üzərində qeyd edilir.

-qoruyucu şərtlər (*guard conditions*) – keçidin həyata keçirilməsi üçün qoyulan şərtlərdir.Qoruyucu şərtlər hadisənin daından sonra keçid xətti boyunca kvadrat mötərizə daxilində göstərilir.

-əməl (*action*) – keçid prosesində icrası tələb olunandır və keçid xətti boyunca "/" işarəsindən sonra göstərilir.

-hadisə - məqsəd fəaliyyəti (*send target*) üçün arqumentlərlə (*send arguments*) birlikdə ötürüləndir (*send event*).

Fəaliyyət diaqramları tərkib-təşkiledici fəaliyyətləri (vəziyyətləri) də təsvir etməyə imkan verir. Fəaliyyət hər hansı bir hadisənin baş verməsi ilə və ya ilk fəaliyyətin inisiallaşdırılması ilə başlana bilər.

Sinxronlaşdırma (*Synchronizations*) düyünü axını bir-neçə paralel axına parçalamaq və ya, əksinə, bir-neçə giriş axınını bir çıxış axını şəklində birləşdirmək üçündür.

Həll düyünü (*decision*) komentariya (izah) şəklində olan keçid şərti ilə müşayiət olunmalıdır. Bu zaman hər bir axın şaxəsi qoruyucu şərtlə (*guard condition*) təchiz edilməlidir. Bir-neçə axın birləşdirilərkən bu şərt həll düyününü qovuşma düyününə çevirir.

Bölmələr (*swimlanes*-üzgüçülük yolları) fəaliyyəti presedentlər[24] üzrə qruplaşdırmaq yolu ilə siniflərə, komponentlərə, rollara və s. parçalamaq üçündür.

Tarixi vəziyyət ("H"-la işarə edilir) - əvvəlki aktiv vəziyyəti kompozisiya vəziyyətinə bərpa edən psevdovəziyyətdir. Bu, *StopState* və ya *StartState* kimi tərkib-təşkiledici vəziyyətlərdən birində ola bilən *OpenState* kompozisiya vəziyyətinin hansı

[24] presedent; misal (keçmişdə olmuş və sonra olan eyni tipli hadisələr üçün misal ola biləcək hadisə və ya hərəkət)

tərkib-təşkiledici vəziyyətlə bağlı olduğunu yadda saxlamağa imkan verir. Yəni, *OpenState* vəziyyətindən çıxma məqamında həmin tərkib-təşkiledici vəziyyətlərdən hansının cari olduğu yadda saxlanır. Nəticədə OpenState vəziyyətinə ixtiyari keçid zamanı başlanğıc vəziyyətə deyil, məhz həmin tərkib-təşkiledici vəziyyətə qayıdış baş verir.

Vəziyyətlər diaqramları (*Statechart Diagram*) fəaliyyət diaqramlarının xüsusi halıdır və obyektin vəzziyyətlərini və keçid şərtlərini təsvir etmək üçündür. Bütün bunlar isə bir küll halında obyektin yaşam tsikli ərzindəki davranışını xarakterizə edir. Vəziyyətlər diaqramı, əslində, hər hansı avtomatı təsvir edən xüsusi qraf növüdür. Bu qrafın təpələri vəziyyətləri, qövsləri isə vəziyyətdən-vəziyyətə keçidi əks etdirir.

Qarşılıqlı fəaliyyət diaqramları (*sequence u collaboration*) və həmçinin fəaliyyət diaqramları (*activity*) təhlil və layihələşddirmə mərhələlərində nəticələrin sənədləşdirilməsi üçün geniş istifadə edilir. Lakin bu diaqramların əsas təyinatı predmet oblastının (texnoloji prosesin) təhlili üçündür. Predmet oblastının modeli gələcək sistemin təməlini təşkil edir. Buna görə də təhlil mərhələsinin əsas vəzifəsi predmet oblastının anlayışlarını identifikasiya etməkdən və nəticələri predmet oblastının modeli kimi təqdim etməkdən ibarətdir. Layihələşdirmənin sonunda bu model koda çevrilir. Bununla əlaqədar olaraq predmet oblastının modelinin qurulması məsələləri xüsusi diqqət tələb edir.

Siniflər diaqramı *(Class Diagram)*

Predmet oblastının modeli layihələşdirici mövqeyindən predmet oblastı terminlərində əsas anlayışlar siniflərini (konseptual sinifləri) əks etdirir. *UML* ilində predmet oblastının modeli siniflər diaqramı şəklində təqdim edilir. Həmin diaqramlarda proqram komponentləri deyil, gerçək dünyanın anlayışlar sinifləri təsvir edilir. Konseptual siniflərdən əlavə, predmet oblastının modeli digər siniflərlə konseptual siniflər arasındakı assosiasiyaları da əks etdirir. Lakin siniflər diaqramındakı əməliyyatlarda predmet oblastının modeli təsvir edilmir.

Predmet oblastının modeli iterativ qaydada, bir-neçə inkişaf fazası iterasiyaları ardıcıllığı ilə qurulur. Hər bir növbəti iterasiyada modelə yeni konseptual siniflər əlavə

edilir və ya mövcud siniflər növbəti istifadə variantı təhlil edilərkən dəqiqləşdirilir.

Konseptual siniflərin identifikasiya edilməsi üçün ən sadə priyom predmet oblastının mətn formalı təsvirindəki isimlərin seçilməsi və onların konseptual siniflərə və ya atributlara namizəd kimi istifadə edilməsidir.

Diaqramda siniflər 3 zonaya bölünmüş düzbucaqlı şəklində təsvir edilir. 1-ci zonada sinfin adı, 2-cidə atributlar, 3-cüdə əməliyyatlar təsvir edilir. Əməliyyat funksiyanın formasını və ya prosedurunu (siqnaturasnı[25]) təyin edir. Əməliyyatın konkret reallaşdırılması metod adlanır. Sinifin adı təklik halında olan isim olur. Obyektlər birləşdirici rabitələr vasitəsilə məlumat mübadiləsi edirlər. Obyektlər arasında rabitənin olması üçün həmin obyektlərin sinifləri arasında assosiasiya mövcud olmalıdır. Çünki obyektlərarası rabitə siniflər arası assosiasiya nüsxəsidir. Assosiasiya – siniflər arasında müəyyən əhəmiyyətli münasibəti əks etdirən rabitədir. Assosiasiyaların təhlili prosesində minimallıq prinsipi rəhbər tutulmalıdır. Çünki siniflərin sayı *N* olarsa, siniflərarası əlaqələrin mümkün sayı *N(N-1)* olacaqdır. Buna görə də modelə yalnız elə assosiasiyalar daxil edilir ki, onların haqqındakı biliyin müəyyən vaxt ərzində saxlanması zəruridir. Həm assosiasiyaya, həm də assosiasiyanın hər iki ucundakı siniflərə ad verilir. Bundan əlavə, obyekt sinifləri bir assosiasiya nüsxəsi şəklində əlaqələndikdə onların oynadığı rollara da ad verilir. Obyektlər arasındakı əlaqəni təsvir edən ox informasiya ötürülüşünün istiqamətini göstərir. Məsələn, fərz edək ki, şirkət *n* sayda işçini işə götürür. Çoxsaylılığa (*multiplicity*) qoyulmuş "*1*" və "*n*" məhdudiyyətləri göstərir ki, *Person* obyektləri verilmiş vaxt momentində yalnız bir *Company* obyekti tərəfindən işə götürülə bilər və *Person* işsiz ola bilməz. *Company* obyekti *Person* obyektlərinə xəbər göndərə bilər, lakin əks istiqamətli xəbər göndərilişi mümkün deyildir. Düzgün qiymətləndirilməyən çoxsaylılıq proqram əlavəsinin çevikliyini məhdudlaşdırır. Məsələn, şəxsi menecerlər bir işçi üçün bir-neçə telefon qorumağa (saxlamağa) icazə vermirlər. Assosiasiyanın xüsusi halı assosiasya-sinif (*Association Class)* adlanır və həm sinfin, həm də assosiasiyanın xassəsi kimi təzahür edir. Assosiasya-sinif – assosiasiyaya aid

[25] ж xüs. siqnatura (1. reseptdə dərmanların işlədilmə qaydaları haqqında göstəriş; 2. dərman şüşəsi və ya qutusuna yapışdırılan reseptin surəti; 3. mətb. kitabı düzgün səhifələmək üçün hər çap vərəqinin birinci və üçüncü səhifəsinin alt tərəfində qoyulan rəqəm).

olan atributların və əməliyyatların saxlandığı məkandır. Özü-özü ilə assosiasya halında, yəni refleksiv[26] assosiasyada assosiasiya qütblərinə ad verilməsi məcburidir. Refleksiv assosiasiya verilmiş sinfə aid obyektlərin həmin sinfə aid digər obyektlərlə əlaqəli olduğunu ifadə edir. Məsələn, hər bir *Directory*[27] obyekti sayı *0*-dan *n*-dək olan və *subdirectory*[28] rolunda çıxış edən digər *Directory* obyektləri ilə əlaqəli ola bilir. Digər tərəfdən, alt kataloqun 1 valideyn ktaloqu ol bilər və ya ümumiyyətlə, valideyn kataloqu olmaya bilətr. Bundan əlavə, *Directory* ixtiyari sayda fayl obyektləri ilə əlaqəli ola bilər. *UML* dilində "çoxun-çoxa" tipli assosiasiyalar geniş istifadə edildiyindən, xüsusi assosiasiya sinifləri nəzərdə tutulmuşdur. Adi siniflərə oxşar olaraq, assosiasiya sinifləri də əlaqələrə aid olan atributlara və əməliyyatlara malik olur ki, bunlar da heç bir digər iştirakçı obyektə verilə bilməz. Məsələn, *AccessibleBy*[29] sinfinin *accessPermission*[30] atributu eyni zamanda həm fayla, həm də istifadəçiyə aiddir və heç bir informasiya itkisinə yol vermədən bunlardan yalnız birinə təhkim edilə bilməz.

Assosiasiyanın *(association)* konkretləşdirilmiş formaları aşağıdakılardan ibarətdir:

-asılılıq (*dependency*) – bir sinfin digər sinifdə edilmiş təyindən asılı olduğunu göstərən biristiqamətli əlaqə. Bu əlaqə o halda yaranır ki, bir sinif digərini əməliyyat parametri kimi istifadə edir. İlkin sinif üçün kodun generasiya edilməsi (yüklənməsi) zamanı məqsəd sinfinin atributları əlavə edilmir, lakin koda *"#include"* tipli operator əlavə edilir.

-aqreqasiya (*aggregation*) – assossiasiyanın xüsusi halı olub, tam-hissə münasibətini[31] ifadə edir. "Əgər A B-nin hissəsi, B isə C-nin hissəsidirsə, A C-nin hissəsidir" halında aqreqasiya tranzitivdir – deyirlər. Aqreqasiya romb şəklində təsvir edilir. Romb aqreqat olan qütbə yaxın yerləşdirilir. Hissə-obyektin "yaşam tsikli" tam-obyektin "yaşam tsikli" ilə üst-üstə düşür. Aqreqasiyanın nisbətən güclü növü "kompozisiya"[32]

---

[26] Refleksiv (reflektiv); qeyri-ixtiyari, qeyri-şüuri, qeyri-iradi.
[27] Siyahı
[28] Altsiyahı
[29] əlçatandır
[30] Müraciətə icazə vardır
[31] məsələn, avtomobil-mühərrik
[32] *composition* – birləşdirmə

tipli əlaqədir. Bu əlaqə assosiasiyaya əlavə 2 məhdudiyyət daxil edir: 1) təşkiledici yalnız bir aqreqata aid ola bilər; 2) təşkiledici aqreqatın ömür müddətinə sahib olur. Kompozisiya rənglənmiş rombla göstərilir.

Beləliklə, aqreqasiya – bir-neçə obyektin bir konteyner-obyekt halında qurulması üsuludur ki, bu zaman həmin obyektlər konteyner metodları ilə emal edilir.

Konteyner super siniflə alt sinif arasında varislik əlqəsi yaradır. Həm də alt sinif özünün üst sinfinə aid bütün atributlara, əməliyyatlara, münasibətlərə və məhdudiyyətlərə varis olur. Ümumiləşdirmə (aqreqasiya) çox vaxt mücərrəd siniflərlə birgə istifadə edilir. Mücərrəd siniflərin atributları və əməliyyatları da mücərrəd olur. Mücərrəd siniflər nüsxə yaratmır. Həddən artıq dərin sinif iyerarxiyası yaratmaq məqsədəuyğun deyildir. Adətən 3 səviyyəli iyerarxiyadan istifadə edilir. 5-6 səviyyəli iyerarxiya layihələşdirilən sistemin xarakterindən asılı olaraq məqsədəuyğun olub-olmaya bilər. Bundan əlavə, ümumiləşdirmə yalnız o halda istifadə edilməlidir ki, heç olmazsa bir alt sinfin atributları, əməliyyatları və ya assosiasiyaları supersinfə tətbiq olunan olmasın. Məsələn, super siniflə eyni siqnaturaya malik və eyni davranış nümayiş etdirən alt siniflərdə ümumiləşdirmə etmək olmaz.

Beləliklə, “ümumiləşdirmə” tipli assosiasiyalar polimorfizm təmin edir. Bu o deməkdir ki, yeni alt sinif daxil edilərkən super sinfin davanışı avtomatik olaraq yeni alt sinfə varis edilir və yeni alt sinif mövcud sinfin işini pozmur.

Bəzi proqramlaşdırma dillərində çoxluq varisliyi yaratmaq mümkündür. Bu halda bir sinif eyni zamanda bir-neçə sinifdən varislik qəbul edə bilir.

Çoxluq varisliyindən modeli yeni quruluşda yaratmaq yolu ilə imtina etmək olur. Atributun “görünənliyi” onun digər siniflər tərəfindən də istifadə edilə bilənliyini göstərir. Bir sinif digərini o halda “görür” ki, “görünən” “görənin” fəaliyyət sahəsinə daxil olmuş olsun və bunların arasında aşkar və ya gizli münasibət mövcud olsun. *UML* dilində “görünənliyin” 4 səviyyəsi təyin edilmişdir:

- *public* (ümumi) – verilmiş sinfi “görən” ixtiyari xarici sinif “görünən” sinfin ümumi xassələrindən istifadə edə bilir. Bu halda atributun və ya əməliyyatın qarşısında «+» işarəsi qoyulur;

- *protected* (qorunmuş) – verilmiş sinif və ya onun törəmələrinin yalnız qorunmuş xassələrindən istifadə edilir. Bu halda «#» işarəsindən istifadə edilir;

- *private* (bağlanmış) – verilmiş sinif yalnız bağlanmış atributları ilə istifadə edilə bilir. Bu halda «-» işarəsindən istifadə edilir;

- *package or implementation* (paketli) – atribut sinfin yerləşdiyi paket çərçivəsində ümumidir. Bu halda «~» işarəsindən istifadə edilir.

"Görünənlik" təyin edilərkən aşağıdakılar rəhbər tutulur:

1. Sinif yuxarıda sadalanan 4 səviyyə "görünənliyi" nümayiş etdirməlidir;
2. Çoxluq siniflərini təyin edən açıq metodlar bütövlükdə modelin uyuşanlığını poza bilər.

Sinif əməliyyatları və atributlarının mühüm xarakteristikalarından biri də verilmiş sinfə və ya obyektə aid olan təşkiledicinin (elementin) fəaliyyət (*scope*) sahəsidir. Obyektlərin müxtəlif qiymətlər ala bilən atributlarının özəl nüsxələri olur. Bu atributların və əməliyyatların nüsxələrinin (*instance*) hər birinin fəaliyyət sahələri ayrıca qiymətlə ifadə olunur. Bəzən bütün sinif obyektləri üçün yeganə qiyməti olan atributlardan istifadə edilir. Bəzən də heç bir konkret sinif nüsxəsinə aid olmayan əməliyyatlardan istifadə edilir. Məsələn, obyektin yaradılması əməliyyatı – konstruktor bu qəbildəndir. Belə atributlar və əməliyyatlar təsnifləşdirici (*classifier*) kimi işləyir. Bunların bütün nüsxələri birgə istifadə edilən ümumi xassəyə malik olmaqla, diaqramda altdan xətlə işarə edilir. Bunlar statik atributlar və əməliyyatlar hesab edilir.

Ümumilik xassəsi nümayiş etdirən siniflərin qruplaşdırılması üçün layihələşdirmə prosesində modelin təşkili, idarə edilməsi və oxunaqlığının yüksəldilməsi vasitələri olan paketlər tətbiq edilir. Paketlər diaqramları paket sinifləri və onların arasındakı qarşılıqlı asılılıqları əks etdirir. Bundan əlavə, paketlər həm də layihələşdirmə prosesində sistem modullarını (alt sistemləri) əks etdirir. Siniflərin məcmusu toplanmış alt sistem özünün interfeysində təqdim edilir.

Paketə daxil olan bəzi elementlər digər paketlər tərəfindən iqtibas edilə bilən ola bilir. Əgər paketin bir elementi digər paketin elementindən asılıdırsa, bu asılılıq əlaqə şəklində təsvir edilir. Hər bir paketdə adlar sahəsi (*namespace*) vardır ki, burada da hər

bir sinif unikal (təkrar olunmayan) ada malik olur. Bir sinfi digərindən fərqləndirmək üçün bitkin addan (*fully qualified name*) istifadə edilir. Çünki bitkin adda sinfin daxil olduğu paket də (struktur da) göstərilir.

UML dilində paketlərin adında ikiqat iki nöqtədən istifadə edilir, məsələn: *System::Date* kimi.

Paketin adını həm ayrıca, həm də məzmunu ilə birlikdə göstərmək mümkündür. *UML* konsepsiyasına müvafiq olaraq paketlərdəki siniflər açıq (*public*) və qapalı (*private*) ola biləndir. Açıq siniflər paket interfeysinin müəyyən hissəsini tutur və digər paketlərdəki siniflər tərəfindən istifadə edilə bilirlər. Qapalı siniflərə xaricdən müdaxilə etmək mümkün deyildir. İstənilən sinfi qapalı etmək üçün "görünənlik" modifikatorunu *private* etmək kifayətdir. Bu halda həmin sinif yalnız öz paketinə aid olan siniflər üçün açılır. Həmin siniflər xarici istehlak üçün əlavə açıq siniflər yaratmaq üçün istifadə edilir. Fasad (Facades) adlanan bu əlavə siniflər açıq əməliyyatları paketin digər siniflərinə həvalə edir (göndərir).

Komponentlər diaqramı *(Component Diagram)*

Bu diaqram cari modelin fiziki təsvirini yaratmağa imkan verir. Komponentlər diaqramı ikilik və ya icra edilən fayllardakı ilkin kodda təqdim edilən proqram komponentlərinin təşkilini və qarşılıqlı əlaqələrini göstərir. Sistem müxtəlif tip modullardan təşkil edildiyindən, istifadəçi həmin müxtəlifliklər tə yin etmək üçün aşağıdakı stereotiplərdən[33] istifadə edə bilir:

- *Package* (paket) – modeldəki komponentləri qruplar halında birləşdirir;
- *Main program* (baş proqram);
- *Subprogram body* (alt proqramın mətni (gövdəsi));
- *Package specification/body* (paketin gövdəsinin (mətninin)/ təyin edilməsi).

Yerləşmə diaqramı proqram və aparat sistemlərinin qarşılıqlı əlaqələrini fiziki cəhətdən təsvir edir. Bu baxımdan həmin diaqramlar component və obyektlərin

[33] Stereotiplərin dəyişməsi adətən komponentin diaqramdakı təsvirini dəyişdirir. Стереотип - м mətb. stereotip (metal, rezin lövhəcik üzərinə keçirilmiş mətbəə naboru yaxud klişesi surəti).

sistemdə yerləşməsini yaxşı təsvir edən vasitə hesab olunur. Yerləşmə diaqramındakı hər bir düyün hesablayıcı qurğunun bir tipini, çox hallarda isə aparatura hissəsini təsvir edir. Bu aparatura sadə qurğu və ya datçik (informasiya vericisi), yaxud da maynfreym ola bilər. Yerləşmə diaqramı şəbəkənin və şəbəkədəki müxtəlif komponentlərin fiziki yerləşməsini əks etdirir.

9. Yaşam tsiklinin (YT)-nin dəstəklənməsinin instrumental sistemləri

Müasir yüksək texnoloji məmulatların istehsalı çoxsaylı müəssisələrin əlaqələndirilmiş (razılaşdırılmış) fəaliyyət göstərməsini nəzərdə tutur. Məmulatların layihələşdirilməsi, istehsalı və reallaşdırılması və istismar edilməsində iştirak edən bütün müəssisələrin razılaşdırılmış işləməsini təmin etmək üçün məmulatın YT-nin mərhələlərinin müvafiq informasion dəstəklənməsindən istifadə edilir ki, bu da *CALS (Continuous Acquisition and Lifecycle Support)* adı almışdır. *CALS* – YT-nin bütün mərhələlərində informasion inteqrasiya və varislik hesabına məmulatın yaşam tsiklinin gedişində yerinə-yetirilən biznes-proseslərin səmərəliliyinin yüksəldilməsi üzrə qlobal strategiyadır. *CALS* konsepsiyası istehsal edilən məmulatın modelini yaratmaqdan ibarətdir. Bu model məmulatı bütün YT müddətində izləməlidir. Bunun üçün vahid informasiya fəzasının, başqa sözlə, inteqrasiya edilmiş informasiya mühitinin yaradılıb aktual vəziyyətdə saxlanması lazımdır. İnteqrasiya edilmiş informasiya mühiti məhsulun sifarişçisi, istehsalçısı və istehlakçısı arasında məlumat mübadiləsi vasitəsilə yaradılır. Beləliklə, konkret məmulatın yaşam tsiklinin bütün mərhələlərində forma və məzmunca unifikasiya edilmiş informasiyadan istifadə etmək imkanı yaranmış olur.

