Лекція №2 ПОіП « Поняття програмного виробу»

(Модуль 1 -)

План лекції.

1.3. Поняття програмного виробу

Суспільний поділ праці зумовлює перетворення сфери виробництва програмних засобів на складову суспільного матеріального виробництва.

Аналізуючи види програм, доходимо висновку, що програмні засоби мають ***властивості, притаманні товару*** (рис. 1.6), а отже, цілком правомірним є термін «програмний виріб», що означає програму на носії даних, яка є продуктом промислового виробництва [8].

Програмний виріб — це універсальне програмне забезпечення, призначене для широкого кола користувачів, яке має розроблятися, підтримуватися в роботоздатному стані й модифікуватися протягом тривалого часу.

Умови застосування програмних виробів висувають вимоги до виробництва, які суттєво відрізняються від процесу розробки програм для власного споживання (див. рис. 1.6).

Рис. 1.6. Вплив умов застосування ПЗ на вимоги до виробництва

Уже недостатньо, щоб програма якось працювала. Ідеться про забезпечення якісно нового рівня зручності щодо керування виконанням програми, підготовки початкових даних, оформлення та обробки результатів. Потрібно так налагодити програму, щоб користувач не дуже часто стикався з помилками, і передбачити захист від хибних дій користувача; забезпечити захист програм і даних; так документувати програму, щоб користувач мав змогу без додаткових консультацій зрозуміти правила роботи з нею і відрізняти свої помилки від помилок програми або збоїв ЕОМ, а також виправляти власні помилки.

Отже, програмні вироби плануються та проектуються так, щоб виконувалися певні вимоги; розробляються на основі ретельного вивчення потреб і впроваджуються тільки в тому разі, якщо попит на них може бути задоволений з користю для виробника.

1.4. Поняття життєвого циклу

Для управління процесом розробки великих програмних проектів користуються поняттям ***життєвого циклу***, який складається з таких етапів:

1) аналіз вимог до системи;
2) визначення специфікацій;
3) проектування;

4) кодування;
5) тестування;
6) експлуатація та супроводження.

Часові витрати на реалізацію окремих етапів можуть бути різними залежно від вибору методів, засобів, технологій, використовуваних на кожному етапі.

На етапі *аналізу вимог до системи* визначаються основні вимоги, виконання яких дають змогу дістати прийнятне вирішення проблеми: час обробки інформації, вартість обробки, імовірність помилки випадкової та навмисної. Такий підхід сприяє поглибленому усвідомленню проблеми, з'ясуванню компромісних ситуацій, вибору оптимального рішення.

Необхідно визначити просторово-часові обмеження, засоби системи, які можуть змінюватись у майбутньому, ресурси, необхідні для реалізації системи (фінансові, технічні, кадрові), можливості використання наявного програмного забезпечення, управління розвитком системи.

Притому слід розрізняти жорсткі вимоги й вимоги, виконання яких не є обов'язковим.

На етапі *визначення специфікацій* подається опис функцій, реалізовуваних на ЕОМ, задається структура вхідних і вихідних даних, засобів розміщення і зберігання даних, організовуються бази даних тощо.

Чим докладніші специфікації, тим менша ймовірність виникнення помилок, непорозумінь між замовниками та виконавцями. Дані для тестування системи мають узгоджуватись зі специфікаціями і подаватись на початкових етапах розробки системи.

На етапі *проектування* розробляються алгоритми відповідно до специфікацій, формується загальна структура системи в міру виконання специфікацій на окремі модулі та підмодулі.

Етап *кодування* вважається найпростішим, рутинним, його реалізація полегшується використанням алгоритмічних мов високого рівня, методів структурного, об'єктно-орієнтованого програмування та CASE-засобів. Дослідження [2] показали, що 64 % помилок вноситься у проект на етапі проектування, а 36 % — на етапі кодування. Етап кодування, який і вважався власне програмуванням, є найбільш освоєним.