Yaşam tsiklinin (YT)-nin dəstəklənməsinin instrumental sistemləri aşağıdakılardan ibarətdir:

CAE (Computer Aided Engineering) — avtomatlaşdırılmış hesablamalar və təhlil;

CAD (Computer Aided Design) — avtomatlaşdırılmış layihələşdirmə;

CAM (Computer Aided Manufacturing) — avtomatlaşdırılmış istehsalın texnoloji hazırlığı;

PDM (Product Data Management) — layihə mməlumatlarının idarə edilməsi;

ERP (Enterprise Resource Planning) — müəssisənin planlaşdırılması və idarə edilməsi;

MRP II (Manufacturing Requirement Planning) — istehsalın planlaşdırılması;

MES (Manufacturing Execution System) — istehsal-icraedici sistem;

SCM (Supply Chain Management) — malgöndərmələr zəncirinin idarə edilməsi;

CRM (Customer Relationship Management) — sifarişçilərlə qarşılıqlı münasibətlərin idarə edilməsi;

SCADA (Supervisory Control And Data Acquisition) — istehsal proseslərinin dispetçer idarəetməsi;

CNC (Computer Numerical Control) — kompüterləşdirilmiş rəqəmsal idarəetmə;

S&SM (Sales and Service Management) — satışın və xidmətlərin idarə edilməsi;

CPC (Collaborative Product Commerce) — birgə elektron biznes.

Mütəxəssislərin hesablamalarına görə, ABŞ sənayesində *CALS*-texnologiyanın tətbiqinin səmərəlilik göstəriciləri aşağıdakı kimidir:

- layihələşdirməyə birbaşa xərclərin azaldılması - 10 - 30 %;
- məmulatın işlənib hazırlanması müddətinin ixtisar edilməsi - 1,5 - 2 dəfə;
- yeni məmulatların bazara çıxarılması müddətinin ixtisar edilməsi – 25 - 75%;
- konstruktiv dəyişikliklərin həcminin və zayın xüsusi çəkisinin azaldılması - 23 - 73 %;
- texniki sənədləşmənin hazırlanmasına çəkilən xərclərin ixtisar edilməsi - 40 %-dək;
- istismar sənədləşməsinin hazırlanmasına çəkilən xərclərin ixtisar edilməsi -30 %-dək.

Hal-hazırda kifayət qədər çox sayda "İnteqrasiysa edilmiş müəssisə idarəetmə sistemləri" (İMİS-lər) mövcuddur. Lakin bunların içərisində dünya bazarında ən populyar olan sistem *SAP* adlı alman firmasının *R/3* sistemidir. *SAP R/3* sistemi şirkətin müxtəlif biznes-proseslərini dəstəkləyən tətbiqi modullar yığımından ibarətdir və bu modullar öz aralarında inteqrasiya edilərək real vaxtda fəaliyyət göstərən *maliyyə (Fİ), nəzarət (CO), əsas vəsaitlərin idarə edilməsi (AM), layihələrin idarə edilməsi (PS),*

istehsalın planlaşdırılması (PP), material axınlarının idarə edilməsi (MM), satış (SD), keyfiyyətin idarə edilməsi (QM), avadanlığın təmiri və texniki xidmət göstərilməsi (PM), işçi heyətin idarə edilməsi (HR), informasiya axınlarının idarə edilməsi (WF) və sahəvi həllər (qərarlar) (IS) kimi funksional təşkiledicilərdən ibarətdir.

Bazis sistem *SAP R/3* sistemi üçün təməl təşkil edərək bütün tətbiqi modulların inteqrasiya edilməsinə və aparat platformasından asılı olmamağa zəmanət verir. Bazis sistem çoxsəviyyəli paylanmış müştəri-server arxitekturasında işləmək imkanını təmin edir. *SAP R/3* sistemi *UNIX, AS/400, Windows NT, S/390* kimi serverlərdə və *Informix, Oracle, Microsoft SQL Server, DB2* kimi müxtəlif VBİS-lərlə fəaliyyət göstərir. İstifadəçilər *Windows, OSF/Motif, OS/2* və ya *Macintosh* mühitində işləyə bilirlər.

Yuxarıda baxdığımız *CALS*-sistemlər AVİNEMSİ-nin layihələşdirilməsi kontekstində *CAD* adlı avtomatlaşdırılmış layihələşdirmə sisteminin *CASE (Computer Aided Software Engineering)* adlanan instrumental vasitələrinin alt siniflərindən birində toplanmışdır. *CASE* instrumental vasitələri predmet oblastını əyani formada modelləşdirməyə imkan verir.

Bütün müasir *CASE*-vasitələr əsasən tiplər və kateqoriyalar üzrə təsnifləşdirilir. Tiplər üzrə təsnifat *CASE*-vasitələrin bu və ya digər YT proseslərinə funksional yönümünü əks etdirir. Kateqoriyalar üzrə təsnifat isə icra edilən funksiyaların və vasitələrin inteqrasiya edilmə səviyyələrini təyin edir. Yəni bu təsnifat əlahiddə həll edilən kiçik məsələdən və qismən inteqrasiya edilmiş lokal vasitələrdən İNSİ-dəki bütün YT-ləri dəstəkləyən bütövlükdə inteqrasiya edilmiş vasitələrədək qruplaşdırma həyata keçirir.

Tiplər üzrə təsnifat aşağıdakı əsas *CASE*-vasitələri qruplarını ayırır:

- predmet oblastı modellərini qurmaq və təhlil etmək (*Upper CASE*) üçün nəzərdə tutulan vasitələr (*Design/IDEF (Meta Software), BPwin (Logic Works*));
- *Vantage Team Builder (Cayenne), Designer/2000 (ORACLE), Silverrun (CSA), PRO-IV (McDonnell Douglas), CASE , Analitik (MacroProject)*və s. kimi layihə spesifikasiyalarının yaradılması üçün istifadə edilən geniş yayılmış layihələşdirmə metodologiyası ilə dəstəklənən layihələşdirmə və təhlil vasitələri (*Middle CASE*). Bu vasitələrin

çıxışı (nəticəsi) komponentlərin spesifikasiyalarından, sistemin interfeyslərindən, sistemin arxitekturasından, alqoritmlərdən, verilənlərin strukturundan və nisbətən geniş istifadə edilən VBİS-lər üçün, adətən *SQL* dilində qenerasiya edilən (yüklənən) verilənlər bazasının modelləşdirilməsi vasitələrindən ibarət ola bilər. Bu vasitələrə: *ERwin (Logic Works), S-Designer (SDP) и DataBase Designer (ORACLE)* aiddir. *CASE*-vasitələrinin tərkibində verilənlər bazasının layihələşdirilməsi vasitələri olan *Vantage Team Builder, Designer/2000, Silverrun и PRO-IV* geniş tətbiq edilir;

• proqram əlavələrinin işlənib hazırlanması vasitələri. Bu vasitələrə: *4GL (Uniface (Compuware), JAM (JYACC), PowerBuilder (Sybase), Developer/ 2000 SQL Windows (Gupta), Delphi (Borland*) və s.) və *Vantage Team Builder, PRO-IV*-ün və qismən də *Silverrun* sistemlərinin tərkibinə daxil olan kod generatorları aiddir;

• müxtəlif modellər və layihə spesifikasiyaları əsasında formalaşdırılan VB sxemləri və proqram kodlarının təhlilini aparan reinjinirinq vasitələri. VB cxemlərinin təhlili və *ERD*-nin formalaşdırılması vasitələri *Vantage Team Builder, PRO-IV, Silverrun, Designer/2000, ERwin* və *S-Designor* sistemlərinin tərkibinə daxildir.

C++ proqramlaşdırma dilində reinijiniriq təmin edən *Rational Rose (Rational Software), Object Team (Cayenne))* kimi obyekt yönümlü CASE-vasitələri proqram kodlarının təhlili sahəsində geniş tətbiq tapmışdır.

CASE-vasitələrin köməkçi tipləri aşağıdakılardan ibarətdir:

• layihənin planlaşdırılması və idarə edilməsi vasitələri (*SE Companion, Microsoft Project* və s.);

• konfiqurasiyalı idarəetmə vasitələri (*PVCS (Intersolv*));

• testləşdirmə (yoxlama) vasitələri (*Quality Works (Segue Software*));

• sənədləşdirmə vasitələri (*SoDA (Rational Software*)).

Proqram təminatı yaradılması sahəsində işləyən *IBM (RationalSoftware), Oracle* və *Borland* kimi 3 tanınmış şirkətin təqdim etdiyi *CASE*-vasitələr komplektləri nümunə göstərilə bilər. Korporativ dünya standartı kimi seçilə biləcək ən mükəmməl texnologiya *Rational Unified Process (RUP)*-a məxsusdur. *RUP* hal-hazırda *IBM*-in

tərkibinə daxil olan *RationalSoftware* şirkəti (*www.rational.com*) tərəfindən hazırlanmış proqram məhsuludur. *RUP* çərçivəsində 6 əsas İNSİ prosesləri müəyyən edilmişdir: biznes-modellərin qurulması; tələblərin müəyyən edilməsi; təhlil və layihələşdirmə; reallaşdırma (satış); testləşdirmə; genişləndirmə (inkişaf etdirmə).

Oracle (www.oracle.com) korporasiyasının PT yaratmaq üzrə metodiki əsasını PT-nin YT-sinin əksər proseslərini əhatə edən *Oracle Method* adlanan metodlar kompleksi təşkil edir. Bu kompleksin tərkibi aşağıdakılardan ibarətdir:

- *CDM (Custom Development Method)* – Tətbiqi proqram təminatının yaradılması;
- *PJM (Project Management Method)* – layihənin idarə edilməsi;
- *AIM (Application Implementation Method)* – Tətbiqi proqram təminatının tətbiq edilməsi;
- *BPR (Business Process Reengineering)* – biznes-proseslərin reinjinirinqi;
- *OCM (Organizational Change Management)* –dəyişikliklərin idarə edilməsi və s.

Oracle Developer Suite kompleksi proqram əlavələrinin sürətlə yaradılması üçün inteqrasiya edilmiş vasitələr yığımına malikdir. *Oracle Developer Suite Oracle Database və Oracle Application* server ilə inteqrasiya edilərək proqram əlavələrinin yaradılması və qurulması üzrə vahid platformaya çevrilmişdir.

Oracle Developer Suite J2EE standartlarına34 cavab verir. Bunun tərkibinə, həmçinin, *XML* analizatoru və *XSLT* prosessoru daxildir. *XML* və *XSLT* prosessor sxemi servlet olmaqla *XML*-əlavələrin yaradılması üçündür.

Oracle Developer Suite-də modellər əsasında əlavələrin işlənib hazırlanması üçün *UML* dilini dəstəkləyən modul vardır. Modellər *Oracle*-nin ümumi repozitoriyalarında[35] saxlanır ki, bu da böyük layihələşdirici kollektivləri dəstəkləmək üçündür.

Oracle Developer Suite aşağıdakıları özündə birləşdirir:

- *Oracle Designer* – əlavələrin modelləşdirilməsi və generasiya edilməsi vasitəsi;
- *Oracle Forms* – əlavələrin sürətlə işlənib hazırlanması vasitəsi;
- *Oracle Reports* – hesabatların hazırlanmasının vizual vasitəsi;

---

[34] *Enterprise Java Beans (EJB), JavaServer (JSP)* servletlərinə və səhifələrinə
[35] boşalmış yaddaş sahələrində

• *Oracle JDeveloper – Java* dilində vizual proqramlaşdırma vasitəsi;

• *Oracle Discoverer* – analitik əlavələrin işlənib hazırlanması üçün vasitə;

• *Oracle Warehouse Builder* – verilənlərdən ötrü saxlanc yerlərinin qurulması üçün sistem;

• *Oracle Portal* – təşkilatın informasiya portalının işlənib hazırlanması vasitəsi.

Oracle Designer-in tərkibinə aşağıdakı komponentlər daxildir:

• *Repository Administrator* – repozitoriyanın idarə edilməsi[36] vasitəsi;

• *Repository Object Navigator* – repozitoriyanın bütün elementlərinə müraciəti reallaşdıran çoxpəncərəli obyekt yönümlü interfeysi təmin edən repozitoriyaya müraciət vasitəsi;

• *Process Modeler* – biznes-proseslərin təhlili və modelləşdirilməsi vasitəsi;

• *Systems Modeler* – layihələşdirilən sistemin funksional və informasiyon modellərinin qurulması vasitələri yığımı;

• *Systems Designer* – PT-nin layihələşdirilməsi vasitələri yığımı[37];

• *Server Generator –Oracle* VB-si obyektlərinin[38] təsviri generatoru;

• *Forms Generator –Oracle Forms* üçün əlavələr generatoru. Generasiya edilmiş (yüklənmiş) əlavələr müxtəlif ekran formalarını, verilənlərə nəzarət və bütövlük məhdudiyyətlərinin yoxlanması vasitələrini, həmçinin avtomatik kömək vasitələrini özündə birləşdirir;

• *Repository Reports –Oracle Reports*-la inteqrasiya edilmiş standart hesabatlar generatoru.

Borland şirkəti (*www.borland.com*) öz layihələrini inkişaf etdirməklə və bir-sıra şirkətləri satın almaqla əlavələrin tam yaşam tsiklini (YT) idarə edən *Application Life*

---

[36] əlavələrin yaradılması və ləğv edilməsi, müxtəlif istifadəçilər tərəfindən verilənlərə müraciətin idarə edilməsi, verilənlərin ixrac və idxal edilməsi

[37] Bu vasitələr yığımı *PL/SQL* dilinin saxlanmış prosedurları ilə reallaşdırılan relyasiyalı VB strukturlarını qurmaq üçün olan vasitələri, verilənlərlə qarşılıqlı fəaliyyəti və əlavələrin iyerarxiyasını, strukturunu və məntiqini əks etdirən diaqramqurma vasitələrini özündə birləşdirir.

[38] cədvəllərin, indekslərin, açarların, ardıcıllıqların və s.

Cycle Management, (ALM) adlı inteqrasiya edilmiş instrumental vasitələr kompleksi təklif etmişdir.

Borland şirkəti tərəfindən dəstəklənən instrumental vasitələrin tərkibinə aşağıdakılar daxildir:

• *CaliberRM –CaliberRM* tələblərinin idarəetmə sistemi. *CaliberRM* tələbləri VB-də saxlayır. Sənədlər öz izahları (təsvirləri) ilə VB-dəki *MS Word* şablonları üzrə sənədləri generasiya edən mexanizmin köməyi ilə yaradılır. *CaliberRM* tələblər arasındakı asılılıqları müxtəlif vizuallaşdırma metodlarını dəstəkləyir ki, bunların da köməyi ilə istifadəçi təhlilin miqyasını lazım olan qədər kiçildib yalnız dəyişikliyə uğrayan tələb üzərində cəmləməklə kifayətlənə bilir.

• *Together ControlCenter – J2EE (Java)* və *.Net (C#, C++* və *Visual Basic)* platformaları üçün əlavələrin yazılmasından ötrü *UML* dilində vizual modelləşdirməni dəstəkləyən inteqrasiya edilmiş layihələşdirmə və işləyib hazırlama mühiti. Bu sistemdə *LiveSource* texnologiyası reallaşdırılmışdır ki, bu da proqram əlavəsi layihələri ilə dəyişikliklər arasında sinxronlaşdırmanı təmin edir. Belə ki, ilkin mətndə dəyişiklik edildikdə proqramın modeli dəyişir, model dəyişdikdə isə proqramlaşdırma dilində mətn tələb olunan şəkildə dəyişilir.

• Versiyalara nəzarət *Together* və *StarTeam* sistemlərinin funksional inteqrasiyası sayəsində həyata keçirilir. Həmçinin *Rational ClearCase* adlı konfiqurasiyanın idarə edilməsi sistemi ilə inteqrasiya da dəstəklənir.

• *Optimizeit Suite 5, Optimizeit Profiler for .NET* – aparat resurslarından[39] istifadə problemi. *Optimizeit Suite 5*-in *Jbuilder* layihəsi mühitinə, *Optimizeit Profiler* layihəsinin isə – *C#Builder* və *Visual Basic .Net* mühitinə inteqrasiya edilməsi əlavələrin yoxlama sınağından keçirilməsinə və layihələşdirmənin gedişində məhsuldarlığa mənfi təsir edən blokların dəyişdirilməsinə imkan verir.

[39] yaddaşdan və müvafiq olaraq *J2EE* və *.Net* platformalarındakı prosessor güclərindən

Optimizeit ServerTrace – verilmiş xidmət səviyyəsinə nail olmaq baxımından server əlavələrinin məhsuldarlığının idarə edilməsi və virtual *Java*-maşınlar üzrə nəzarət verilənlərinin yığılması üçün nəzərdə tutulmuşdur.

StarTeam – PT-nin yaşam tsikllərinin (YT-nin) bütün əsas fazalarını birləşdirir. Bu, versiyalara nəzarəti həyata keçirir; dəyişiklikləri idarə edir; qüsurları izləyir; *CaliberRM*-lə inteqrasiya edilərək tələbləri idarə edir; məsələlər axınını idarə edir; layihəni idarə edir.

StarTeam Microsoft Source Code Control interfeysi ilə uyuşandır və *API* tərəfindən dəstəklənən istənilən layihələşdirmə sistemi ilə inteqrasiya edilə bilir. Bundan əlavə, bu sistemdə *Together, JBuilder, Delphi, C++Builder u C#Builder* kimi layihələşdirmə və modelləşdirmə vasitələri ilə inteqrasiyaedici vasitələr də mövcuddur.

10. Tələblərin yığılması və təhlili

Layihələşdirilən sistemə qarşı qoyulan bütün tələblər bir-neçə iyerarxiya səviyyəsində yerləşdirilir. Bu iyerarxiyanın ən aşağı səviyyəsində elə tələblər yerləşdirilir ki, onlar konkret tətbiq sahəsinin xüsusiyyətlərini nəzərə almadan bütövlükdə texnoloji proseslərin avtomatlaşdırılması üçün eyni ilə uyğun gəlir. Burada əsas diqqət dövlət standartlarına və digər normativ sənədlərə yönəldilir. İyerarxiyanın aşağıdan ikinci səviyyəsində verilmiş texnoloji prosesin qaydalarını və izahlarını təyin edən normativ sənədlərə uyğun olan və avtomatlaşdırılmış sistemdə müəyyən verilmiş sinfə aid tələblər yerləşdirilir. Nəhayət, aşağıdan üçüncü səviyyədə konkret sistemə aid tələblər yerləşdirilir. Bundan əlavə, *IEEE 830-1993* standartında PT-yə qarşı qoyulan tələblər Spesifikasiyasında (*Software Requirements Specification – SRS*) bütün tələblər iki qrupa bölünmüşdür. Tələblərin birinci qrupu sifarişçinin istədiklərini sənədləşdirir və sifarişçiyə aydın olan dildə yazılır. Buna *C*-tələblər deyilir. İkinci qrup xüsusi, strukturlaşdırılmış formada sənədləşdirilən tələblərdir. Bu, layihələşdirici tələbləridir. Bu *D*-tələblər adlanır.

Dövlət standartında[40] AVİNEMSİ-nin layihələşdirilməsi zamanı rəhbər tutulması zəruri olan aşağıdakı ümumsistem prinsipləri sadalanmışdır:

1. ***sistemlik*** – yəni AİS[41] yaradılarkən, fəaliyyət göstərərkən və inkişaf etdirilərkən onun bütövlüyünü təmin edən struktur elementləri arasında əlaqə yaradılmalı və qorunmalıdır;

2. ***inkişaf*** – yəni, AİS mövcud texniki və proqram vasitələrinin təkmilləşdirilməsi yolu ilə funksional və təminat elementlərinin zənginləşdirilməsi imkanları nəzərə alınmaqla yaradılmalıdır;

3. ***birgəlik (uyuşanlıq)*** – yəni, AİS müxtəlif növ fəaliyyətlər və müxtəlif səviyyəli proseslərlə birgə qarşılıqlı fəaliyyət göstərmək qabiliyyətinə malik olmalıdır;

4. ***standartlaşdırma və unifikasiya*** – yəni, AİS yaradılarkən və inkişaf etdirilərkən tipik, unifikasiya edilmiş və standartlaşdırılmış elementlərdən istifadə edilməlidir;

5. ***səmərəlilik*** – yəni, AİS-in yaradılmasına çəkilən xərclərlə onun fəaliyyəti nəticəsində əldə edilən qazanc arasında faydalı nisbət təmin edilməli, başqa sözlə, qazanc xərci artıqlaması ilə örtən olmalıdır.

Bundan əlavə, Dövlət standartında[42] göstərilir ki, AİS yaradılarkən tipik layihə həllərindən, tətbiqi proqram paketlərindən (TPP), unifikasiya edilmiş paketlərdən maksimal dərəcədə istifadə edilməli və yeni obyektlər üçün AİS yaradılarkən mövcud AİS layihəsinə əsaslanmaq lazımdır.

Bu qayda PT mühəndisliyi prinsiplərinə tamamilə uyğundur. Çünki bu prinsiplərdən biri komponentlərdən təkrar istifadə konsepsiyasıdır.

Dövlət standartı[43] hər bir təminat növü üçün ayrıca tələblər müəyyən etmişdir. Bunlardan ən vacibləri aşağıdakılardır:

Proqram təminatı (PT) üzrə tələblər:

1.AİS-də, əsasən, mövcud qaydada qeydiyyatdan keçirilmiş və qurulmuş VBİS-lərdən[44] istifadə edilməlidir.

---

[40] ГОСТ 24.103-84 «Автоматизированные системы управления. Основные положения»
[41] Avtomatlaşdırılmış idarəetmə sistemi
[42] п.3.4. ГОСТ 24.103-84
[43] ГОСТ 24.104-85 «Автоматизированные системы управления. Общие требования»

2. AİS-in proqram təminatında AİS-in texniki vasitələrinin diaqnostikası və giriş informasiyanın gerçəkliyinə nəzarət vasitələri olmalıdır.

3. AİS-in proqram təminatında informasiyanın daxil edilməsi və emalı zamanı səhvlərdən qorunma tədbirləri nəzərdə tutulmalıdır ki, lazımi funksional keyfiyyət təmin edilə bilsin.

İnformasiya təminatı üzrə (İT) tələblər:

1. AİS-i tərk edən çıxış informasiyanın forması sifarişçi (sistem istifadəçisi) ilə razılaşdırılmalıdır.

2. AİS-in informasiya massivlərindəki verilənlərə nəzarət və verilənlərin təzələnməsi, həmçinin AİS-ə aid olan bu və ya digər texniki vəsaitin imtinasından sonra massivlərin bərpası üzrə tədbirlər nəzərdə tutulmalıdır. Bundan əlavə, AİS-dəki VB-də olan eyniadlı informasiyanın tamoxşarlığına nəzarət edilməsi mümkün olmalıdır.

Təhlükəsizlik üzrə tələblər:

1) AİS-in işçi heyətinin düzgün olmayan fəaliyyəti qəza situasiyası yaratmamalıdır.

2) Ümumsistem tələbləri içərisində etibarlılıq və təhlükəsizlik tələbləri xüsusi rol oynayır.

3) Tələblərin yığılması ilə yanaşı tələblərin idarə edilməsi də mühüm əhəmiyyət daşıyır.

Rational Unified Process texnologiyasında tələblərin idarə edilməsi prosesində 5 kamillik səviyyəsi təyin edilmişdir:

1. tələblər razılaşdırılmış formatda yazılmalıdır;

2. tələblər təşkil olunmuşdur, tələblər barədə olan standart formatlardan və metaverilənlərdən istifadə edilir, versiyalara nəzarət;

3. tələblər öz tipləri (funksional və qeyri-funksional) üzrə strukturlaşdırılmışdır;

4. tələblər öz tipləri üzrə izlənirlir;

5. tələblər dəyişikliklərin idarə edilməsi, modelləşdirilməsi, kodlaşdırma və s. proseslərlə inteqrasiya edilir.

Birinci səviyyəyə uyğun olmaq üçün standart mətn redaktoru və ya tələblərin

---

[44] VBİS - verilənlər bazasını idarə edən sistem

saxlanması üçün elektron cədvəl seçilmişdir; bu səviyyədə tələblərin istifadə edilməsi prinsipləri müxtəlif əmr qrupları üzrə standartlaşdırılmamışdır.

İkinci səviyyədə tələblərin təsvirinin qeyd edildiyi sənədlər razılaşdırılmış formatda olmaqla, müəyyən nömrələmə sxeminə və nəzarət versiyasına uyğun olur.

Strukturlaşdırılmış tələblər səviyyəsi tələblərin prioriteti, risk, dəyər və s. kimi bir-sıra atributlara malik standart spesifikasiyaların yaradılmasına keçildiyini göstərir.

Növbəti səviyyələr müxtəlif tələblər arasındakı asılılıqları və bu tələblərin PT-nin yaşam tsiklindəki ddigər modellərə təsirini izləmək vəzifəsini qoyur.

Yaxşı spesifikasiyanı aşağıdakı elementlər fərqləndirir:

1. Xırdalıqlara aksent edilməsi və onların dəqiq müəyyənləşdirilməsi.

2. Qeyri-düzgün anlama mümkünsüzlüyü barədə narahatlıq.

3. Verilmiş spesifikasiya daxilində və digər spesifikasiyalarla əlaqədar ziddiyyət-sizlik.

4. Komponentlərin qarşılıqlı məntiqi əlaqəsi.

5. Predmetin əhatəedilmə tamlığı.

6. Normativ aktlara uyğunluq.

7. İşgüzar praktikaya uyğunluq.

C-Tələblər

Sistemə sifarişçi tələbləri, yəni *C*-tələblər layihələşdiricilər tərəfindən təşkil edilən xüsusi sorğu-müsahibə aparılması yolu ilə qeyd edilib toplanır. Lakin bundan əvvəl sistem istifadəçilərini identifikasiya etmək, yəni yaradılacaq sistemlə işləyəcək şəxslərin kateqoriyalarını təəyin etmək lazımdır. Sistemin miqyasından asılı olaraq bu məsələ trivial[45] olmaya bilər. Buna görə də xüsusi diqqət tələb edir. Çünki sistem yaradılarkən bütün marağı olan şəxslərin tələbi öyrənilib nəzərə alınmalıdır.

Müsahibə götürməzdən əvvəl: müsahibə götürüləcək səxslər sadalanmalı, nüfuz dərəcəsi üzrə çeşidlənib siyahıya alınmalıdır; müsahibənin başlanması və bitməsi vaxtları dəqiq müəyyən edib planlaşdırılmalıdır; predmet oblastının əsas terminlərinin

[45] тривиальный - прил. bayağı, adi, duzsuz, çeynənmiş, köhnəlmiş, köhnə.

sadalandığı layihə sözlüyü (qolossari) tərtib edilməlidir.

Müsahibə zamanı: tələblər sənədləşdirilməlidir; qrafik interfeysin eskizi tərtib edilir; növbəti görüş (müsahibə) planlaşdırılır.

Müsahibədən sonra: CASE-instrumentlərdən istifadə etməklə *C*-tələblərin qaralaması tərtib edilir; avadanlıqların konfiqurasiyası müəyyənləşdirilir; *C*-tələblərin qaralaması sifarişçiyə ötürülür ki, sifarişçi öz irad və düzəlişlərini etsin.

Sifarişçilər öz konsepsiyalarını adətən ətraflı düşünmədən və natamam hazırlayırlar. Odur ki, sifarişçi konsepsiyası işin gedişində az və ya çox dərəcədə dəyişikliyə uğrayır. Bu konsepsiya bəzən proqram əlavəsinin modeli, bəzən də işin konsepsiyası adlandırılır. Sifarişçinin konsepsiyasını formallaşdırmaq üçün mühəndislər aşağıdakı texnologiyaların kombinasiyasından istifadə etməli olurlar:

- *SADT* metodologiyası;
- *UML*-in istifadə variantları (nümunələr) diaqramları (*use case*).
- *UML*-in ardıcıllıqlar qiaqramı (*sequence diagram*).
- UML-in vəziyyətlər diaqramları (*statechart diagram*).
- İstifadəçi interfeysinin dizaynının prototipləşdirilməsi proqram təminatının layihələşdirilməsi fazasına daxil olsa da, bunu tələblərin formalaşdırılması fazasının bir hissəsi də hesab etmək olar.

Sifarişçi tələblərinin təhlili qaydası aşağıdakı addımlardan ibarətdir:

1. Əgər tələb sadədirsə və digər tələblərlə əlaqəli deyildirsə, onda SRS[46] tələblər spesifikasiyasının müvafiq bölməsində olan cümlələrə dəqiq uyğun deyildir;

2. Əgər tələb istifadəçi ilə proqram əlavəsinin qarşılıqlı fəaliyyətini əks etdirirsə, onda onu istifadə variantının (nümunənin) köməyi ilə ifadə edirlər. Bunun üçün: 1) nümunənin adını verirlər; 2) fəaliyyətdə olan şəxsləri təyin edirlər; 3) istifadəçi və proqqram əlavəsinin icra etdiyi əməllərin ardıcıllığını qeyd edirlər; 4) budaqlanmanı minimumlaşdırırlar.

3. Əgər tələb emal elementlərinə toxunursa və bunların hər biri verilən alıb-verirsə,

[46] *Software Requirements Specification*

onda verilənlər axını diaqramından (*DFD*) istifadə edilir və bu zaman: 1) emal elementi[47] müəyyən edilir; 2) verilənlərin saxlanc yeri təyin edilir; 3) emaledici elementlər arasında verilənlərin ötürülməsi yolu və hər bir halda ötürülən verilənin tipi göstərilir.

4. Əgər tələb proqramın və ya proqram hissəsinin mövcud vəziyyətinə toxunursa, onda vəziyyətlər diaqramını aşağıdakı ardıcıllıqla qururlar: 1) vəziyyət təyin edilir[48]; 2) ilkin (başlanğıc) və sonuncu vəziyyətlər göstərilir; 3) sistemin baxılan hissəsindən kənarda baş verən və vəziyyətdən-vəziyyətə keçidə səbəb olann hadisələr müəyyənləşdirilir; 4) iç-içə olan vəziyyətlər[49] təyin edilir.

C-tələblər yığılan kimi, bir qayda olaraq, *SPMP* təzələnməsinə zərurət yaranır. Bu cür təzələnmə sistemin bütün yaşam tsikli ərzində baş verir. *C*-tələblər təhlil edildikdən sonra sistemin dəyəri dəqiqləşdirilir.

Tələblərin sayı çoxaldıqca formalaşdırılması və səhih idarə edilməsi məsələsi xeyli mürəkkəbləşir. Bu məsələnin formalaşdırılmasına xeyli əsərlər həsr edilmişdir[50]. Bundan əlavə, tələblərin idarə edilməsi üzrə instrumental *CASE*-sistemlər[51] işlənib hazırlanmış və uğurla istifadə edilir.

D-Tələblər

Proqram təminatı layihələşdiriciləri üçün müvafiq baza lazımdır. Bu baza xırdalıqlı tələblərdir, yəni D-tələblərdir. D-tələblər proqramdakı konkret xassə və funksionallıqlar siyahısıdır. Bu tələblərin hər biri nömrələnmiş, nişanlanmış və layihələşdirmə gedişində izlənilir. D-tələblər C-tələblərlə razılaşdırılmalıdır. D-tələblərin bir-neçə tipi vardır:

1. Funksional tələblər. Bunlar layihələşdirilən sistemin görəcəyi işi təyin edir. Məsələn, “proqram əlavəsi istifadəçi portfelinin dəyərini hesablayacaqdır”.

---

[47] adətən yüksək səviyyə elementi nəzərdə tutulur
[48] hər bir vəziyyət felləşdirilmiş isim, məsələn, “Gözləmə” kimi təyin edilir
[49] yəni biri digərinin içində olan vəziyyətlər
[50] Lauesen S. User Interface Design. A Software Engineering Perspective / S. Lauesen. – Harlow : Addison Wesley, 2005. – 623 p.; Вигерс К. Разработка требований к программному обеспечению / К. Вигерс. – М. : Издательско-торговый дом «Русская редакция», 2004. – 576 с.; Халл Э. Разработка и управление требованиями. Практическое руководство пользователя / Э. Халл, К. Джексон, Д. Дик. – Оксфорд: Springer, 2005. – 240 с.
[51] məsələn, *RequisitePro, DOORS* və s.

2. Qeyri-funksional tələblər. Buraya: 1) Məhsuldarlıq; 2) Etibarlılıq və təhlükəsizlik; 3) Səhvlərin emalı; 4) İnterfeys tələbləri; 5) Məhdudiyyətlər daxildir.

Məhsuldarlığa qarşı qoyulan tələblər proqramın icrası müddətinə qoyulan məhdudiyyətdir. Sifarişçilər və layihələşdiricilər hesablama müddətini, operativ yaddaşdan istifadəni, yaddaş qurğularının tutumunu və s. müzakirə edirlər.

Etibarlılığa qarşı qoyulan tələblər etibarlılığı ölçülə bilən kəmiyyətlərlə təyin edir. Bu tip tələblər proqramın ideal olmayan işləmə ehtimalını nəzərdə tutur və qeyri-mükəmməllik oblastını məhdudlaşdırır. Layihə tapşırığında sistemin təhlükəsizliyinin təmin edilməsinə xüsusi diqqət yetirilir. *Təhlükəsizliyə* qarşı qoyulan tələblər icazəsiz müdaxiləyə, sistem və tətbiqi proqram təminatlarının imtinasına məhdudiyyətlər şəklində formalaşdırılır.

Səhvlərin emalına qarşı qoyulan tələblər meydana gələn səhvlərə proqramın necə reaksiya verəcəyini təyin edir. Məsələn, əgər proqramın başqa proqramdan aldığı xəbər icazəsiz formatdadırsa, xəbər alan proqram nə etməlidir? Bu, səhv xəbər alan proqramın generasiya etdiyi səhv deyil.

İnterfeys tələbləri proqramın ərafdakılarla ünsiyyət saxlama formatını təsvir edir.

Məhdudiyyətlər proqram əlavəsinin necə düşünülüb hazırlanmasına hədlər qoyulmasını nəzərdə tutur. Məsələn, proqramlaşdırma dilinə, layihələşdirmə instrumentlərinə məhdudiyyətlər qoyula bilər.

3. Əks tələblər. Bu tələblər sistemin təmin etmədiyi funksionaldır.

Obyekt yönümlü yanaşmadan istifadə etməklə, funksional *D*-tələblərin tipik alınma ardıcıllığı aşağıdakı kimidir:

1. Proses istifadə variantlarında sözü gedən təhlil siniflərinin sadalanması ilə başlanır.

2. Alınmış siniflər yığımı adətən tam (bitkin) olmur. Buna görə də predmet oblastının yerdə qalan siniflərini də axtarıb tapmaq lazımdır.

3. Alınmış siniflərin hər biri üçün həmin sinfin cavab verəcəyi bütün zəruri proqram funksionallığı mövcud olmalıdır. Belə ki, məsələn, “hər bir müştərinin adı

olmalıdır”[52] və “proqram hər bir müştərinin kapitalını hesablamaq imkanına malik olmalıdır”[53]. Sinif obyektlərinin emal edəcəyi hadisələr də təyin edilməlidir. Hər bir *D*-tələb üçün testləşdirmə planı işlənib hazırlanmalıdır.

4. Siniflər arasındakı əlaqələr iki mərhələdə təyin edilir: 1) əlaqələrin ilkin yığımı əməkdaşlıq (*collaboration*) və ya ardıcıllıq diaqramlaarının təhlili əsasında müəyyən edilir. Əgər iki obyekt qarşılıqlı fəaliyyət göstərirsə, yəni məlumat mübadiləsi edirlərsə, diaqramda onların arasında əlaqə olduğu (qarşılıqlı fəaliyyətin yolu) göstərilməlidir. Bu yol müvafiq siniflər arasında iki istiqamətli assosiasiya yaradır. Əgər hər hansi iki obyekt cütlüyü arasında xəbər ötürülüşü bir istiqamətlidirsə, onda bu assosiasiyaya naviqasiya (axtarış) istiqaməti daxil edilir. 2) mahiyyət-siniflər arasındakı assosiasiyalar təhlil edilir və dəqiqləşdirilir. Assosiasiyaların gücü göstərilir. Bunlar: çoxluq assosiasiyaları, aqreqasiyalar, üümumiləşdirmələr və sinif-assosiasiyalar ola bilər.

5. *D*-tələblər yoxlanılır və *C*-tələblərlə müqayisə edilir.

6. *D*-tələblər sifarişçi tərəfindən yoxlanır və sonra nəşr edilir.

Təhlil siniflərinin alınması məsələsinə xüsusi diqqət yetirmək lazımdır. Çünki bu, predmet oblastı haqqındakı informasiyanın təhlili ilə sıx bağlı olan yaradıcılıq işidir.

Təhlil siniflərinin alınması zamanı aşağıdakı aspektləri nəzərə almaq lazımdır:

1. Təhlil sinfi predmet oblastının konkret bir elementini modelləşdirən dəqiq abstraksiyadır. Bu, 3-5 əməliyyatdan ibarət olmalıdır. Sinfin bütün xassələri onun təyinatını reallaşdırmağa xidmət göstərir. Lakin sinif öz təyinatını reallaşdırmaq üçün digər siniflərlə minimal sayda əlaqəyə malik olmalıdır.

2. Təhlil siniflərində yalnız olduqca yüksək abstraksiya səviyyəsində təyin edilmiş açar atributlar və əməliyyatlar iştirak edir. Bir qayda olaraq, təhlil səviyyəsinin hər bir əməliyyatı növbəti səviyyədə, yəni layihə səviyyəsində bir-neçə əməliyyata parçalanır.

3. Üç və daha artıq səviyyədən ibarət olan daha dərin iyerarxiya ağaclarından qaçmaq lazımdır. Çünki varislik iyerarxiyası qurularkən tez-tez buraxılan səhvlər funksional dekompozisiyada ağır nəticələr verir. Belə ki, hər bir abstraksiya səviy-

[52] Müştəri sinfinin atributudur
[53] Müştəri sinfinin metodudur

yəsində yalnız bir əməliyyat olur. Varislik yalnız predmet oblastından bilavasitə çıxan varislik iyerarxiyası olan yerdə istifadə edilir.

4. *C*-tələblər mətni təhlil edilərkən siniflər aşkarlanması zamanı isim sinfi və ya atributu, feil isə əməliyyatın əlamətini göstərir. Lakin sinonim və omonimlər yalançı siniflərin yaranmasına səbəb ola bilər.

5. İstifadəçilərlə birgə sinif aşkarlanması zamanı beyin hücumunun köməyi ilə aparılan *CRC*-təhlildən[54] istifadə edilir.

Hər bir tələbin layihədən həmin tələbi reallaşdıran proqram koduna keçirilməsinə dəqiq nəzarət təmin edilmədən proqramın düzgün işləməsinə inanmaq çətindir. Tələb dəyişdikdə bu iş daha da mürəkkəbləşir.

Layihənin müvafiq hissələrinə aid olan hər bir tələbin layihədə və proqramda əks etdirilməsi imkanı izləmə[55] adlanır.

AVİNEMSİ-yə qarşı tələblər formalaşdırılarkən bir-sıra göstəricilərə, o cümlədən, *izləmə, tamlıq, uyğunluq*[56] kimi daha vacib göstəricilərə nail olmaq lazımdır.

11. AVİNEMSİ konsepsiyasının işlənib hazırlanması

AVİNEMSİ konsepsiyasının işlənib hazırlanması aşağıdakı addımları özündə birləşdirir:

1. Sistemin arxitekturasının yaradılması (strukturlaşdırılması).

2. Altsistemlərin qarşılıqlı fəaliyyəti üçün interfeyslərin təyin edilməsi.

3. Altsistemlər modullara parçalanır (dekompozisiya edilir). Modulların tipi və modullararası birləşmələr təyin edilir.

IEEE 1471 Recommended Practice for Architectural Description of Software tərəfindən verilən tərifə görə, arxitektura dedikdə, sistemin fundamental (təməlli) təşkili nəzərdə tutulur ki, bu da özünü komponentlərdə, komponentlərarası və ətraf mühitlə münasibətlərdə, həmçinin sistemin layihələşdirilməsi və inkişafı prinsiplərində ifadə edir.

---

[54] *CRC - Class-Responsibilities-Collaborators*

[55] проследить - сов. 1. güdmək, təqib etmək, izləmək, pusmaq; 2. öyrənmək, tədqiq etmək.

[56] Согласовать - несов.1. uyğunlaşdırmaq; 2. razılaşdırmaq; 3. qram. uzlaşdırmaq.

Arxitekturanın layihələşdirilməsi

Proqram təminatının müxtəlif arxitekturalarının təsvirinə həsr edilmiş çoxsaylı ədəbiyyat mövcuddur. Hal-hazırda səmərəli İNSİ arxitekturası qurmaq prinsipləri işlənib hazırlanır.

Şou və Qarlan İNSİ-ləri aşağıdakı siniflərə bölməyi təklif etmişlər:

1. ***Verilənlərin axınları arxitekturası***. Bu arxitekturada sərbəst surətdə layihələşdirilən və fəaliyyət göstərən modullar bir moduldan digərinə yerdəyişən verilənlərin axınlarını emal edir. İNSİ-nin arxitekturasını verilənlərin axınları diaqramı şəklində təsvir etmək universal metod olsa da, diaqramları proqram koduna çevirdikdə qeyri-müəyyənlik meydana çıxa bilir.

2. ***Sərbəst komponentlər arxitekturası***. Bu arxitekturada komponentlər zamanda paralel fəaliyyət göstərirlər. Bunun aşağıdakı variantları vardır: 1) müştəri-server arxitekturası. Müştəri-server arxitekturasının da çoxsaylı modifikasiyaları vardır. Bunlardan ən geniş yayılmışı 3 səviyyəli sistemlərdir. Bu sistemlərdə ənənəvi müştəri və server arasında proqram əlavəsinin məntiqi kimi aralıq səviyyə vardır ki, bu da verilənləri yenidən dəyişdirib başqa səviyyəyə yönəldir. Bu zaman interfeys səviyyəsi proqramın icra edilən əsas funksiyalrından asılı olmur. İnterfeys səviyyəsinin elementləri (pəncərələr, *web*-səhifələr) yalnız sorğuları proqramın məntiqi səviyyəsinə yönəldirlər. Bunun nəticəsi olaraq proqramın məntiqi səviyyəsi də verilənlərin saxlanması səviyyəsi ilə qarşılıqlı fəaliyyətə başlayır. 2) qarşılıqlı fəaliyyət göstərən proseslərin paralel arxitekturasının fərqləndirici cəhəti odur ki, burada eyni vaxtda bir-neçə proses müxtəlif axınlar şəklində hərəkətə gətirilir. 3) hadisəvi-idarəetmə sistemlərinin arxitekturası ilə qurulan proqram əlavələri elə komponentlər yığımından ibarət olur ki, həmin komponentlərin hər biri bir vəziyyətdən digərinə keçmək üçün başqa komponentin təsiretmə məqamını gözləyir.

3. ***Virtual maşınlar arxitekturası***. Bu arxitekturada proqram əlavəsinə xüsusi dildə yazılmış proqram kimi baxılır. Bu zaman həmin dil üçün interpretator hazırlamaqdan ötrü bir-neçə proqram yazılmalıdır.

4. ***Repozitorlar (siyahılar) arxitekturası***. Bu arxitekturaya uyğun qurulmuş

sistemlərin əksəriyyəti VB-yə nəzərən tranzaksiyaların emalı üçün nəzərdə tutulmuşdur.

Arxitekturanın layihələşdirilməsi modul (alt sistem) anlayışı ilə sıx bağlıdır. Modul dedikdə, proqram kodunun fraqmenti nəzərdə tutulur. Arxitekturanın layihələşdirilməsi interfeys və reallaşdırma kimi iki hissədən ibarətdir.

Modulluq – sistemin bölünəbilmək (dekompozisiya) xassəsidir. Dekompozisiya sıx əlaqəli hissələri, zəif əlaqəli hissələrdən ayırmaqla həyat keçirilən və quruluş yaradan bölmə (parçalama) əməlidir.

Alt sistemlərə bölmə (dekompozisiya) əməli aşağıdakı məqsədləri güdür:

- sistemin abstraksiyası səviyyəsinin yüksəldilməsi;
- sistemin dekompozisiya edilməsi, sistemin digər hissələrinə təsir etmədən, ayrı-ayrı hissələri müxtəlif konfiqurasiyalar şəklində qurmağa, ayrıca göndərməyə, dəyişilməz interfeys şəraitində ayrıca hazırlamağa, paylanmış mühitdə ayrıca yerləşdirməyə imkan verir;
- əsas resurslara müdaxiləni məhdudlaşdırmaq məqsədi ilə sistemin hissələrə bölünməsi;
- dəyişdirilməsinə icazə verilməyən mövcud məhsulların və ya xarici sistemlərə nəzərən komponentlərin təqdim edilməsi.

Arxitekturanın layihələşdirilməsi üzrə əsas məsələ aşağıdakı 2 prinsipi rəhbər tutulmaqla həll edilir:

- ”zəif xarici əlaqəlilik” (*Low Coupling*) prinsipi – yəni ayrı-ayrı alt sistemlər arasındakı əlaqələrin sayı minimal olmalıdır;
- “güclü daxili bağlılıq” (*High Cohesion*) prinsipi– yəni hər bir alt sistem daxilindəki əlaqələrin sayı maksimal olmalıdır.

Əksər korporativ proqram əlavələrinin arxitekturası 3 laylıdır: 1-ci lay təqdimat layıdır ki, bu da istifadəçi interfeysi və əmr sətrinin köməyi ilə verilənlərin giriş-çıxışı üçün nəzərdə tutulmuşdur. Lakin təqdimat layı verilənlərin emalını və proqram əlavəsinin funksiyalarını dəstəkləmir; 2-ci lay predmet oblastı (biznes-məntiqi və ya domen) layıdır. Bu, biznes-məntiqini – qarşıya qoyulan məqsədə çatmaqdan ötrü

yaradılan proqram əlavəsinin əsas funksiyalarını - təsvir etmək üçündür; 3-cü lay verilənlərin mənbəyi (data source) layıdır. Bu, VB-yə müraciəti, xəbər mübadiləsini, xarici proqram əlavələri ilə tranzaksiyaların idarə edilməsini reallaşdırır.

Biznes-məntiqi və verilənlərin mənbəyi laylarının təqdimat layından asılılığı qəti qadağandır. Bu, təqdimat layının adaptiv olmasına imkan verir.

Predmet oblastı (biznes-məntiqi və ya *domen*) layı

Layların fiziki yerləşdirilməsi məsələsi həll edilərkən, ilk yanaşmada çalışırlar ki, onların fəaliyyətini serverdə təmin etsinlər. Çünki bu, səhvləri düzətməyi və versiyaları təzələməyi asanlaşdırır. Lakin bu halda sorğuya cavabvermə (reaksiya) sürəti aşağı düşür və daimi olaraq şəbəkəyə bağlılıq zərurəti yaranır. Əgər layihələşdirilən İNSİ üçün bu xarakteristikalar kritikdirsə[57], onda təqdimat və biznes-məntiqi layları müştəri tərəfdə yerləşdirilir. Lakin biznes-məntiqi layını müştəri ilə server arasında didib parçalamaq[58] məqsədəuyğun deyildir.

Biznes-məntiqi layının təşkili üçün aşağıdakı 3 yanaşma geniş tətbiq edilir:

1. **"Tranzaksiya ssenariləri" (*transaction script*) adlı tipik həll.** Bu yanaşma, təqdimat layından alınmış giriş informasiyanın emal edilib, yoxlanması və zəruri olduqda digər sistem əməliyyatının aktivləşdirilməsi prosedurunu nəzərdə tutur. Sonra bu prosedur işin nəticəsini lazımi formatda təqdimat layına qaytarır, lazım gəldikdə isə verilənlərin mənbəyi layına ötürür. Beləliklə, bu halda biznes-məntiqi hər bir əməliyyat və ya istifadə variantı üçün müəyyən konkret prosedurlar yığımından ibarət olur ki, bunlar da müxtəlif alt proqramlarda saxlana biləndir.

Tranzaksiya ssenarilərinin kodunun siniflər üzrə yayılmasının[59] 2 üsulu vardır:

1) bir sinfin bir-neçə tranzaksiya ssenarilərinin reallaşdırılması üçün istifadə edilməsi;

---

[57] vaz keçilməzdirsə

[58] Расщепление - ср 1. yarma, yarılma; 2. parçalama, parçalanma; 3. kim. hissələrə ayırma, hissələrə ayrılma (kimyəvi reaksiya vasitəsilə).

[59] Разнесённый прич. 1. aparılıb paylanmış, aparılıb paylaşdırılmış (verilmiş); 2. köçürülmüş, yazılmış, qeyd edilmiş; 3. dan. yayılmış (xəbər); 4. məc. dağıdılmış, parçalanmış, pərən-pərən salınmış, sovrulmuş.

2) hər bir tanzaksiya ssenarisi üçün ayrıca, məxsusi sinfin hazırlanması. Bu halda, baza metodla valideyn sinif yaradılır və verilmiş ssenarini reallaşdıran sinif valideyn sinfin varisi kimi çıxış edir.

"Tranzaksiya ssenariləri" adlı tipik həllin üstünlüyü: sadəliyində və əyaniliyində, həmçinin tranzaksiyanın dəqiq sərhədlərə malik olmasındadır. Nöqsanı isə proqram əlavəsinin strukturunu əhəmiyyətli dərəcədə mürəkkəbləşdirməsində və biznes-məntiqi mürəkkəbləşərkən kkodun təkrarlanmasındadır.

2. **"Predmet oblastının modeli" (*domain model*) adlı tipik həll.** Bu, obyekt yönümlü yanaşmaya əsaslanır. Sistemin reallaşdırdığı bütün funksiyalar yığımına müvafiq məntiqi özündə əhatə edən bir alt proqramdan istifadə etmək əvəzinə, hər bir obyektə özünün təbiətinə uyğun metodlar ayrılır.

İlk yanaşmada predmet oblastının modeli 2 tipə ayrılır: 1) predmet oblastının "sadə" modeli və 2) predmet oblastının "mürəkkəb" modeli. "Sadə" model VB-nin *ER*-modeli ilə üst-üstə düşür. Bu modeldə domenin hər bir obyektinə VB-nin bir cədvəli uyğun gəlir. "Mürəkkəb" modeldə domenin strukturu VB-nin *ER*-modeli ilə üst-üstə düşmür və tərkibinə varislik iyerarxiyası və qarşılıqlı əlaqəli xırda obyektlərin mürəkkəb şəbəkəsi daxildir.

İNSİ-nin biznes-məntiqini reallaşdırmaq üçün predmet oblastı modelinin tətbiqi VB-nin relyasiyalı strukturunda obyektlərin əks etdirilməsi ilə bağlı olan əlavə çətinliklərlə müşayiət olunur. Lakin bu məsələni həll etməyin bir-neçə üsulu vardır ki, bunlardan da biri "verilənlər mənbəyi" layında *pattern*[60] (şablon) kimi işlədilən **verilənlər dəyişdiricisindən** istifadə edilməsidir.

Verilənlərin mənbəyi (*data source*) layı

Verilənlərin mənbəyi layının rolu, əsasən, əksər hallarda relyasiya VB ilə qarşılıqlı fəaliyyət göstərməkdən ibarət olur. Verilənlərin mənbəyi layının reallaşdırılması üçün geniş yayılmış yanaşmalar aşağıdakılardır:

---

[60] *pattern* ['pætən] n nümunə; çeşid və ya *pattern* ['pætən] 1) nümunə, model, ülgü (şablon) 2) attr. nümunəvi, nümunə

1. *SQL* kodunun yaxşı köçürülməsini dəstəkləmək üçün onu biznes-məntiqindən ayırıb xüsusi siniflərdə yerləşdirirlər. Həmin siniflər VB-nin hər bir cədvəlində struktur baxımından yenidən yaradılır və şlüz formalaşdırılır. Bu şlüz cədvələ müraciət imkanını dəstəkləyir. "Şlüz" adlı tipik həllin (*gateway*) reallaşdırılmasının 2 variantından istifadə edilir:

a. **verilənlərin yazılması şlüzü (*row data gateway*)** VB-yə yönəldilən sorğunun emalı nəticəsi kimi qaytarılan hər bir yazı üçün şlüz nüsxəsindən istifadə edilməsini nəzərdə tutur. Bu cür reallaşdırılma zamanı verilənlərin yazılması şlüzü obyekt olaraq VB-nin bir cədvəl sətrini tamamilə təkrarlayır[61]. Verilənlərin yazılması şlüzü reallaşdırılarkən axtarış metodunun yerləşdirilməsi barədə qərar qəbulunda tez-tez çətinliklər yaranır. Müxtəlif verilənlər mənbələri ilə qarşılıqlı fəaliyyət zəruri olduqda hər bir VB cədvəli üçün ayrıca sinif yaratmaq məqsədəuyğun olur.

b. **verilənlər cədvəli şlüzü (*table data gateway*)** VB-nin ayrıca cədvəli ilə müqayisə edilən, sorğuları aktivləşdirmə metodlarını saxlayan və yazılar çoxluğunu qaytaran siniflərdən istifadəni nəzərdə tutur. "Yazılar çoxluğu" modeli VB-də saxlanan və cədvəl təqdimat formasını imitasiya (təqlid) edən verilənlərin təməl strukturudur. "Yazılar çoxluğu" patternini reallaşdıran qrafik interfeys elementləri müxtəlif instrumental sistemlərlə (*IDE*) dəstəklənir. **Verilənlər cədvəli şlüzü interfeysi** verilənləri VB-dən çıxaran bir-neçə axtarış metodunu, həmçinin təzələmə, daxiletmə və ləğvetmə metodlarını özündə birləşdirir. Hər bir metod arqumentləri müvafiq *SQL* əmrinə göndərir və əlaqə yaradılmış VB ilə kontekstdə icra edir.

2. Verilənlərin mənbəyi layının təqdimatı üçün "sadə" tipə aid olan predmet oblastı modelindən istifadə etməklə biznes-məntiqinin təsviri zamanı "aktiv yazı" (*active record*) adlı tipik həllindən istifadə edilə bilər.

Hər bir aktiv yazı informasiyanın VB-də saxlanmasına və yüklənməsinə, həmçinin verilənlərə tətbiq edilən domen məntiqinə cavabdehdir. Aktiv yazı verilənlərinin strukturu VB cədvəlinin strukturuna tam dəqiq uyğun olmalıdır. Yəni obyektin hər bir polesi cədvəlin bir sütununa uyğun gəlməlidir. Bir qayda olaraq, aktiv yazı aşağıdakı

[61] Cədvəlin hər sütunu bir yazı polesinə uyğundur.

əməliyyatların icrası üçün nəzərdə tutulan metodları özündə birləşdirir:

1) *SQL*-corğu icrası nəticəsində alınmış sətr əsasında aktiv yazı nüsxəsinin yaradılması;

2) sonra cədvələ daxil edilmək üçün yeni aktiv yazı nüsxəsinin yaradılması;

3) standart *SQL*-sorğular icra edən və aktiv yazıları VB-yə qaytaran statistik axtarış metodları;

4) VB-nin təzələnməsi və aktiv yazıdan verilənlərin VB-yə daxil edilməsi;

5) pole qiymətlərinin çıxarılması və daxil edilməsi (*get*- və *set*-metodlar);

6) biznes-məntiqi fraqmentlərinin reallaşdırılması.

Aktiv yazının nöqsanı odur ki, onun obyektlərinin strukturu VB-nin strukturundan sıx asılılıqdadır.

3. Biznes-məntiqi layının təşkili üçün predmet oblastının "mürəkkəb" modelindən istifadə edilərkən çalışırlar ki, predmet oblastının modeli VB-dən tamamilə təcrid edilsin. Çünki bu halda domen obyektlərinin VB cədvəlində təsvirinin bütün məsuliyyəti verilənlərin mənbəyi layının üzərinə keçirilə bilir. Bu halda verilənlərin mənbəyi layı biznes-məntiqi tərəfindən təşəbbüsləndirilən informasiyanın yüklənməsi və saxlanması üzrə bütün əməliyyatlara xidmət göstərir və bir-birindən asılı olmayaraq həm predmet oblastının modelini, həm də VB-nin sxemini modifikasiya etməyə imkan verir. Relyasiyalı VBİS-lər və obyektlər verilənlərin strukturlaşdırılması üçün müxtəlif mexanizmlərdən istifadə etdiklərindən, relyasiyalı VB-lərdə obyektlərin bir çox xarakteristikaları əks etdiirilmir. Bununla əlaqədar olaraq, obyekt modeli ilə relyasiyalı VBİS arasında məlumat mübadiləsini təmin etmək üçün "verilənlər dəyişdiricisi" ***(data mapper)*** patternindən (şablonundan), yəni proqram təminatı (PT) layından istifadə edilir. Çünki PT layı operativ yaddaşdakı obyektləri VB-dən ayırır.

Verilənlər dəyişdricisi funksiyasına aşağıdakılar daxildir: VB ilə obyektlər arasında verilənlərin ötürülməsi və bunların bir-birindən təcrid edilməsi; bir cədvəldə birləşdirilmiş polelərdəki siniflərin emal edilməsi; bir-neçə cədvələ müvafiq siniflərin və varis siniflərin emal edilməsi, həmçinin yüklənmiş obyektləin əlaqələndirilməsinin təmin edilməsi.

Yuxarıda sadalanan funksiyalar trivial (adi) məsələlər deyil. Verilənlər dəyişdiricisinin tərkibinə bir-sıra dar ixtisaslaşdırılmış patternlər (şablonlar) daxildir. Buna baxmyaraq bütövlükdə verilənlər mənbəyi layının layihələşdirilməsi zamanı geniş tətbiq edilir. Verilənlərin mənbəyinin "xidmətçi" tipik həlləri içərisində nisbətən geniş yayılan patternlər (şablonlar) aşağıdakılardır:

1. **iş vahidi (*unit of work*)** – bu pattern (şablon) biznes-tranzaksiyalarla əhatə edilən və VB-dəki dəyişikliklərin idarə edilməsi və paralellik problemlərinin həlli üçün nəzərdə tutulan obyektlərin siyahısını saxlayır;

2. **obyektlər kolleksiyası (*identity map*)** – bu pattern (şablon) hər bir obyektin VB-dən yalnız bir dəfə yüklənməsinə zamin durur və yüklənmiş obyekti xüsusi kolleksiyada saxlayır. Sorğu alınan kimi lazımi obyektin axtarışı kolleksiyada aparılır;

3. **tələb üzrə yükləmə (*lazy load*) –** bu pattern (şablon) hal-hazırda bütün tələb olunan verilənləri saxlamayan, lakin lazım gəldikdə yüklənən obyektlərdir;

4. **identifikasiya polesi (*identity field*)** – bu pattern (şablon) VB sətri ilə proqram əlavəsinin obyekti arasındakı uyğunluğu dəstəkləmək üçün verilənlərin yazılarının identifikatorunu saxlayır;

5. **xarici açarların əks etdirilməsi (*foreign key mapping*) –** bu pattern (şablon) VB cədvəlləri arasında xarici açara iqtibas edən obyektlər arasındakı assosiasiyaları əks etdirmək üçün nəzərdə tutulmuşdur;

6. **assosiasiya cədvəllərinin köməyi ilə əksetdirmə (*association table mapping*) –** bu pattern (şablon) assosiasiyalarla bağlanmış və xarici açarlara malik olan cədvəlləri assosiasiyalar çoxluğu şəklində saxlayır;

7. **asılı obyektlərin əks etdirilməsi (*dependent mapping*) –** bu pattern (şablon) törəmə (övlad) sinif üçün əksetdirmə səlahiyyətini müəyyən xidmətçi sinfə həvalə edir;

8. **tətbiq edilmiş qiymət (*embedded value*) –** bu pattern (şablon) obyekti digər obyektə müvafiq olan cədvəlin bir-neçə polesində əks etdirir;

9. **seriyalanmış iri obyekt (*serialized large object*)** – bu pattern (şablon) obyektlər qrafını, onları vahid iri obyekt halında seriyalaşdırmaq yolu ilə saxlayır və VB polesində yerləşdirir;

10. **bir cədvəldən varis edilmiş (*single table inheritance*) –** bu pattern (şablon) siniflərin varisliyi iyerarxiyasını bir cədvəl şəklində təqdim edir ki, bu cədvəlin sütunları iyerarxiyaya daxil olan siniflərin bütün polelərinə uyğundur;

11. **hər sinif üçün cədvəllərdən varis edilmiş (*class table inheritance*)** – bu pattern (şablon) hər sinif üçün bir cədvəldən istifadə edərək siniflərin varisliyi iyerarxiyasıdır;

12. **hər bir konkret sinif üçün cədvəllərdən varis edilmiş (*concrete table inheritance*)** – bu pattern (şablon) hər bir konkret sinif üçün bir cədvəldən istifadə edərək siniflərin varisliyi iyerarxiyasıdır;

13. **varislik dəyişdiriciləri (*inheritance mappers*)** – bu pattern (şablon) varislik iyerarxiyaları ilə işləyən dəyişdiriciləri təşkil etmək üçün nəzərdə tutulan strukturdur.

Təqdimat layı

İNSİ üçün təqdimat layının layihələşdirilməsinə başlayarkən, ilk növbədə, "varlı" və "kasıb" müştərilər arasında interfeys tipi seçilir. Varlı müştəri zəngin qrafik interfeys, cəld işləyən sistem istəyir. Bu tələblər isə müştəriyə əlavə xərc bahasına başa gəlir. Bununla əlaqədar olaraq, layihələşdirmənin birinci mərhələsində təqdimat layına kasıb müştərinin imkanları baxımından yanaşılır və yalnız müştəri bu təkliflə razılaşmadıqda varlı müştəri modelinə müraciət edilir. Təcrübə göstərir ki, korporativ proqram əlavələri layihələşdirilərkən lazımi funksiyaları icra edən kasıb müştəri modeli kifayət edir. Bununla əlaqədar olaraq, burada kasıb müştəri üçün təqdimat layının tipik patternlərinin (nümunələrinin) nəzərdən keçirilməsi məqsədəuyğun sayılmışdır.

Web-əlavələrin layihələşdirilməsi üçün nisbətən geniş yayılmış şablon "kontroller-təqdimat modeli"[62] hesab edilir. Kontroller sorğunu qəbul edir, sorğudakı informasiyanı götürüb idarəni modelin müvafiq obyektinə ötürür. Sözü gedən model predmet oblastının modeli formasında və ya tranzaksiyaların ssenarisi şəklində təqdim edilə bilər. Model verilənlər mənbəyinə müraciət edir və sorğuda nəzərdə tutulan əmliyyatı icra edir. Sonra idarə kontrollerə qaytarılır. Kontroller alınmış nəticəni təhlil edərək

[62] *Model* ***View Controller - MVC***

cavabın təqdimatı variantını seçmək haqqında qərar qəbul edir və idarəni təqdimat layına ötürür. Təqdimat layı emalın məticəsini qrafik interfeysin vasitələrinin köməyi ilə əks etdirir. Bir qayda olaraq, kontrollerlə təqdimatın qarşılıqlı fəaliyyəti birbaşa deyil, *HTTP*-seansın bu və ya digər obyekti vasitəsilə həyata keçirilir. Əksər hallarda eyni bir model üçün bir-neçə təqdimat yaradılır. Lakin modeldə dəyişiklik edildikdə həmin dəyişikliklər təqdimatlarda da edilməlidir. Bu məsələnin həlli "**müşahidəçi**" ***(observer)*** və ya "**dinləyici**" **(*listener*)** adlı tipik həllərdən istifadə etməklə mümkün olur.

Kontroller (*MVC*) modelində təqdimatın reallaşdırılması üçün aşağıdakı 3 tipik həlldən istifadə edilir:

1. **şablon üzrə təqdimat (*template view*)**. Bu, *web*-səhifənin strukturunda dinamik məzmunun yerini göstərən xüsusi markerlərin yerlşməsinə uyğundur. Bir çox instrumental platformalar (*ASP, JSP, PHP*) *web*-səhifədəki mətnə bu və ya digər ssenari dilində yazılmış proqram kodu[63] daxil etməyə imkan verir. Lakin skriptletlərdən istifadə edilməsi proqram sisteminin laylarının qarışmasına gətirib çıxarır, kodun təkrarlanmasına səbəb olur və həm obyekt yönümlü, həm də struktur-funksional metodologiya ilə dəstəklənən bir çox strukturlaşdırma vasitələrinin tətbiqini çətinləşdirir. Bununla əlaqədar olaraq, şablon üzrə təqdimatdan istifadə edilməsi hər səhifəyə köməkçi obyektin (*helper object*) təyin edilməsini nəzərdə tutur ki, bu da domenin bütün məntiqini özündə saxlayır. Bu halda səhifənin işi yalnız köməkçi obyekti çağırmaqdan ibarət olur. Səhifənin tərtibatı və proqram məntiqinin işlənib hazırlanması məsələlərinin ciddi şəkildə ayrılması qrafik dizaynerlərlə proqramçıların sərbəst surətdə işləməsinə imkan verir. Şablon üzrə təqdimat adlı tipik həllin nöqsanı odur ki, bu təqdimat istisnasız olaraq *wev*-serverlə birgə işlədiyindən, testləşdirmə çətinləşir. Bu nöqsan "dəyişdirici ilə təqdimat" adlı tipik həlldə aradan qaldırılmışdır. Çünki bu tipik həll *web*-serversiz də fəaliyyət göstərə bilir.

2. **dəyişdirici ilə təqdimat (*transform view*).** Bu, xüsusi dəyişdirici proqramdan istifadə edilməsini nəzərdə tutur. Həmin proqram domendəki verilənlərə baxır və onları *HTML* koduna çevirir. Hal-hazırda dəyişdirici ilə təqdimatların yazılması üçün istifadə

[63]skriptlet

edilən ən populyar dil funksional proqramlaşdırma dili olan *XSLT*-dir. Bu halda domendəki verilənlər *XML* formatında saxlanmalıdır. *XSLT* dili o halda daha səmərəli olur ki, *web*-saytda tez-tez və qlobal dəyişikliklər etmək tələb olunur.

3. **iki mərhələli dəyişdirmə (*two step view*).** Bu, iki mərhələdən istifadə edilməsini nəzərdə tutur, belə ki, əvvəlcə domen verilənləri əsasında məntiqi struktur[64] formalaşdırılır, sonra bu struktur *HTML* koduna transformasiya edilir (çevrilir). İkinci mərhələ metodlarına bəlli olur ki, məntiqi struktura hansı elementlər daxildir və onları necə vizuallaşdırmaq olar. Bu halda, lazım gəldikdə, *web* -saytın xarici görünüşünün qlobal dəyişilməsi yalnız ikinci dəyişdirmə mərhələsində həyata keçirilə bilir. Bu, qrafik interfeysin yenidən işlənməsinin münasibliyi baxımından iki mərhələli dəyişdirmə adlı tipik həllin səmərəliliyini artırır. Çox hallarda iki mərhələli dəyişdirmə müxtəlif müşahidəçilərin[65] köməyi ilə *web*-kontentə[66] müraciəti təmin etmək üçün və ya bir *web*-saytın xarici görünüşünün bir-neçə variantının[67] dəstklənməsi zəruri olduqda istifadə edilir.

MVC-modeli kontroller *HTTP*-sorğuların emalını və dispetçerlənməsini icra edir. Aşağıdakı 2 tipik həll daha geniş yayılmışdır:

1.“səhifələrin kontrolleri” **(*page controller*)** patternindən (nümunəsindən) istifadə edilərkən *HTTP*-sorğuların emalı məsələləri və onun istiqamətinin dəyişdirilməsi ilə bağlı qərarların qəbulu bir-birindən ayrılır. Səhifələrin kontrollerinin funksiyaları aşağıdakılardan ibarətdir:

a. sorğunun icrası üçün məlumat toplanması: *URL*-in təhlili və istifadəçi tərəfindən müvafiq formada daxil edilmiş verilənlərin çıxarılması;

b. model obyektləərinin yaradılması və onların verilənlərin emalı üçün lazım olan metodlarının çağırılması. Bu zaman bütün lazımi verilənlər *HTTP*-sorğulardan modelə ötürülməlidir;

c. modeldən alınmış istifadə üçün lazımi informasiyanın təqdimatının təyin

---

[64] daxiletmə polesi, yuxarı və aşağı kolontitullar, cədvəllər, açarlar və i.a. daxil olmaqla
[65] Обозреватель м 1. müşahidəçi, seyrçi; müşahidə edən, seyr edən; 2. icmalçı, icmal yazan.
[66] məzmuna
[67] məsələn, stasionar və “mobil” versiyalarının

edilməsi.

Səhifələrin kontrolleri ssenari şəklində (*CGI*, servlet və i.a.) və ya server səhifələri (*ASP, PHP, JSP* və i. a.) formasında hazırlana bilər. Server səhifəsindən istifadə adətən səhifələrin kontrolleri və şablon üzrə təqdimatın bir faylda birləşdirilməsini nəzərdə tutur.

Bir qayda olaraq, səhifələrin kontrolleri kifayət qədər sadə məntiqə malik olan saytlar üçün tətbiq edilir. Çünki *web*–saytın hər bir məntiqi səhifəsi üçün ayrıca kontrollerin olmasını nəzərdə tutur. Əgər saytın strukturu mürəkkəbdirsə, həmçinin təhlükəsizliyin yoxlanması, beynəlmiləlliyin təmin edilməsi və ya xüsusi istifadəçilər kateqoriyasından ötrü xüsusi təqdimatların açılması üçün xüsusi tələblər qoyulursa, onda səhifələrin kontrolleri kodun təkrarlanmasına gətirib çıxarır. Kodun təkrarlanmasını aşağıdakı iki üsulla aradan qaldırmaq olar: 1) kontrollerlərin ümumi məntiqinin layın super tipini təşkil edən xidmətçi siniflərə daxil etməklə; 2) bilavasitə sorğular kontrolleri adlı tipik həlldən istifadə etməklə.

3.Sorğular kontrolleri (***front controller***) bütün sorğuların emalı üçün nəzərdə tutulan vahid obyektdən istifadə edilməsini nəzərdə tutur ki, bu da aşağıdakı 2 hissədən ibarət olur:

a. *web*-emaledici – *web*-serverdən daxil olan *POST*-sorğuların və *GET*-sorğuların faktiki alınmasını icra edən obyektdir. Bu obyekt *URL*-dən lazımi informasiyanı və sorğunun giriş verilənlərini çıxarır, sonra əmələin tşəbbüsləndirilməsi barədə qərar qəbul edib icrasını müvafiq əmrə həvalə edir.

b. *web*-emaledici daxil olan sorğunu emal etmək üçün lazımi əmri əmrlər iyerarxiyasından statik və ya dinamik qaydada seçir. Əmrin statik seçilməsi *URL* ünvanın sintaksis təhlilini aparmağı və şərti məntiq tətbiq etməyi, dinamik seçim isə *URL* ünvanın müəyyən standart fraqmentini ayırmağı və əmrlər sinfinin nüsxəsinin dinamik yaradılmasını nəzərdə tutur.

Web-emaledici alınmış *URL* ünvanını interpretasiya (şərh) edir və sorğuya xidmət etmək üçün ayrıca obyekt yaradır. Beləliklə, bütün *HTTP*-sorğuların işlənməsi üzrə fəaliyyətin vahid obyekt çərçivəsində mərkəzləşdirilməsi mümkün olur ki, bu da saytın

funksionalı modifikasiya edildikdə *web*- serverin konfiqurasiyasını dəyişmək zərurətini aaradan qaldırır. Bundan əlavə, sorğular kontrolleri "tutucu[68] filtr" adlı tipik həll ilə yaxşı uyuşur. Həmin filtr (süzgəc) özünü dekorator (bəzəkçi) kimi təqdim edir. Çünki *web*-emaledicinin örtüyü rolunu oynayır. Həmin örtük filtrlər zənciri yaratmağa imkan verir. Filtrlər zənciri saytda qeydiyyat, əslləşdirmə[69] və kodlaşdırma proseslərini emal etmək üçündür. "Təqdimat-kontroller" cütünün qarşılıqlı fəaliyyətinin təmin edilməsi üçün tipik həllin seçilməsi istifadə edilən instrumental mühitdən çox asılıdır. Məsələn, *Visual Studio*-dan istifadə edilərkən çox hallarda "səhifələrin kontrolleri-şablon üzrə təqdimat" bağlantısından, *Struts(Java)* mühitindən istifadə edilərkən isə "sorğular kontrolleri-şablon üzrə təqdimat" bağlantısından istifadə edilir.

Səhifələrin kontrolleri və sorğular kontrolleri giriş kontrollerləri adını almışdır. Bu kontrollerlər onlarla səhifəlik web-saytların layihələşdirilməsi zamanı tətbiq edilir. Layihələşdirilən sayt 30 və daha artıq səhifədən ibarətdirsə, lazımi ekranın seçilməsi məntiqi bir-neçə kontrollerdə təkrarlanır ki, bu da qətiyyən arzuolunmaz haldır. Bu halda giriş kontrollerləri ilə yanaşı əlavə servis layı, yəni proqram əlavəsi kontrolleri **(*application controller*)** yaradılır.

Proqram əlavəsi kontrolleri 2 əsas funksiya icra edir: 1) konkret situasiyada tətbiq edilən domenin məntiqini seçir; 2) sorğuya cavabın əks etdirildiyi təqdimatı seçir. Bu 2 funksiyanı icra etmək üçün proqram əlavəsi kontrolleri siniflərə iki iqtibas kollek-siyasını dəstəkləyən olmalıdır: biri domen layında icra edilən əmrlər üçün, digəri də təqdimatlar üçün.

Domen əmri olaraq proqram əlavəsi kontrolleri layının hissəsi olan əmr obyektləri və ya tranzaksiya ssenarilərinə iqtibaslar, yaxud da domen obyektlərinin metodları çıxış edə bilər. Əgər təqdimatlar qismində server səhifələrindən istifadə edilərsə, onda təqdimatın çağırılması üçün müvafiq səhifənin adı tətbiq edilə bilər.

---

[68] **перехватывающий фильтр** - перехватать сов. 1. tutmaq, yaxalamaq, ələ keçirmək (hamısını); 2. qapmaq, qamarlamaq, qapışdırmaq; 3. əlləmək, əl vurmaq (hər yerinə).

[69] аутентификация - аутентичный прил. , mötəbər, eyni qüvvəyə malik, əslinə uyğun, əslinə mütabiq; аутентичный текст əslinə uyğun tekst.

Predmet oblastının modeli

Predmet oblastının modeli aşağıdakı prinsiplər əsasında yaradılmalıdır:

1. Konseptual siniflər və onların atributlarının adlandırılması zamanı lüğətdən istifadə edilməlidir;

2. Predmet oblastının modelində tələblərə və baxılan problemə dəxli olmayan anlayışlar olmamalıdır.

Predmet oblastının modelini qurarkən nisbətən geniş yayılmış səhv hər hansı obyekti atributlara və ya konseptual siniflərə aidetmə zamanı buraxılır[70]. Bu cür səhvdən qorunmaq üçün aşağıdakı qaydadan istifadə edilir: əgər obyekt reallıqda ədəd və ya mətn deyildirsə, deməli, çox güman ki, atribut deyil, konseptual sinifdir.

3. Elementlərin təsviri və ya bu obyektlərin konkret nüsxələrinin mövcudluğundan asılı olmayan xidmətlər vacib olduqda və həmin nüsxə silindikdə qiymətli informasiya itkisi baş verdikdə proqram təminatının modelində spesifikasyadan istifadəyə üstünlük verilir. Məsələn, əmtəə barədə olan: *id, sn, description, price,…* kimi məlumatlar bir yazıda saxlanarkən, yazını silmə zamanı əmtəənin qiyməti barədəki informasiya itkisi baş verir. Belə itkiyə yol verilməməsi üçün spesifikasiyaya 2 sinif və əmtəə daxil etmək kifayətdir. Bu halda əmtəənin silinməsi onun qiymətinin silinməsinə səbəb olmur.

Proqram təminatının modelinin yaradılması iterativ prosesdir. Layihələşdirici gerçək dünya mahiyyətlərini təhlil edib onları proqram siniflərində necə təsvir etmək barədə qərar qəbul edir.

Beləliklə, birinci iterasiyada istifadə variantları, ardıcıllıqlar, fəaliyyət və i.a. diaqramlarının köməyi ilə tələblər modeli, ikinci iterasiyada predmet oblastının modeli, üçüncü iterasiyada isə layihələşdirmə modeli qurulur.

Atributların əlavə edilməsi

Predmet oblastının modelinə yalnız müvafiq tələblərlə[71] təyin edilmiş atributlar

---

[70] Misal: "reysin təyinat yeri (məntəqəsi) (*Destination*) "uçuş" (*Flight*) anlayışının, yoxsa "aeroport" obyektinin atributudur?"- sualına necə cavab verilməlidir?

[71] texniki tapşırıqda, spesifikasiyada, istifadə variantları (nümunələr) modelində göstərilən tələblər nəzərdə tutulur.

daxil edilir. Predmet oblastının modelində atribut qismində əsasən sadə tiplərdən[72] istifadə edilir. Layihələşdirmə və reallaşdırma mərhələlərində isə atribut qismində siniflərarası assosiasiyalar yaratmaq məqsədi ilə digər siniflərə iqtibaslardan, siyahılardan, lüğətdən və s. istifadə edilə bilir. Əgər obyektin xassəsi (parametri) predmet oblastının modelində sadə tip kimi təqdim edilə bilirsə, onda onu modelə atribut kimi daxil edirlər, əks halda, modelə əlavə konseptual sinif daxil edilir.

Layihələşdirmə patternləri (şablonları)

Predmet oblastının modelini qurub sınaqdan keçirdikdən sonra əməliyyatların təhlili və konseptual siniflər arasında düşünülmüş şəkildə yerləşdirilməsi məsələləri həll edilir. Bu məqamdan başlayaraq layihələşdirici layihələşdirmə patternlərindən (şablonlarından) fəal istifadə etməyə başlayır. Patternlər (şablonlar) dedikdə, sinfin müəyyən məsələsini həll etmək məqsədi ilə birləşdirilmiş struktur vahidlər, geniş mənada isə layihələşdirmə qaydaları və standart layihə həlləri nəzərdə tutulur ki, bunlar da layihələşdirmənin sürətləndirilməsi və keyfiyyətinin yüksəldilməsi üçün istifadə edilir.

90-cı illərin ortalarından proqramlaşdırıcılar arasında belə bir fikir meydana çıxmışdır ki, hansı layihə, hansi proqramlaşdırma dilindən istifadə edilməklə və hansı yanaşma tətbiq edilməklə hazırlanır-hazırlasın, mövcud layihələr bu və ya digər şəkildə modifikasiya edilib yeni yaradılan layihələrdə uğurla istifadə edilir. Çünki standart layihə həlləri heç nədən asılı olmayaraq bir-birinə bənzəyir. Bu layihə həlləri şablon (pattern) adlandırılmışdır. Bu şablonların (patternlərin) nisbətən geniş yayılmış 23-ü 1994-cü ildə *Addison Wesley* nəşriyyatında *«Design Patterns. Elements of Reusable Object-Oriented Software»* adlı kitabda[73] nəşr edilmişdir. 2001-ci ildə *Willey* nəşriyyatında *Pattern-Oriented Software Architecture. A System of Patterns V.1*. kitabı (I cild), 2007-ci ildə isə 5 cildlik ensiklopediya[74] nəşr edildi. Bundan əlavə, şablonlar barədə *http://www.hillside.net/patterns*. saytına da baxmaq olar.

---

[72] ədəd, sətr, tarix, məntiqi tip
[73] Э.Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес adlı peşəkar müəlliflər tərəfindən hazırlanmışdır.
[74] *On Patterns and Pattern Languages.*

Hər bir patterndə (şablonda): ad, həll edilən məsələnin formulirovkası[75], həll üsulunun təsviri, üstün və nöqsan cəhətləri, digər patternlərlə əlaqəsi göstərilir.

Bir sıra ədəbiyyatlarda şablonlar səviyyələr üzrə bölünür:

1. İrimiqyaslı layihə həlləri ilə əlaqəli olan arxitektura şablonları. Bunlara misal olaraq *MVC* həllərindən biri olan *Layers* şablonu göstərilə bilər. Arxitektura şablonlarına həsr edilmiş ədəbiyyatlarda, bir qayda olaraq, bütövlükdə proqram əlavəsi üçün predmet yönümlü həllər təsvir edilir. Məsələn, arxitektura şablonları yalnız varlı və ya kasıb müştəriləri olan korporativ proqram əlavəsinin reallaşdırılması üçün nəzərdə tutulmur. Bu şablonlar həm də *OS*, qərar qəbulunu dəstəkləyən sistemlər (QQTS-lər), avtomatlaşdırılmış layihə həlləri sistemləri, VBİS-lər və s. kimi xüsusi təyinatlı proqram əlavələrinin reallaşdırılması üçün nəzərdə tutulur ki, buraya da *Eric Evans Domain-Driven Design: Tackling Complexity in the Heart of Software* şablonları, həmçinin korporativ əlavələrlə inteqrasiya edilmiş şablonlar[76] aid edilir.

2. Layihələşdirmə şablonları orta səviyyə şablonlarıdır. Bunlar paket (alt sistem) səviyyəsində layihələşdirmə qaydalarını təsvir edir. Bu qrupa *GRASP* və *GoF* şablonlarının əksəriyyəti aid edilir. Bu səviyyə üçün də əsas ədəbiyyat Гамма Э., Хелм Р., Джонсона Р., Влисседес Дж. «Приемы объектно-ориентированного проектирования: паттерны проектирования» kitabıdır ki, burada da *GoF*-patternlər adlanan 23 təməl şablon təsvir edilmişdir. Bu patternlər Stiven Sterlinq və Olav Maasen tərəfindən nəşr edilmiş «Применение шаблонов *Java*. Библиотека профессионала» kitabında da təsvir edilmişdir. Birinci və ikinci səviyyələr arasında aralıq mövqe Martin Faulerin «Архитектура корпоративных программных приложений» kitabında *web*-interfeysli korporativ proqram əlavələrinin layihələşdirilməsində geniş istifadə edilən 40 pattern (şablon) nəzərdən keçirilmişdir.

---

[75] формулировка ж 1. qısaca və dürüst ifadə etmə (edilmə); xülasə etmə (edilmə); 2. formul, xülasə edilmiş fikir.

[76] Грегор Хоп, Бобби Вульф «Шаблоны интеграции корпоративных приложений» Bunlara *GoF* şablonları da deyilir. *G*-Greqor sözünün birinci hərfi, *o*-müəlliflərin adlarında daha çox iştirak edən hərf, *F*-Vulf sözünün son hərfidir.

3. İdiomlar[77]. Bunlar proqramlaşdırma dilləri və ya reallaşdırma stilləri (üslubları) ilə əlaqəli aşağı səviyyəyə aid layihə həlləridir. Buna misal olaraq, bu və ya digər sinfin yalnız bir nüsxəsinin yaradılması qaydasını təsvir edən *Singleton* şablonu (patterni) göstərilə bilər. Bu qrupa aid 77 pattern Kent Bek tərəfindən nəşr etdirilmiş «Шаблоны реализации корпоративных приложений» kitabında sistemləşdirilmiş və təsvir edilmişdir. Sözü gedən patternlər: sinif, vəziyyət, davranışlar, metodlar və kolleksiyalar kimi 5 qrupda birləşdirilmişdir.

Patternlərdən istifadə edilməsi İNSİ-lərin layihələşdirilməsinin bir-neçə fundamental (təməl) məsələsini, o cümlədən, "zəif xarici, güclü daxili əlaqə" prinsipinin ödənməsinin təmin edilməsi, həmçinin komponentlərin təkrar istifadə edilməsi məsələlərini həll etməyə imkan verir.

Komponentlərin təkrar istifadə edilməsini, həmçinin sistemin funksionallığının və həmin səhvləri düzəldən *GoF*-patternlərin artırılmasını çətinləşdirən səhvlər aşağıdakılardan ibarətdir:

• bir-biri ilə sıx əlaqəli siniflər monolit altsistemlər yaratdığından sinifləri modifikasiya etmədən heç nəyi dəyişmək və ya silmək mümkün deyildir. Xarici əlaqəliliyi azaltmaq üçün laylardan, abstrakt əlaqələrdən, həmçinin aşağıdakı patternlərdən istifadə edilir: **abstrakt fabrika (*abstract factory*), körpü (*bridge*), vəzifələr zənciri (*chain of responsibility*), əmr (*command*), fasad (*facade*), vasitəçi (*mediator*), müşahidəçi (*observer*).** Alt sistemin ayrı-ayrı siniflərinin bir-birindən asılı olmamasının yüksəldilməsi üçün: **adapter (*adapter*), dekorator (*decorator*), müştəri[78] (*visitor*).**

• obyektin yaradılması zamanı sinfiin adından istifadə edilməsi layihələşdiricini interfeysə deyil, konkret reallaşdırmaya (layihə həllinə) bağlayır ki, bu da gələcəkdə obyektlərin dəyişilməsini çətinləşdirə bilir. Bu problemin həlli **abstrakt fabrika, fabrik metodu (*factory method*), prototip (*prototype*)** patternlərinin köməyi ilə dolayı obyektlər yaratmaqla mümkün olur;

---

[77] Идиом м dilç. idiom (bir dilə xas olan və mənası tərkib hissələrini təşkil edən sözlərin ayrı-ayrılıqda götürülən mənası ilə düz gəlməyən sabit söz birləşməsi, məs.: rusca „баклуши бить" yaxud azərbaycanca "gözünün odunu almaq").

[78] Посетитель м 1. müştəri; 2. gələn; görüşə, tamaşaya, kitabxanaya və s. gələn adam.

• konkret əməliyyatdan istifadə edilməsi sorğunun icrasını yeganə bir üsulla məhdudlaşdırır. **Vəzifələr zənciri və əmr** ptternləri imkan verir ki, sorğuları koda daxil etmədən sorğunun cavablandırılması həm kompilyasiya, həm də icra mərhələlərində dəyişdirilə bilsin;

• **abstrakt fabrika** və **körpü** patternləri konkret platformadan[79] asılılığı məhdudlaşdırır;

• müştəri qurularkən bu və ya digər obyektin necə təqdim edildiyi, saxlandığı və ya reallaşdırıldığı barəədə informasiyadan istifadə edilərsə, obyekt dəyişikliyə uğradıqda müştərinin də dəyişilməsi tələb oluna bilir. Sadalanan informasiyanın inkapsulyasiya edilməsi[80] **abstrakt fabrika, körpü, saxlayıcı[81] (*memento*), müavin (*proxy*)** patternlərinin köməyi ilə həyata keçirilir;

• genişləndirmə, optimallaşdırma və s. zərurəti ilə bağlı olaraq dəyişən alqoritmlərdən asılı olan obyektlər təzədən layihləşdirilməli olur. **Körpü, iterasiya edici (*iterator*), strategiya (*strategy*), şablon metodu (*template method*) və müştəri** kimi patternlər alqoritmləri dəyişilməkdən yüksək ehtimalla izolə (təcrid) edir;

• obyektlərin kompozisiyası[82] (quruluşu) nümayəndə göndərilməsi (делегирование) – bunlar davranışları kombinə etmək (birləşdirmək) üçün varisliyin çevik alternativləridir. Çünki bu halda yeni funksionallıq sistemə varislik vasitəsilə yeni alt siniflər daxil etməklə deyil, obyektlərin kompozisiyasında dəyişiklik etməklə əlavə edilir. Lakin təkcə obyektlərin kompozisiyasından istifadə etmək layihənin strukturunu əhəmiyyətli dərəcədə mürəkkəbləşdirə bilər. Bununla əlaqədar olaraq belə bir qaydadan istifadə edilir: əvvəlcə alt sinif müəyyən edilir, sonra tələb olunan ixtisaslaşmanı təmin etmək üçün onun obyektlərini mövcud obyektlərlə kombinələşdirirlər. Bu qayda öz əksini aşağıdakı patternlərdə tapmışdır: **körpü, vəzifələr zənciri, yerbəyeredici**

---

[79] *API*-də təqdim edilən əməliyyat sistemi nəzərdə tutulur

[80] İnkapsulyasiya- atributların, parametrlərin bir kapsul içinə yığılması

[81] хранитель м 1. saxlayan, mühafizə edən, qoruyan; 2. mühafiz (muzeylərdə, kitabxanalarda və s. bəzi vəzifələrin adı).

[82] Композиция ж kompozisiya (1. musiqi əsərləri yaratma nəzəriyyəsi; класс композиции kompozisiya sinfi; 2. musiqi əsəri, musiqi yaradıcılığı; 3. quruluş, tərtib; композиция романа romanın kompozisiyası; 4. xəlitə).

(компоновщик) (*composite*), dekorator (*decorator*), müşahidəçi, strategiya.

Sistem sinifllrinin daxil edilməsi ilə siniflərin vəzifələrinin təyin edilməsindən, həmçinin *GRASP* və *GoF* patternlərinə müvafiq olaraq siniflərin interfeyslərini nəzərdən keçirdikdən sonra daha yüksək abstraksiyaların, yəni İNSİ-nin arxitektura laylarının təsvirinə keçilir.

İndi tanış olacağımız patternlərin birinci qrupu əməliyyatların paylanması üçün, daha geniş mənada isə siniflər arasındakı vəzifələr üçün nəzərdə tutulmuşdur.

Vəzifələr və əməliyyatlar bir-birinə qarşı tamın hissəyə olan münasibətindədir. Vəzifələr bir-neçə metodu özündə birləşdirir ki, bunlar da reallaşdırıldıqda obyektdə və ya qarşılıqlı əlaqəli obyektlər yığımında müəyyən məqsədə çatmağa imkan verir.

İlk yanaşmada 2 tip vəzifəni ayırmaq olar: əməl (*doing*) (müəyyən əməliyyatların təşəəbbüsləndirilməsi, icrası və icranın idarə edilməsi), bilik (*knowing*) (inkapsulyasiya edilmiş verilənlər, əlaqəli obyektlər, hesablanmış kəmiyyətlərin qiymətləri və s. barədəki informasiyanın saxlanması).

Siniflər arasındakı vəzifələrin paylanması üçün nəzərdə tutulan qrupa aid olan patternlər *GRASP*-patternlər (*General Responsibility Assignment Software Patterns*) adını almışdır. *GRASP*-patternlər arasında aşağıdakılar daha geniş yayılmışdır:

1) informasion ekspert (*Information Expert*) – bu və ya digər funksional vəzifələrin təyin edilməsi üçün nəzərdə tutulan bir-neçə namizədlər arasından sinfin seçilməsi qaydası;

2) yaradıcı (*Creator*) – müəyyən şərtlərin yerinə-yetirilməsi zamanı B sinfinə A sinfinin nüsxəsini yaratmaq vəzifəsini təyin etməyə imkan verən qayda;

3) kontroller (*Controller*) – sistem xəbərlərinin emalediciləri adlı xüsusi (xidmətçi) sinfin yaradılması.

İNSİ-nin universallığının və/və ya gələcəkdə genişləndirilməsinin təmin edilməsi üçün nəzərdə tutulan polimorfizm (*Polymorphism*), xalis sintetika (*Pure Fabrication*), istiqamətdəyişmə (*Indirection*), qorunmuş variasiyalar (*Protect Variation*) kimi əlavə *GRASP* şablonlarından da istifadə edilir.

İnformasion ekspert (*Information Expert*) şablonu

Bu şablon informasion ekspertə, yəni vəzifələrin yerinə-yetirilməsi üçün tələb olunan informasiyaya malik olan sinfə vəzifələr təyin etməyə imkan verir. İnformasion ekspert patterni hər bir obyektə müvafiq obyekt yönümlü proqramlaşdırmannın məlum qaydasını təhkim edir ki, bu qaydaya uyğun olaraq həmin obyekt özündəki informasiya ilə bağlı olan əməli icra edir.

Vəzifələr düzgün paylandıqda sistemin başa düşülməsi, dəstəklənməsi və genişlən-dirilməsi, həmçinin ayrı-ayrı komponentlərin təkrar istifadə edilməsi sadələşir.

Vəzifələr dəqiq dürüst ifadə (формулировка) edilərək paylanmalıdır.

Sonra layihələşdirmə modeli (əgər varsa) və ya predmet oblastının modeli təhlil edilir və informasion ekspert rolunda çıxış etmək üçün daha çox uyğun gələn konseptual sinif aşkar edilir. Məsələn, satışın ümumi dəyərini hesablamaq üçün bütün satılmış malların dəyərlərini cəmləmək lazımdır (*SalesLineItem* – sadəcə nüsxə deyil, *list* tipli kolleksiya obyektidir). Bu zaman hər bir mal üzrə satış məbləğini hesablamaq üçün onun qiymətini satılmış malların miqdarına vurmaq lazım gəlir. Satışın ümumi dəyərini hesablamaq üçün lazımi informasiya tərkibində *getTotal* metodu olan *Sale* obyektinə əlçatandır. Ümumi dəyəri hesablamaq üçün verilmiş konkret malın aralıq satış məbləği müəyyən edilməlidir. Satılmış malların miqdarı barədə informasiya *SalesLineItem* sinfinə məxsus olduğundan, *Expert* şablonuna müvafiq olaraq həmin sinfin tərkibinə *getSubTotal* metodu daxil edilmişdir ki, bu da öz növbəsində malın qiymətini götürmək üçün *ProductSpecification* obyektinə müraciət edir. Beləliklə, *getTotal, getSubTotal* və *getPrice* kimi 3 metod informasion ekspert (*Information Expert*) patterninə müvafiq olaraq 3 sinif arasında paylanmışdır.

Siniflərin metodları qarşılıqlı fəaliyyət diaqramlarının təhlili yolu ilə mmüəyyən edilir. Belə ki, əgər *getPrice()* xəbəri *ProductSpecification* sinfinin nüsxəsinə ötürü-lürsə, onda *ProductSpecification* sinfində *getPrice* metodu təyin edilməlidir.

Bir qayda olaraq, vəzifələrin paylanması qarşılıqlı fəaliyyət diaqramlarının yaradılması prosesində icra edilir və sonar alınmış nəticələr yeni metodlar şəklində siniflər diaqramlarında əks etdirilir.

Yaradıcı (*Creator*) pattern (şablon)

Bu, "B sinfi A obyektlərini aqreqasiya edirsə"[83], "B sinfi A obyektlərini saxlayırsa", "B sinfi A sinfinin obyektlərinin nüsxələrini yaddaşa yazırsa", "B sinfi A obyektlərindən intensiv şəkildə istifadə edirsə", "B sinfi təşəbbüsləndirici verilənlərə malikdirsə və bunları A obyektləri yaradılarkən onlara ötürürsə" kimi şərtlərdən hər hansı biri ödəndikdə, A sinfinin nüsxələrini yaratmaq üçün B sinfinə vəzifə təyin etməyə imkan verir.

Sale sinfi *SalesLineItem* obyektlərinin bir-neçə nüsxəsini saxladığından, *Creator* şablonuna uyğun olaraq *Sale* sinfi *SalesLineItem* obyektlərinin nüsxələrini yaratmaq imkanına malik olmalıdır. Bu cür məsələni həll etmək üçün *Sale* obyektində *makeLineItem()* metodu təyin edilməlidir.

İstifadəçi serverə yeni fayl yükləmək istədikdə sistemdə «Файл УММ» sinfi peyda olmalıdır. Yükləmələr spesifikasiyası da faylın serverə uğurlu yüklənməsi zamanı yaradılır (əlavə edilir). Bunun üçün «Файл УММ» obyektində *create()*, "Yükləmələr spesifikasiyası"[84] obyektində isə *createLineItem()* metodu nəzərdə tutulmalıdır.

Nüsxə yaradılması kənar xassələrdən asılı olan bəzi şərtlərin reallaşdırılması zamanı icra edilən mürəkkəb əməliyyat olduqda nüsxə yaradılması üçün *Factory* şablonundan istifadə edilir.

"Zəif xarici əlaqə" (*Low Coupling*) və "Güclü daxili əlaqə" (*High Cohesion*) şablonları

Sistemin arxitekturasının layihələşdirilməsində əsas məsələ aşağıdakı iki prinsipin gözlənilməsidir:

1) "Zəif xarici əlaqə" (*Low Coupling*) – yəni, ayrı-ayrı alt sistemlər (siniflər) arasındakı əlaqələrin sayı minimal olmalıdır;

2) "Güclü daxili əlaqə" (*High Cohesion*) – yəni, hər bir alt sistemin daxilindəki

---

[83] iriləşdirirsə
[84] «Спецификация загрузок»

ayrı-ayrı hissələr arasındakı əlaqələrin sayı maksimal olmalıdır.

Əgər sinif yüksık dərəcədə əlaqəlidirsə, bu o deməkdir ki, hhəmin sinif çoxsaylı başqa siniflərdən asılıdır. Deməli:

- əlaqəli siniflərdə baş verən dəyişikliklər bu sinifdə lokal dəyişiklik törədir;
- hər bir ayrıca sinfin başa düşülməsi çətinləşir ki, bu da təkrar istifadəni mürəkkəbləşdirir;
- sistemin funksonal imkanlarının genişləndirilməsi mürəkkəbləşir.

Daxili bağlılığı ölçmək üçün bağlılıq gücü (CC[85]) anlayışından istifadə edilir.

Bağlılıqın 7 tipi vardır:

1. Üst-üstə düşmə ilə bağlılıq (CC=0). Sinfin əməliyyatlarında əyani görünən daxili əlaqələr yoxdur.

2. Məntiqi bağlılıq (CC=1). Sinfin əməliyyatları bir-biri ilə funksional bənzərlik prinsipi üzrə birləşmişdir. Məsələn, sinif müxtəlif səhvləri emal etməyə vəkil edilmişdir.

3. Müvəqqəti bağlılıq (CC=3). Sinfin əməliyyatları bağlı deyillər, lakin sistemin eyni bir zaman kəsiyindəki işində lazımdırlar.

4. Prosedur bağlılıq (CC=5). Sinfin əməliyyatları onların icra edilmə qaydası ilə bağlıdır. Bu qayda müəyyən davranış alqoritmi (ssenarisi) ilə reallaşdırılır.

5. Kommunikativ bağlılıq (CC=7). Əməliyyatlar verilənlər üzrə bağlıdırlar (eyni bir verilənlər strukturu ilə işləyirlər).

6. İnformasion (ardıcıl) bağlılıq (CC=9). Bir metodun qaytardığı nəticələr nüsxənin digər metodlarında arqument kimi istifadə edilir.

7. Funksional bağlılıq (CC=10). Sinfin atributları və əməliyyatları verilmiş davranışı və ya konkret vəzifələrin təmin edilməsi məqsədi ilə bir-biri ilə qarşılıqlı fəaliyyət göstərirlər.

Qeyd etmək lazımdır ki, 1,2,3 bağlılıqları siniflər arasında vəzifələrin düzgün paylanmamasının nəticəsidir (4-cü tip bağlılıq isə arzuolunmazdır).

“Zəif xarici, güclü daxili əlaqə” prinsipi İNSİ-nin layihələşdirilməsinin təməl prinsipi olmaqla struktur-funksional metodologiyanın əsasını təşkil edir və modul və

[85] CC - силы связности

onun funksiyası anlayışları üçün təyin edilmişdir. Siniflər paketlər (modullar) üzrə yerləşdirilərkən sözü gedən prinsiplərə əməl olunmalıdır.

Qeyd etmək lazımdır ki, "güclü daxili əlaqə" prinsipinin bəzi hallarda pozulması da məlumdur. Məsələn,

1. «insan amili». Əgər layihədə çoxsaylı siniflər SQL operatorları vasitəsilə iri VB-yə müraciət edirsə, onda SQL üzrə mütəxəssisi dəstəkləmək üçün arxitektor onları bir sinifdə birləşdirir;

2. paylanmış sistemlərdə şəbəkə düyünləri arasında qısa xəbərlər (*setName, readFIO* və s.) şəklində intensiv mübadilə məhsuldarlığı azalda bilir. Bu problemi məsafədən müraciət edilən interfeysdən (*remote facade*) istifadə etməklə həll etmək olur. Çünki bu halda bir iriləşdirilmiş əməliyyat (*setData*) icra edilir.

Kontroller *(Controller)* şablonu

Bu, sistem xəbərlərini emal etmək vəzifəsini xidmətçi sinfə həvalə edir. Xidmətçi sinif dedikdə, bütövlükdə sistem (alt sistem), yaxud bu və ya digər pesedentin[86] ssenarisi üçün (<Прецедент>*Handler*, <Прецедент>*Coordinator*, <Прецедент> *Session*) kimi sistem hadisələrinin emalını təmin etmək üçün zəruri olan struktur başa düşülür. Sistem hadisəsi (xəbəri) dedikdə, baxılan sistemə və ya mənbə alt sisteminə nəzərən xaricdən yüklənən yüksək səviyyə hadisəsi nəzərdə tutulur. Kontroller sistem əməliyyatlarının icrası üçün metodlar təyin edir. Bu funksiyaları digər obyektlərə həvalə edir, onların fəaliyyətlərini əlaqələndirir (koordinasiya edir), lakin özü sərbəst surətdə funksiya icra etmir. Kontroller predmet oblastının obyekti deyil, bu, sistemin həyat fəaliyyətini dəstəkləyən xidmətçi sinifdir. Məsələn, istifadəçi formada «Найти» (axtar) düyməsini sıxırsa, o, bütövlükdə sistemə müraciət edir, yəni sistem əməliyyatını təşəbbüsləndirir. Sonra istifadəçi ilə qarşılıqlı fəaliyyət layının hansısa sinfi xəbəri kontroller obyektinə: *ID* düyməsinə və sorğu mətninə göndərir. Kontroller isə bu parametrləri icraçı sinfə göndərir. Bu halda, yəni istifadəçi interfeysindən sorğu

[86] прецедент м presedent; misal (keçmişdə olmuş və sonra olan eyni tipli hadisələr üçün misal ola biləcək hadisə və ya hərəkət)

proqramın müştəri hissəsinə aid olan lokal sinfə-kontrollerə ötürülür. Kontroller sinfi bütövlükdə sorğunu və ya onun bir hissəsini aralı xidmətlərə (şöbələrə) göndərir ki, bunu nəticəsində də istifadəçi interfeysi ilə aralı şöbələr arasında bağlılıq dərəcəsi azaldılır.

Əgər interfeys sinfi sistem əməliyyatını emaal edərsə, onda domenin məntiq hissəsi interfeysin əsirinə çevrilər ki, bunun da nəticəsində komponentlərin təkrar istifadə edilməsi əhəmiyyətli dərəcədə azalar.

Sistemin davranışının təhlili prosesində identifikasiya edilmiş ssistem əməliyatları layihələşdirmə mərhələsində bir və ya bir-neçə kontrollerin təərkibinə daxil edilir. Bununla əlaqədar olaraq kontroller domen layı ilə, məsələn, istifadəçi interfeysi layı arasında interfeys kimi başa düşülə bilir. Kontrollerlərin olduqca çoxsaylı vəzifələrlə yüklənməsi onların hazırlanması zamanı buraxılan tipik səhvdir. Kontrollerlərin tətbiqi komponentlərin təkrar istifadə edilməsini və həmçinin sistem hadisələrinin ardıcıl emalı üçün tələb olunan bu və ya digər presedentin təmin olunmasını sadələşdirir. Məsələn, istifadəçi yeni kitab əlavə edərkən formada: müəllif, ad, annotasiya, faylın titulla birgə yerləşməsi, kitab faylının yerlşməsi və i.a. kimi polelər doldurmalı olur. Bu polelərin hər biri üçün onları VB-yə daxil (əlavə) etməkdən ötrü ayrıca metod mövcud ola bilər, lakin həmin metodların çağırılması qaydası kontroller vasitəsilə həyata keçirilir. Beləliklə, kontroller interfeys ekranlarının və ya proqram əlavəsinin funksiyaları axınlarının mərkəzləşdirilmiş qaydada idarə edilməsi vasitəsi kimi istifadə edilə bilir. Sistem əməliyyatları az olduqda bir kontroller – “Proqram əlavəsi kontrolleri” kifayət edir. Sistem əməliyyatları çox olduqda hər bir presedent üçün ayrıca kontroller yaradılır.

Kontroller şablonunun (*Controller*) ikişaf etdirilməsi nəticəsində Fasad (*Façade*) şablonu yaradılmışdır. Fasad şablonu bütün alt sistemlərə vahid giriş məntəqəsi – fasad obyekti yaratmağı təklif edir. Bu halda bütün xarici obyektlər yalnız fasad obyekti ilə iş görür.

12. Texniki tapşırıq

AVİNEMSİ yaradılması üçün texniki tapşırıq (TT) əsas sənəddir. Bu sənəd

avtomatlaşdırılmış sistem yaradılması qaydalarını və zəruri tələbləri təyin edir. Yəni AVİNEMSİ-nin işlənib hazırlanması və istismara verilməsi qaydaları TT-də əks etdirilir. TT hazırlanacaq məmulata qarşı qoyulan geniş tərkibli tələblərin sistemli təsvirini özündə saxlayır. AVİNEMSİ yaradılması üçün TT bütöv və ya təşkiledici sistem variantlarından biri kimi tərtib edilə biləndir. Bu o deməkdir ki, AVİNEMSİ bütöv sistem və ya alt sistem üçün yaradıla biləndir.

AVİNEMSİ yaradılması üçün TT-yə daxil edilən tələblər dünya təcrübəsində əldə edilmiş bütün ən son nailiyyətlərə müvafiq olmalıdır. Buna baxmayaraq, AVİNEMSİ yaradılması üçün TT layihələşdiricidən ötrü geniş yaradıcılıq imkanı da verən olmalıdır.

AVİNEMSİ yaradılması üçün TT-yə yalnız bu sistemin təşkiledici qismində işləyəcəyi avtomatlaşdırılmış idarəetmə sistemi (AİS) üçün nəzərdə tutulan sistem tələblərini tamamlayan tələblər daxil edilməlidir. AİS layihəsi ümumsistem tələbləri ilə yanaşı, konkret obyektin xüsusiyyətlərini də nəzərə alan tələbləri özündə əks etdirir. Buna görə də burada: sistemin strukturu və fəaliyyətinə, sistemin işçi heyətinin sayı və peşəkarlığı və iş rejiminə, təyinat göstəricilərinə, etibarlılığa, təhlükəsizliyə, erqonomikaya və texniki estetikaya, istismara, icazəsiz müdaxilədən informasiyanın mühafizəsinə, qəza baş verdikdə informasiyanın qorunmasına, kənar təsirlərlərdən qorunmalara, patent təmizliyinə, standartlaşdırma və unifikasiyaya, informasiya, riyazi, linqvistik, proqram, texniki, metroloji, təşkilati, metodiki və s. təminatlara qarşı tələblər təhlil edilməlidir. Texniki tapşırıqda aşağıdakılar nəzərdə tutulur:

1. Ümumi məlumat
2. Sistemin yaradılmasının (inkişaf etdirilməsinin) təyinatı və məqsədləri
3. Avtomatlaşdırma obyektlərinin xarakteristikası
4. Sistemə qarşı tələblər
5. Sistemin yaradılması üzrə işlərin tərkibi və məzmunu
6. Sistemin yoxlanması və qəbulu qaydası
7. Avtomatlaşdırma obyektinin fəaliyyətə buraxılmasına hazırlıq üzrə işlərin tərkibi və məzmununa qarşı tələblər
8. Sənədləşdirməyə qarşı tələblər

9. Layihələrin mənbələri

13. Eskiz[87] layihə

Təfərrüatlı[88] layihələşdirmənin məqsədi heç bir əlavə informasiyaya ehtiyac olmadan sistemin yaradılmasına imkan verən layihə sənədləri hazırlanmasından ibarətdir. Təfərrüatlı layihələşdirmə prosesinə, adətən, daha çox risk tələb edən aspektlərdən başlamaq ağlabatan hesab edilir. Birinci addım hər bir alt sistem üçün müvafiq arxitektura həlli (layihəsi) hazırlamağı nəzərdə tutur.

Sistemin arxitekturasının təfərrüatlı layihələşdirilməsi mərhələsində aşağıdakılar həyata keçirilir:

-aşağıdakı qlobal paketlər daxil edilir:

☺ təməl (*foundation*) siniflər (siyahılar, növbələr və s.);

☺ səhv emal ediciləri (*error handling classes*);

☺ riyazi kitabxanalar (bibliotekalar);

☺ utilitlər;

☺ digər malgöndərənlərin kitabxanaları (bibliotekaları).

-aşağıdakı layihə sinifləri (*design classes*) dəqiqləşdirilir:

☺ təhlil sinfi sadədirsə və ya yeganə məntiqi abstraksiyanı ifadə edirsə, layihə sinfində əks etdirilir;

☺ mürəkkəb təhlil sinfi bir-neçə sinfə, paketə və ya alt sistemə bölünə bilər.

İkinci mərhələdə arxitekturanı predmet oblastının sinifləri ilə bağlayan siniflərin təsviri dəqiqləşdirilir. Siniflər arasındakı əlaqə tipləri dəqiqləşdirilir.

Siniflərin təfərrüatlı layihələşdirilməsi aşağıdakı addımlarla həyata keçirilir:

1. sinif atributlarının dəqiqləşdirilməsi:

☺ atributun adından əlavə, susma rejimində tipi və qiyməti də verilir;

☺ atributların adlandırılması üzrə layihədə və reallaşdırma dilində qəbul edilmiş razılaşma nəzərə alınır;

---

[87] Эскиз м eskiz (bir şəklin, əsərin və s.-nin ilk layihəsi, qaralaması; model, nümunə, taslaq).

[88] Hərtərəfli, bütün xırdalıqları ilə

☺ *public, private və ya protected* kimi görünənlik atributları verilir;

☺ zəruri olduqda törəmə (hesablanan) atributlar təyin edilir.

2. Sinif əməliyyatlarının dəqiqləşdirilməsi:

☺ əməliyyatın[89] qısa təsviri yaradılır;

☺ *public, private və ya protected* kimi görünənlik əməliyyatları təyin edilir;

☺ əməliyyatın təsir oblastı[90] təyin edilir;

☺ funksiyanın ilkin şərtləri[91] daxil edilir. İlkin şərtlər funksiyanın icra edilməsi məqamına mövcudluğu nəzərdə tutulan dəyişənlərlə sabitlər arasındakı münasibətləri təsvir edir.

☺ funksiyanın icra sonrası şərtləri[92] verilir;

☺ fəaliyyət diaqramlarından, blok-sxemdən, psevdokoddan, əməliyyatın icrası zamanı müxtəlif obyektlərin qarşılıqlı təsirii diaqramlarından istifadə etməklə əməliyyatın icra alqoritminin təsviri tərtib edilir. Layihələşdirmə üçün zəruri olan hallarda əlavə əməliyyatlar daxil edilir.

3. atribut və metodların obyekt modelində təqdim edilməsi.

4. Sinif invariantlarının təyin edilməsi.

3-cü addımda yaradılmış layihənin tamlığı və obyekt modelinin istifadə variantları ilə təmin edilməsi yoxlanır.

Sinif obyektlərinə qarşı səmərəli tələbləri ifadə edən sinif invarintlarının təsviri sinif atributları terminlərində ifadə edilir. Sinif invariantları qiymətlərə məhdudiyyətlər forması alır. *C*-tələblər kimi, bunlar da tez-tez biznes qaydaları adlandırılır. Məsələn, “Bütün veb-auksion iştirakçıları öz kredit kartlarının nömrəsini təqdim etməlidir” tələbi iştirakçı sinfinin xüsusi invariantında aşağıdakı kimi təsvir edilməlidir:

зарегистрирован[93] == *true AND* 400000001 <= номерКредитнойКарты[94] <= 699999999 *OR* зарегистрирован == *false AND* номерКредитнойКарты == 0

---

[89] bütün parametrlərinin mənalandırılması da daxil olmaqla

[90] nüsxəsi (obyekt əməliyyatı) və ya təsnifləşdirici (sinif əməliyyatı)

[91] icra ediləcəyi şərait

[92] icra nəticəsində nələrin baş verəcəyi

[93] qeydiyyatdan keçmişdir

[94] Kredit Kartının nömrəsi

İNSİ-nin lazımi səfiyyədə fəaliyyət göstərməsi üçün reallaşdırılması zəruri olan bütün funksiyalar təyin edildikdən sonra həmin funksiyaların icrası alqoritmləri blok-sxemdən və ya psevdokoddan istifadə edilməklə təsvr edilməlidir.

Eskiz layihənin hazırlanması zamanı istifadəçi interfeysinin prototipinin yaradılması mühüm əhəmiyyət daşıyır. İstifadəçi interfeysinin ilkin cizgiləri *C*-tələblərin formalaşdırılması mərhələsində istifadəçi ilə birgə hazırlanmış olur.

14. Proqram interfeyslərinin təşkili

Vahid İNSİ çərçivəsində müxtəlif proseslərin eyni vaxtda gedişinin təşkili günbəgün aktuallaşmaqdadır. Bir-neçə prosesin eyni vaxtda vahid düyündə icrasına paralel hesablamaların təşkili kimi baxılır. Eyni vaxtda müxtəlif şəbəkə düyünlərində gedən proseslər zamanı məqsədə çatmaq üçün düyünlərarası verilən mübadiləsi təşkil etmək lazım gəlir. Bu halda baş verən prosesə paylanmış hesablamalar kimi baxılır. Paylanmış hesablamalara (verilən emalına) əyani misal olaraq korporativ proqram əlavəsində müştəri ilə server arasında qarşılıqlı fəaliyyətin təşkili göstərilə bilər.

Paylanmış hesablamalarda ən vacib problem xeyli aralı şəbəkə düyünləri arasında rabitə kanalları ilə ötürmələrin minimumlaşdırılmasından ibarətdir. Bu problemi həll etmək üçün məsafədən müraciət interfeysindən[95] istifadə edilir.

Müştərilərlə qarşılıqlı fəaliyyət zamanı seanslar arasında obyektlərin vəziyyətini yadda saxlamaq tələb olunduqda aşağıdakı tipik həllərdən birindən istifadə etmək lazım gəlir: 1) seansın vəziyyəti müştəri tərəfdə saxlanır; 2) seansın vəziyyəti VB-də saxlanır; 3) seansın vəziyyəti server tərəfdə saxlanır.

Bəzi hallarda bir məsafədən müraciət interfeysi tərkibi mürəkkəb olan bir-neçə obyekt üçün şlüz rolunda çıxış edə bilir. Məsələn, Sifariş (*Заказ*) obyekti üçün məsafədən müraciət interfeysi sifariş haqqında məlumat toplamaq və ya həmin sifarişi yerləşdirən alıcı barədə bəzi məlumatları əldə etmək üçün tətbiq edilə bilər. Bu zaman müştəri üçün predmet oblastındakı obyektlərə aid serverdən məlumatlar lazım ola bilər.

[95] *remote facade*

Bu iş üçün verilən ötürən obyekt[96] tətbiq edilir. Bu halda müştəriyə həm sorğuya cavab, həm də gələcəkdə lazım ola biləcək əlavə məlumatlar ötürülür.

15. VB-nin layihələşdirilməsi üçün UML notasiyaları

Verilənlər bazasının (VB-nin) layihələşdirilməsi verilənlərin saxlanması üçün istifadə edilən VBİS-in[97] tipindən, yəni relyasiyalı və ya obyektli olmasından asılıdır. Obyektli VB üçün heç bir layihələşddirmə tələb olunmur. Çünki mahiyyət-siniflər bilavasitə VB-də əks etdirilir. Halbuki relyasiyalı VB-lər üçün obyekt modelinin mahiyyət-sinifləri relyasiyalı VB-nin cədvəllərində əks etdirilməlidir. Cədvəllər çoxluğu və onların arasındakı əlaqələr *ER*-diaqramları adlanan sinif diaqramları şəklində təsvir edilir. Siniflərin VB cədvəllərində əks etdirilməsi üçün tətbiq edilən qaydalar yığımı faktiki olaraq mahiyyətlərin və əlaqələrin dəyişdirilməsi qaydaları ilə üst-üstə düşür.

VB sxeminin təsviri üçün *UML* dilinin[98] aşağıdakı elementləri yığımı tətbiq edilir:

- cədvəl «*Table*» stereotipi ilə sinif şəklində təqdim edilir;
- təqdimat «*View*» stereotipi ilə sinif şəklində təqdim edilir;
- cədvəlin sütunu verilən tipinə müvafiq gələn sinif atributu şəklində təqdim edilir;
- adi assosiasiya və aqreqasiya[99] «*Non-Identifying*»[100] stereotipli assosiasiya şəklində təqdim edilir;
- kompozisiya «*Identifying*»[101] stereotipli assosiasiya şəklində təqdim edilir;
- VB sxemi «*Schema*» stereotipi ilə sinif-cədvəlləri saxlayan paket şəklində təqdim edilir;
- saxlanmış prosedurlar konteyneri «*SP Container*» stereotipi ilə sinif şəklində təqdim edilir;
- bütövlük məhdudiyyətləri, indekslər və triggerlər «*PK*» (*Primary Key*), «*FK*» (*Foreign Key*), «*Unique*», «*Check*», «*Index*» və «*Trigger*» stereotipləri ilə sinif-cədvəl

[96] *data transfer object*
[97] Verilənlər bazasının idarəetmə sisteminin
[98] öz stereotipləri (*UML* profili) ilə birlikdə
[99] iriləşdirmə, yuvarlaqlaşdırma
[100] *IDEF1X* terminologiyalarında – identifikasiya etməyən əlaqə
[101] *IDEF1X* terminologiyalarında – identifikasiya edici əlaqə

əməliyyatları şəklində təqdim edilir;

- fiziki VB «*Database*» stereotipi ilə komponent şəklində təqdim edilir.

VB-nin layihələşdirilməsi zamanı strukturun yaradılmasından əlavə, bir qayda olaraq, sistemin tam gücü ilə fəaliyyət göstərməsi üçün nəzərdə tutulması zəruri olan bütün sorğular sadalanmalı, həmçinin *SQL*-sorğuların tiplərinin, saxlanmış prosedurların, təqdimatların və triggerlərin sintaksisi və təsviri verilməlidir ki, VB-nin vəziyyətinin monitorinqini keçirmək, sistemin fəaliyyətinin etibarlılığını yüksəltmək, genişlənəbilənliyini təmin etmək və müştəri tərəfdə işləməyi asanlaşdırmaq mümkün olsun.

Proqram təminatının sənədləşdirilməsi[102] üçün *IEEE* 1016-1987 standartında sistemin layihə sənədlərinin tərtibi və rəsmiləşdirilməsi üzrə təlimatlar vardır.

16. İstifadəçi interfeysinin layihələşdirilməsinin praktiki aspektləri

16.1. İstifadəçi interfeysinin layihələşdirilməsi məsələləri

İstənilən interfeys üçün 4 əsas keyfiyyət kriterisi mövcuddur: istifadəçilərin işləmə sürəti, insan səhvlərinin sayı, öyrənmə sürəti və istifadəçinin subyektiv məmnunluğu[103]. Bu kriterilərə cavab verən interfeys səmərəli sayılır. Səmərəli interfeys qurulması prinsipləri insanın informasiya qəbuletmə üzrə psixi-fizioloji xüsusiyyətləri ilə birbaşa əlaqədar olmalıdır. İstifadçi interfeysi hazırlamaq üçün ən geniş istifadə edilən standartlar toplusu 1987-ci ildə *IBM*-in nəşr etdirdiyi *Common User Access (CUA)* sayılır. Həmin vaxtdan bəri istifadəçi interfeysinin dizaynı üçün çoxsaylı standartlar meydana çıxmışdır.

Bir çox interfeys elementi üçün aid olan əsas təməl prinsiplər aaşağıdakılardan ibarətdir:

1. Pol Fits tərəfindən 1954-cü ildə formalaşdırılan: “Məqsədəçatma müddəti məqsədə qədərki məsafə ilə düz, məqsədin ölçüsü ilə tərs mütənasib asılılıqdadır” qaydasıdır;

---

[102] *SDD – Software Design Document*

[103] yəni interfeysin istifadəçi məsələsinə münasibliyi (uyğunluğu)

2. “Relevant olan - irəli” prinsipi: yəni tez-tez istifadə edilən elementlər ekranın sol yuxarı hissəsində, nadir hallarda istifadə edilənlər isə ekranın aşağı sağ hissəsində yerləşdirilməlidir.

3. Menyü elementlərini qruplaşdırarkən, siyahılar tərtib edərkən və s. qısamüddətli yaddaşa qoyulan məhdudiyyət[104] nəzərə alınmalıdır. Ən yaxşısı 6 (7-1) elementdir.

İstifadəçinin fizioloji xüsusiyyətləri ilə bağlı olaraq interfeysin təşkili üçün ümumi təkliflər aşağıdakılardan ibarətdir:

1. İstifadəçilər heç vaxt kömək üçün verilən arayışı oxumurlar, buna görə də imkan daxilində hər bir ekran yadasalmalarla[105] təchiz edilməlidir.

2. İstifadəçilər bir-neçə ardıcıl açılmış ekranlarda olan informasiyanı yadda pis saxlayırlar və bunu yadda saxlamaq onları işdən yayındırır. Çünki bundan sonra istifadəçi işi davam etdirmək üçün bilməlidir ki:

a. hansı addımda dayanmışdı

b. sistemə hansı əmrləri və parametrləri daxil etmişdi

c. növbəti addımda o nə etməli idi

d. diqqəti yayınan momentdə nəyə baxırdı, yəni son aktiv element nə idi.

3. İstifadəçi səhvləri ilə mübarizə apararkən əsas diqqət:

a. iş prosesində istifadəçilərin axıcı öyrədilməsinə

b. sayıq olmağa tələbin azaldılmasına

c. məsuliyyətli (qayıdışsız) əməliyyatların icrası zamanı diqqətiçəkənliyə[106] və aydınlğın artırılmasına

d.sistemin səhvlərə həssaslığının azaldılmasına[107] yönəldilməlidir.

---

[104] 7±2 bənd yadda saxlamaq mümkündür

[105] подсказка ж dan. 1. gizlicə söyləmə, yerindən söyləmə, xəlvətcə söyləmə, yavaşcadan demə; 2. məc. fikrinə salma, təlqin etmə, öyrətmə, ağzına söz vermə.

[106] İnterfeysdə ən vaciblər daha çox diqqətçəkəkn olmalıdır.

[107] Bunun üçün 3 əsas üsul vardır: 1) əmənin düzgünlüyünə təsdiq alınanadək potensial təhlükəli əmə-lərin bloklaşdırılması; 2) istifadəçinin bütün əmələri qabaqcadan sistem tərəfindən yoxlanmalıdır; 3) zəruri əmr və parametrlər sistem tərəfindən sərbəst surətdə seçilir, istifadəçiyə seçimin düzgünlüyünü yoxlamaq qalır. İstifadəçi interfeysinin inkişafının müasir istiqamətlərindən biri adaptiv interfeys yara-dılmasıdır ki, bu zaman interfeys istifadəçi tərəfindən edilmiş son sazlamaları yadda saxlayır. İnsan səhvlərinin mövcudluğu interfeysin dizaynının yaxşı olmadığını göstərir. 4) funksionallığı kontekstdən

e. Səhvlər barədə məlumatı diqqətlə düşünmək lazımdır. Çünki: 1) səhv haqqında ideal məlumat aşağıdakı 3 suala cavab verməlidir: Problem nədən ibaarətdir? Bu problemi indi necə həll etməli? Necə etməli ki, bu problem təkrarlanmasın? 2) səhv haqqında məlumatın mətni texniki yazara göstərilməlidir. 3) səhv haqqında məlumat mümkün qədər qısa olmalıdır. 4) Proqram təminatında səhv haqqında məlumatın dialoq pəncərəsinə çıxarılması yaxşı üsul deyil. Qovuq[108] (*balloon*), dialoq pəncərəsi ilə müqayisədə bir sıra üstünlüklərə malikdir. Belə ki: bu, pəncərəyə nisbətən diqqəti daha az çəkir; qovuğu bağlamaq üçün istifadəçi hər hansı düyməni sıxmağa məcbur deyil, ekranın istənilən yerində mışı tıklamaqla qovuğu bağlaya bilər; qovuq sistemin əhəmiyyətli hissəsini örtmür; ən əsası, hansı idarəetmə elementində səhv baş verdiyini göstərir.

4. İnterfeys hazırlayarkən:

a. Parlaq rənglərdən qaçmaq lazımdır. Gözü yormayan azsaylı parlaq və yumşaq rənglər vardır.

b. Visualda (görünəndə) iti bucaqlardan qaçılmalıdır.

c. Vizualı (görünəni) yüngül və şəffaf düzəltmək lazımdır.

d. elementlərin sıx yerləşdiyi yerləri kontrast rəngləmək lazım deyil.

e. İnterfeys maksimal dərəcədə vizual qanunlara uyğun yaradılmalıdır. Qanunauyğunluq çox olduqca, harmoniya da çox olur. Ən əhəmiyyətsiz qanunauyğunluq da diqqətəlayiqdir. Qanunauyğunluq dedikdə, burada müxtəlif obyektlərin xassələrinin uyarlığı nəzərdə tutulur. Məsələn, düymənin hündürlüyü dialoq pəncərəsi içliyinin eninin 2 misli qədər olmalıdır.

f. modul şəbəkəsindən istifadə etmək məqsədəuyğundur.

g. bütün ölçülər və koordinatlar qızıl kəsişməyə (0.618 x 0.382) uyğun olmalıdır.

---

asılı olan idarəedici element yaratmaq məsləhət deyil. 5) istifadəçiyə özü tərəfindən icra edilmiş hüdudsuz sayda əmælləri təxirə salmaq imkanı verilməlidir.

[108] пузырь м1. qovuq; 2. qabar, qabarıq, suluq; 3. dan. yumru (koppuş, kök) uşaq;4. anat. kisə, kisəcik; жёлчный пузырь öd kisəsi.

16.2. İstifadəçi interfeysinin idarə edilməsi

İdarəetmə elementlərinin səmərəliliyi kriterisi kimi həmin elementlərin təyinatının dərk edilənliyinin yaxşılaşdırılması, onlara müraciət sürətinin artırılması və eyni zamanda mışın kursorunun yönəldilməsi dəqiqliyinə tələbin minimumlaşdırılması seçilə bilər.

Düymə - birbaşa əməl elementidir. Düymə istifadəçini, internetdə olduğu kimi, kontentin digər fraqmentinə aparırsa, mətni iqtibas kimi, əməli icraya buraxırsa düymə kimi işləyir. Düymə maksimum iri olmalıdır ki, istifadəçi ona lazım olan əməli icrya buraxan düyməni rahat işlədə bilsin. Əks halda, eyni zamanda bir-neçə düymə sıxıla bilər. Əmr düymələrinə infinitiv formalı feillərdən[109] ad verilməlidir. Dialoq pəncərəsinin aşağısında 3 düymə var: Ок, Применить и Отмена (триединство ошибки).

Mətnlə yanaşı, düymə üzərində piktoqram da yerləşdirmək olar. Lakin interfeysin münasibliyi və vəhdətliliyi baxımından çalışmaq lazımdır ki, düymə üzərində piktoqram olmasın. Çünki bunun xərci çox, səmərəsi azdır. Çekbokslar və radiodüymələr təxirə salınmış əməl düymələridir. Bunların sıxılması heç bir təcili əməli təşəbbüsləndirmir. Bu düymələrin köməyi ilə istifadəçi daxil etdiyi parametrlər digər idarəetmə düymələrinin işində istifadə edilir. Çekbokslar və radiodüymələr şaquli qaydada yerləşdirilməlidir. Bu, lazımi elementin axtarışını əhəmiyyətli dərəcədə sürətləndirir. Göstərişlər (подписи) inkaredici deyil, istiqamətverici olmalıdır (например, «Показывать пробелы» и «Показывать табуляции»).

Siyahılar. Siyahının eni elə olmalıdır ki, elementlər arasındakı fərqi təyin emək mümkün olsun. Element siyahıdan enli olduqda fırlatma düymələri daxil etmək lazımdır. İşin sürətləndirilməsi üçün elementlər çeşidlənməlidir. Piktoqramlar daha cəlbedicidir. Açılan siyahılar daha çox naviqasiya yönümlüdür.

Daxiletmə sahələri. Daxiletmə sahəsinin eni daxil ediləcək ən iri mətnin yerləşməsinə imkan verən olsa da, həddən artıq böyük sahə də məqsədəuyğun deyil. Tez-tez istifadə edilən göstərişlər (подписи) daxiletmə sahələrinin üst tərəfində olsa

[109] Прийти, Увидеть, Победить

yaxşıdır.

Fırladıcılar (*spinner, little arrow*) da daxiletmə sahələri sayılır. Lakin bunlar universal sahə deyildir.

Sürüngəclər (Ползунки) qiymət seçmək üçün daha çox istifadə edilir.

Menyü. Menyü sistemin istifadəçiyə təklif etdiyi variantlar toplusudur. Bu baxımdan, daxiletmə sahəsi olmayan çoxdüyməli dialoq pəncərəsi də menyüdür. Menyünün səmərəliliyinə ayrı-ayrı elementlərin quruluşu güclü təsir edir. Menyüdə piktoqram olması səmərəlidir.

Pəncərənin strukturu. Bu, interfeysin keyfiyyətinə güclü təsir edən elementdir. Pəncərənin strukturunun layihələşdirilməsi zamanı aşağıdakılar nəzərə alınmalıdır:

1. pəncərədəkilər asan oxunmalı, tez diqqət cəlb edən olmalıdır.

2. pəncərə mətn kimi oxuna bilən olmalıdır.

3. pəncərə "relevant – irəli" prinsipi ilə qurulmalıdır. Yəni tez-tez istifadə edilən elementlər sol yuxarı küncdə, nadir hallarda istifadə edilənlər isə aşağı sağ küncdə yerləşdirilməlidir.

16.3. İstifadəçi interfeysinin layihələşdirilməsi

Faktiki olaraq dizayn prosesinin müvəffəqiyyətlə və qeyd-şərtsiz davam etdirilməsi üçün: layihələşdirmə, prototipin hazırlanması, testləşdirmə, modifikasiya, testləşdirmə, modifikasiya və s. ardıcıllığı ilə hərəkət etmək lazımdır. İnterfeysin layihələşdirilməsi tələblərin formalaşdırılması zamanı tərtib edilən eskiz (qaralama) layihə hazırlanmasından başlanır. Eskiz layihənin hazırlanması məqamına istifadəçi ilə qarşılıqlı fəaliyyət üçün lazım olan 100% funksionallığı təmin edən ekranların sayı məlum olmalıdır. Beləliklə, interfeysin layihələşdirilməsi, əslində, ayrı-ayrı ekranların ardıcıl layihələşdirilməsindən ibarətdir. Bu zaman hər bir ekran bir-neçə variantda hazırlanmalıdır ki, ən səmərəli variantı seçmək mümkün olsun. Prototiplərin hazırlanması da çoxvariantlı ola bilər. Lakin bu, olduqca uzun və əməktutumlu prosesdir. Bu nöqsanı aradan qaldırmaq üçün 1983-cü ildə Kard, Moran və Hyüvel yaxşı variantın seçilməsindən ötrü

GOMS[110] adlı metod təklif etmişlər. İdeya istifadəçinin hərəkətini tərkib hissələrinə[111] parçalamaqdan ibarətdir.

GOMS aşağıdakıları məqbul sayır:

• *Alt, Ctrl* və *Shift* (K) də daxil olmaqla, ixtiyari klaviatura düyməsinin sıxılması üçün: 0,28 saniyə

• Mışın düyməsinin sıxılması üçün (M): 0,1 saniyə

• Mışın kursorunun yerdəyişməsi üçün (П): 1,1 saniyə

• Mışın götürülməsi və ya mışdan imtina edilməsi (B): 0,4 saniyə

• Əməlin seçilməsi müddəti üçün (Д): 1,2 saniyə

Prototipi hazırlayarkən ən geniş yayılmış səhv onu bitkin sistem kimi təsəvvür etməkdir. Bunu aradan qaldırmaq üçün bütün pəncərələrin kağızda şəklinin çəkilib ilkin baxışdan keçirilməsidir. Münasib variant seçildikdən sonra həmin kağız variant bu və ya digər təqdimat proqramında, məsələn, *MS PowerPoint* qrafik redaktorunda interfeysin elektron şəkli hazırlanmalıdır. Bu zaman hər bir ekran ayrıca slaydda yerləşdirilməlidir.

---

[110] *GOMS* (*Goals, Operators, Methods, and Selection Rules* – цели, операторы, методы и правила их выбора)

[111] Məsələn, mışı götürmək və ya kursoru hərəkət etdirmək

Ədəbiyyat siyahısı

1. «Advanced Software Engineering: Expanding the Frontiers of Software Technology», IFIP 19th World Computer Congress. (2006 ; Santiago, Chile). First International Workshop on Advanced Software Engineering, 26-28 April 2006 / Edited by S.F. Ochoa. – New York : Springer, 2006. – 178 p.
2. «Advances in Databases and Information Systems», ADBIS 2008. (2008 ; Pori, Finland). 12th East European Conference, 5-9 September 2008 / Edited by Hannu Jaakkola. – Berlin : Springer,2008.– 331p.
3. «Agile Processes in Software Engineering and Extreme Programming», 8th International Conference, XP 2007 (2007 ; Como, Italy). 18–22 June 2007 / Edited by Giancarlo Succi. – Germany : Springer-Verlag Berlin Heidelberg, 2007. – 289 p.
4. «Database and XML Technologies», XSym 2003. (2003 ; Berlin, Germany). First International XML Database Symposium, 8 September 2003 / Edited by Rainer Unland. – Berlin : Springer, 2003. – 293 p.
5. «Empirical Software Engineering Issues Critical Assessment and Future Directions», International Workshop (2006; Dagstuhl Castle, Germany), 26-30 June 2006 / Edited by R. Selby. – Berlin : Springer, 2007. – 209 p.
6. «Fundamental Approaches to Software Engineering», ETAPS 2004 (2004 ; Barcelona, Spain). 7th International Conference, Held as Part of the Joint European Conferences on Theory and Practice of Software, March 29 –April 2 2004 / Edited by Tiziana Margaria-Steffen. – Berlin : Springer, 2005. –404 p.
7. «Information Systems Development», Thirteenth International Conference on Information Systems Development Advances in Theory, Practice and Education (2004; Vilnius, Lithuania). 9–11 September 2004 / Edited by Olegas Vasilecas. – New York : Springer, 2005. – 550 p.
8. «Programming Languages and Systems», APLAS 2008 (2008 ; Bangalore, India). 6th Asian Symposium, 9-11 December 2008 / Edited by G. Ramalingam. – Berlin : Springer, 2008. – 350 p.

9. «Web Information Systems Engineering», International Workshops (September 1–4, 2008; Auckland, New Zealand). 1–4 September 2008 / Edited by Markus Kirchberg. – Berlin : Springer, 2008. – 199 p.

10. Abmann U. Invasive Software Composition. / U. Abmann. – Germany: Springer, 2003. – 334 p.

11. Andersson E. Software Engineering for Internet Applications / E. Andersson, P. Greenspun, A. Grumet.– London: MIT Press, 2006. – 411 p.

12. Arnowitz J. Effective Prototyping for Software Makers / J. Arnowitz, M. Arent, N. Berger. – Canada: Elsevier, 2007. – 625 p.

13. Aurum A.·Engineering and Managing Software Requirements / A. Aurum; edited by C. Wohlin. –Berlin: Springer, 2005. – 487 p.

14. Bamford R. ISO 9001:2000 for Software and Systems Providers / R. Bamford, W.J. Deibeler. – London: CRC Press, 2004. – 309 p.

15. Bass L. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman. – 2nd edition. – London: Addison Wesley, 2003. – 560 p.

16. Bean J. SOA and Web Services Interface Design. Principles, Techniques and Standards / J. Bean. –Burlington: Morgan Kaufmann, 2010. – 360 p.

17. Bernus P. Handbook on Architectures of Information Systems / P. Bernus, K. Mertins, G. Schmidt. –Berlin : Springer, – 2006. – 869 p.

18. Boucher T. Design of Industrial Information Systems / T. Boucher, A. Yaçin – Canada : Elsevier, 2006.– 431 p.

19. Brandon D.M. Software Engineering for Modern Web Applications: Methodologies and Technologies /D.M. Brandon – Hershey: IGI Global, 2008. – 403 p.

20. Buschmann F. Pattern-Oriented Software Architecture. A System of Patterns, Volume 1 / F. Buschmann, R. Meunier et al. Chichester : John Willey & Sons, 2001. – 487 p.

21. Buschmann F. Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing / F. Buschmann, K. Henney, D.C. Schmidt. - Glasgow: John Wiley & Sons, Ltd., 2007. – 639 p.

22. Buschmann F. Pattern-Oriented Software Architecture: On Patterns and Pattern Languages / F.Buschmann, K. Henney, D.C. Schmidt. - Glasgow: John Wiley & Sons, Ltd., 2007. – 491 p.

23. Cassidy A. A Practical Guide to Information Systems Process Improvement / A. Cassidy, K.Guggenberger – Florida: CRC Press LLC, 2001. – 286 p.

24. Cecelja F. Manufacturing Information and Data Systems / F. Cecelja – London: Penton Press, 2002. –201 p.

25. Clements P. Documenting Software Architecture / P. Clements, F. Bachmann et al. – London: Addison Wesley, 2001. – 342 p.

26. Cockburn A. Agile Software Development: The Cooperative Game / A. Cockburn - London: Addison Wesley, 2006. – 504 p.

27. Construx. Software Development Best Practices [Электронный ресурс] / Официальный сайт Construx; Steve McConnell – Электрон. дан. – Bellevue: Construx, 2009 – . – Режим доступа: http://www.construx.com, свободный. – Загл. с экрана. – Яз. англ.

28. Ebert C. Best Practices in Software Measurement. How to Use Metrics to Improve Project and Process Performance / C. Ebert, R. Dumke, M. Bundschuh, A. Schmietendorf. – Berlin: Springer-Verlag, 2005.– 299 p.

29. Ellis H.J.C. Software Engineering: Effective Teaching and Learning Approaches and Practices / H.J.C. Ellis, S.A. Demurjian, J.F. Naveda. - New York: Information science reference, 2009. – 433 p.

30. Erl T. SOA Design Patterns / T. Erl. – Boston: Prentice Hall, 2009. – 856 p.

31. Georgakopoulos D. Service-Oriented Computing / D. Georgakopoulos, M.P. Papazoglou. - Cambridge: The MIT Press, 2009. – 387 p.

32. Giorgini P. Agent-Oriented Methodologies / P. Giorgini. – Hershey: Idea Group Inc., 2005. – 429 p.

33. Goodman F.A. Defining and Deploying Software Processes / F.A. Goodman. – New York : Taylor & Francis Group, 2006. –246 p.

34. Gorton I. Essential Software Architecture / I. Gorton. – Berlin: Springer-Verlag,

2006. – 292 p.

35. Greenfield J. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools / J. Greenfield, K. Short et al. - New York: John Wiley & Sons, 2004. – 666 p.

36. Handbook of Enterprise Integration / edited by M.H. Sherif – New York : CRC Press, 2010. – 730 p.

37. Hohmann L. Beyond Software Architecture: Creating and Sustaining Winning Solutions / L. Hohmann.– London: Addison Wesley, 2003. – 352 p.

38. Horstmann C. Object-Oriented Design & Patterns / C. Horstmann. - Hoboken: John Wiley & Sons, Inc., 2006. – 473 p.

39. International Function Point Users Group. [Электронный ресурс] / Официальный сайт International Function Point Users Group; President Tom Cagley – Электрон. дан. – Princeton : IFPUG, 2009 – . –Режим доступа : http://www.ifpug.org, свободный. – Загл. с экрана. – Яз. англ.

40. Jalote P. An Integrated Approach to Software Engineering / P. Jalote. – New York : Springer, 2005. –571 p.

41. Kaisler S.H. Software Paradigms / S.H. Kaisler. – New Jersey : John Wiley & Sons, Inc., 2005. – 458 p.

42. Khosrow-Pour M. Cases on Information Technology: Planning, Design and Implementation / M.Khosrow-Pour. – London : Idea Publishing, – 2006. – 385 p.

43. Kircher M. Pattern-Oriented Software Architecture: Patterns for Resource Management, Volume 3 / M. Kircher, J. Prashant. - New Jersey : John Wiley & Sons, Inc., 2004. – 310 p.

44. Kirn S. Multiagent Engineering Theory and Applications in Enterprises / S. Kirn, O. Herzog, P.Lockemann ; edited by O. Spaniol. – Berlin: Springer, 2006. – 617 p.

45. Krishnamurthy N. Building Software. A Practitioner’s Guide / N. Krishnamurthy, A. Saran. – New York: Taylor & Francis Group, 2008. – 380 p.

46. Krogstie J. Conceptual Modeling in Information Systems Engineering / J. Krogstie, A.L. Opdahl ; edited by S. Brinkkemper. – Berlin: Springer, 2007. – 356 p.

47. Laird L.M. Software Measurement and Estimation. A Practical Approach / L.M. Laird, M.C. Brennan. – New Jersey: IEEE Computer Society, 2006. – 276 p.

48. Langer A.M. Analysis and Design of Information Systems / A.M. Langer. – London: Springer, 2008. – 437 p.

49. Lattanze A.J. Architecting Software Intensive Systems. A Practitioner's Guide / A.J. Lattanze. – London: Taylor&Francis Group, 2009. – 464 p.

50. Lauesen S. User Interface Design. A Software Engineering Perspective / S. Lauesen. – Harlow: Addison Wesley, 2005. – 623 p.

51. Lemos R. Architecting Dependable Systems / R. Lemos, C. Gacek, A. Romanovsky (Eds.). – New York: Springer-Verlag, 2003. - 312 p.

52. Meszaros G. xUnit Test Patterns. Refactoring Test Code / G. Meszaros. – Boston: Pearson Education, 2007. – 948 p.

53. Nilsson J. Applying Domain-Driven Design and Patterns: With Examples in C# and .NET / J. Nilsson. - Harlow : Addison Wesley Professional, 2006. – 576 p.

54. Olivé A. Conceptual Modeling of Information Systems / A. Olivé. – Berlin : Springer, 2007. – 471 p.

. Övergaard G. Use Cases Patterns and Blueprints / G. Övergaard, K. Palmkvist. – Harlow: Addison Wesley Professional, 2004. – 464 p.

56. Pressman R.S. Software Engineering. A Practitioners Approach / R.S. Pressman. – New York: McGraw-Hill, 2001. – 888 p.

57. Pugh K. Interface-Oriented Design / K. Pugh. – Texas: The Pragmatic Programmers LLC, 2006. – 220p.

58. Rosenberg D. Use Case Driven Object Modeling with UML. Theory and Practice / D. Rosenberg, M.Stephens. - New York: Springer-Verlag, 2007. – 471 p.

59. Salzman H. Software by Design. Shaping Technology and the Workplace / H. Salzman, S.R. Rosenthal.– New York : Oxford, 1994. – 361 p.

60. Sametinger J. Software Engineering with Reusable Components / J. Sametinger. – Berlin: Springer-Verlag, 1997. – 285 p.

61. Schmidt M. Implementing the IEEE Software Engineering Standards / M.

Schmidt. – Indianapolis: SAMS, 2000. – 255 p.

62. Shalloway A. Design Patterns Explained A New Perspective on Object-Oriented Design / A. Shalloway, J.R. Trott. New York : Addison Wesley Professional, 2004. – 480 p.

63. Sherif M.H. Handbook of Enterprise Integration / M.H. Sherif. – London: CRC Press, 2010. – 730 p.

64. Shoval P. Functional and Object-Oriented Analysis and Design: An Integrated Methodology / P. Shoval.– Hershey : Idea Group Publishing, 2007. – 357 p.

65. Spolsky J. User Interface Design for Programmers / J. Spolsky. –New York: Springer-Verlag, 2001. –101 p.

66. Szyperski C. Component Software. Beyond Object-Oriented Programming / C. Szyperski, D. Gruntz, S.Murer. – New York : Addison-Wesley, 2002. – 625 p.

67. Tang X. Web Content Delivery / X. Tang, J. Xu, S.T. Chanson. – New York : Springer, 2005. – 387 p.

68. Vasilecas O. Information Systems Development : Advances in Theory, Practice and Education / O.Vasilecas, A. Caplinskas et al. – New York : Springer, 2005. – 550 p.

69. Volter M. Remoting Patterns. Foundations of Enterprise, Internet and Realtime Distributed Object Middleware / M. Volter, M. Kircher, U. Zdun. – Chichester: John Wiley & Sons Ltd, 2005. – 424 p.

70. Wang L. Modern Industrial Automation Software Design / L. Wang, K. Tan. – New Jersey : John Wiley & Sons, Inc., 2006. – 349 p.

71. Wang Y. Software Engineering Foundations. A Software Science Perspective / Y. Wang. – New York: Auerbach Publications, 2008. – 1419 p.

72. Wodtke C. Information Architecture: Blueprints for the Web / C. Wodtke, A. Govella. – Berkeley : New Riders, 2009. – 313 p.

73. Wout J. The Integrated Architecture Framework Explained / J. Wout, M. Waage, et al. - New York, Springer, 2010. – 263 p.

74. Yacoub S.M. Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems /S.M. Yacoub, H.H. Ammar. - New York : Addison Wesley,

2003. – 416 р.

75. Амблер С. Гибкие технологии: экстремальное программирование и унифицированный процесс разработки. Библиотека программиста / С. Амблер. – СПб: Питер, 2005. – 412 с.

76. Арлоу Д. UML 2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование / Д. Арлоу, А. Нейштадт. – 2-е изд. – СПб: Символ-Плюс, 2007. – 624 с.

77. Бек К. Экстремальное программирование: разработка через тестирование / К. Бек. – СПб.: Питер,2003. – 224 с.

78. Белладжио Д. Стратегия управления конфигурацией программного обеспечения с использованием IBM Rational ClearCase / Д. Белладжио, Т. Миллиган. – М.: ДМК Пресс, 2007. –384 с.

79. Благодатских В.А. Стандартизация разработки программных средств / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов ; под ред. О.С. Разумова. – М.: Финансы и статистика, 2005. – 288 с.

80. Брауде Э. Технология разработки программного обеспечения / Э. Брауде. – СПб.: Питер, 2004. –655 с.

81. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений / Г. Буч, Р. Максимчук, М. Энгл и др. – 3-е изд. – М.: ООО «И.Д. Вильямс»,2008. – 720 с.

82. Вендров А.М. Проектирование программного обеспечения экономических информационных систем/А.М. Вендров.– М.: Финансы и статистика, 2006. – 544с.

83.Вендров А.М. Практикум по проектированию программного обеспечения экономических информационных систем : Учеб. пособие / А.М. Вендров. – М.: Финансы и статистика, 2006. –192 с.

84. Вигерс К. Разработка требований к программному обеспечению / К. Вигерс. – М. : Издательско-торговый дом «Русская редакция», 2004. – 576 с.

85. Влиссидес Д. Применение шаблонов проектирования. Дополнительные штрихи / Д. Влиссидес. –М.: Издательский дом «Вильямс», 2003. – 144 с.

86. Гагарина Л.Г. Технология разработки программного обеспечения: учебное пособие / Л.Г.Гагарина, Е.В. Кокорева, Б.Д. Виснадул ; под ред. Л.Г. Гагариной. – М.: ИД «Форум»: ИНФРА-М, 2008. – 400 с.

87. Гайдамакин Н.А. Автоматизированные информационные системы, базы и банки данных. Вводный курс : учебное пособие / Н.А. Гайдамакин. – М.: Гелиос АРВ, 2002. – 368 с.

88. Гамма Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Э.Гамма, Р. Хелм, Р. Джонсон. – СПб.: Питер, 2008. – 366 с.

89. ГОСТ 19.003-80. Схемы алгоритмов и программ. Обозначения условные графические. – Введ.1981–07–01. – М. : Изд-во стандартов, 1982. – 4 с.

90. ГОСТ 2.104-2006. Единая система конструкторской документации. Основные надписи. – Введ. 2006–02–28. – М. : Изд-во стандартов, 2006. – 18 с.

91. ГОСТ 2.105-95. Общие требования к текстовым документам. – Введ. 1996–07–01. – М. : Изд-во стандартов, 1998. – 28 с.

92. ГОСТ 24.103-84. Автоматизированные системы управления. Основные положения. – Введ. 1985–07–01. – М. : Изд-во стандартов, 1985. – 2 с.

93. ГОСТ 24.104-85. Автоматизированные системы управления. Общие требования. – Введ. 1987–01–01. – М. : Изд-во стандартов, 1987. – 7 с.

94. ГОСТ 24.602-86. Состав и содержание работ по стадиям создания. – Введ. 1988–01–01. – М. : Изд-во стандартов, 1988. – 6 с.

95. ГОСТ 34.003-90. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Термины и определения. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1992. – 14 с.

96. ГОСТ 34.601-90. Автоматизированные системы. Стадии создания. – Введ. 1992–01–01. – М. : Изд-во стандартов, 1992. – 3 с.

97. ГОСТ 34.602-89. Техническое задание на создание автоматизированной системы. – Введ. 1990–01–01. – М. : Изд-во стандартов, 1990. – 6 с.

98. ГОСТ Р 7.0.5-2008. Библиографическая ссылка. Общие требования и правила составления. –Введ. 2009–01–01. – М. : Изд-во стандартов, 2009. – 22 с.

99. ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств. – Введ. 1999–12–23. – М. : Изд-во стандартов, 2000. – 84 с.

100.Гранд М. Шаблоны проектирования в Java / М. Гранд. – М.: Новое знание, 2004. – 559 с.

101.Зельцер С.Р. Проектирование автоматизированных систем обработки информации и управления. Часть II. Общие вопросы проектирования: Учебное пособие / С.Р. Зельцер. НФИ КемГУ. – Новокузнецк, 2003. – 121 с.

102.Иванова Г. С. Технология программирования : учебник для вузов / Г. С. Иванова. – М.: Изд-во МГТУ им. Баумана, 2002. – 320 с.

103.Избачков Ю.С. Информационные системы : учебник для вузов. / Ю.С. Избачков, В.Н. Петров. – СПб.: Питер, 2006. – 656 с.

104.Кватрани Т. Rational Rose 2000 и UML. Визуальное моделирование / Т. Кватрани. – М.: ДМК Пресс, 2001. – 176 с.

105.Керневски Д. Рефакторинг с использованием шаблонов / Д. Керневски. – М.: ООО «И.Д. Вильямс», 2006. – 400 с.

106.Ковшов А.Н. Информационная поддержка жизненного цикла изделий машиностроения : принципы, системы и технологии CALS / ИПИ : учеб. пособие для студ. высш. учеб. заведений /А.Н. Ковшов, Ю.Ф. Назаров, И.М. Ибрагимов и др. – М.: Издательский дом «Академия», – 2007.– 304 с.

107.Константайн Л. Разработка программного обеспечения / Л. Константайн, Л. Локвуд. – СПб.: Питер, 2004. – 592 с.

108.Ларман К. Применение UML и шаблонов проектирования / К. Ларман. – М.: Издательский дом «Вильямс», 2004. – 624 с.

109.Леффиигуэлл Д. Принципы работы с требованиями к программному обеспечению.Унифицированный подход / Д. Леффиигуэлл, Д. Уидриг. – М.: Издательский дом «Вильямс», 2002. – 448 с.

110.Липаев В.В. Программная инженерия. Методологические основы / В.В. Липаев. – М.: «ТЕИС», 2006. – 610 с.

111.Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл. – СПб.: Питер, 2005. – 896 с.

112.Маклаков С.В. Создание информационных систем с AllFusion Modeling Suite / С.В. Маклаков. –М.: ДИАЛОГ-МИФИ, 2003. – 432 с.

113.Мандел Т. Разработка пользовательского интерфейса / Т. Мандел. – М.: ДМК Пресс, 2001. – 416с.

114.Мацяшек Л.А. Анализ и проектирование информационных систем с помощью UML 2.0 / Л.А. Мацяшек. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2008. – 816 с.

115.Мыльник В.В. Исследование систем управления: Учебное пособие для вузов / В.В. Мыльник, Б.П. Титаренко, В.А. Волочиенко. – М.: Академический Проект; Екатеринбург: Деловая книга, –2003. – 352 с.

116.Мюллер Р.Дж. Базы данных и UML. Проектирование / Р.Дж. Мюллер. – М.: «ЛОРИ», 2002. – 432с.

117.Нильсоон Д. Применение DDD и шаблонов проектирования: проблемно-ориентированное проектирование приложений с примерами на C# и .NET / Д. Нильсоон. – М.: ООО «И.Д. Вильямс», 2008. – 560 с.

118.Орлов С. Технологии разработки программного обеспечения : учебник / С. Орлов. – СПб. : Питер, 2002. – 464 с.

119.Перегудов Ф.И. Основы системного анализа / Ф.И. Перегудов, Ф.П. Тарасенко. – Томск: Изд-во НТЛ, 2001. – 396 с.

120.Проектирование информационных систем : курс лекций : учебное пособие для студентов вузов, обучающихся по специальностям в области информ. технологий / В.И. Грекул, Г.Н. Денищенко, Н.Л. Коровкина. – М. : Интернет-Ун-т Информ. технологий, 2005. – 304 с.

121. Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования. – Введ. 2001–07–02. – М. : ИПК Издательство стандартов, 2001. – 53 с.

122.Рамбо Дж. UML 2.0. Объектно-ориентированное моделирование и

разработка / Дж. Рамбо, М. Блаха. – СПб.: Питер, 2007. – 544 с.

123.Ройс У. Управление проектами по созданию программного обеспечения. Унифицированный подход / У. Ройс. – М. : Издательство «ЛОРИ», 1998. – 431 с.

124.Соломенцев Ю.М. Информационно-вычислительные системы в машиностроении CALS- технологии / Ю.М. Соломенцев, В.Г. Митрофанов, В.В. Павлов, А.В. Рыбаков. – М.: Наука, 2003.– 292 с.

125.Соммервилл И. Инженерия программного обеспечения / И. Соммервилл. – М. : Издательский дом «Вильямс», 2002. – 624 с.

126.Стелтинг С. Применение шаблонов Java. Библиотека профессионала / С. Стерлинг, О. Маасен. – М.: Издательский дом «Вильямс», 2002. – 576 с.

127.Тарасенко Ф.П. Прикладной системный анализ (Наука и искусство решения проблем) / Ф.П.Тарасенко. – Томск: Изд-во Том. ун-та, 2004. – 186 с.

128.Трофимов С.А. CASE-технологии: практическая работа в Rational Rose / С.А. Трофимов. – М.: Бином-Пресс, 2002. – 288 с.

129.Управляющие вычислительные комплексы: Учеб. пособие / Под ред. Н. Л. Прохорова. – 3-е изд.– М.: Финансы и статистика, 2003. – 352 с.

130.Фаулер М. UML. Основы / М. Фаулер. – СПб: Символ-Плюс, 2004. – 192 с.

131.Фаулер М. Архитектура корпоративных программных приложений / М. Фаулер. – М.: «Вильямс», 2006. – 544 с.

132.Федотова Д.Э. CASE-технологии: Практикум. / Д.Э. Федотова, Ю.Д. Семенов, К.Н. Чижик. – М.: Горячая линия–Телеком, 2005. – 160 с.

133.Халл Э. Разработка и управление требованиями. Практическое руководство пользователя / Э. Халл, К. Джексон, Д. Дик. – Оксфорд: Springer, 2005. – 240 с.

134.Хетагуров Я.А. Проектирования автоматизированных систем обработки информации и управления (АСОИУ): Ученик / Я.А. Хетагуров. – М.: Высш. шк., 2006. – 223 с.

.Хоп Г. Шаблоны интеграции корпоративных приложений / Г. Хоп, Б. Вульф. – М.: ООО «И.Д. Вильямс», 2007. – 672 с.

136.Черемных С.В. Структурный анализ систем: IDEF-технологии / С.В.

Черемных, И.О. Семенов, В.С. Ручкин. – М.: Финансы и статистика, 2003. – 208 с.

137.Шаврин О.И. Как формировать выводы по диссертации и составлять заключение диссертационного совета / О.И. Шаврин. – Ижевск: Изд-во ИжГТУ, 2002. – 28 с.

138.Якобсон А. Унифицированный процесс разработки программного обеспечения / А. Якобсон, Г. Буч, Дж. Рамбо. – СПб.: Питер, 2002. – 496 с.