Етап *тестування* необхідно ретельно спланувати, щоб зменшити витрати часу і коштів. Під час тестування використовуються дані, характерні для системи в робочому стані, які визначаються ще на етапі проектування. Тестування поділяється на три стадії:

- автономне (перевірка кожного модуля з використанням даних, підготовлених програмістом);
- комплексне (спільна перевірка груп взаємно залежних програмних модулів);
- системне (оцінювання за допомогою незалежних тестів замовника).

Якщо порівнюються характеристики кількох готових програмних комплексів для подальшого вибору одного з них, то виконується порівняльне тестування.

На етапі *експлуатації та супроводження* програмного комплексу виправляються помилки, не виявлені на етапі тестування; вносяться зміни, запропоновані замовником;

виконується модифікація комплексу для пристосування до нових умов функціонування; відпрацьовуються різні версії системи; контролюється копіювання системи.

У загальному часі життєвого циклу проекту частка цього етапу становить від 65 до 75 %.

1.5. Структурний підхід до проектування програмного забезпечення

Існує кілька підходів до проектування програмного забезпечення, але найбільш поширеними нині є структурний, або функціонально-модульний, та об'єктно-орієнтований. Вони різняться способом декомпозиції складних програмних систем. У першому випадку структура системи описується як ієрархічна система інформаційно пов'язаних модулів, що реалізують її функції. У другому використовується об'єктна декомпозиція, коли статична структура системи описується в термінах об'єктів і зв'язків між ними, а її поводження — у термінах обміну повідомленнями.

Практично всі поширені методи структурного підходу базуються на таких основних принципах [9]:

- «поділяй і владарюй»;
- ієрархічне впорядкування — організація системи у вигляді деревоподібної структури з додаванням нових деталей на кожному рівні;
- абстрагування — виокремлення істотних аспектів системи і відкидання неістотних;
- несуперечливість — узгодженість елементів системи;
- структурованість даних.

Структурний підхід використовує дві групи засобів, що описують функціональну структуру системи та відношення між даними, кожній з яких відповідають певні моделі:

- DFD — діаграма потоків даних;
- SADT (метод структурного аналізу та проектування) — моделі та функціональні діаграми;
- ERD — діаграми «сутність-зв'язок», найбільш популярні в CASE-засобах.

На кожному етапі життєвого циклу програмного забезпечення використовуються і відповідно інтерпретуються свої діаграми.

На етапах аналізу вимог до ПЗ і відповідного визначення специфікацій використовуються SADT-моделі.

Моделі ERD застосовуються для опису даних на концептуальному рівні, не залежному від засобів СКБД.

На етапі проектування, для опису структури проектованої системи ПЗ використовуються моделі DFD і ERD, які можуть уточнюватись, доповнюватись новими конструкціями для подання даних на логічному рівні.

Отже, сутність структурного підходу до розробки ПЗ полягає в його декомпозиції на автоматизовані функції та підфункції, які, у свою чергу, можуть розбиватися на нові функції, і т.д.

Проте кожна підфункція може містити тільки ті елементи, які належать функції (підфункції) попереднього рівня — «батьківського», причому вона повинна мати всі елементи, що є на «батьківському» рівні, який разом з його інтерфейсами забезпечує контекст підфункції нижчого рівня. Нічого до неї не можна додати або з неї вилучити. У цьому полягає і головна вада структурного підходу: процеси (функції) і дані існують окремо в моделі програмної системи, проектування ведеться від процесів до даних, тобто структури даних перебувають на другому плані. Крім того, такий підхід застосовний тільки у проектуванні «згори донизу», тобто у спадному проектуванні.

1.6. Об'єктно-орієнтований підхід

Як уже зазначалося, об'єктно-орієнтований підхід використовує об'єктну декомпозицію. Кожний об'єкт системи має власне поводження, яке є моделлю поводження об'єкта реального світу.

Основні неодмінні елементи об'єктної моделі такі:

- абстрагування;
- інкапсуляція;
- модульність;
- ієрархія.

Додаткові (не обов'язкові) елементи такі:

- типізація;
- паралелізм;
- стійкість.

Абстрагування — виокремлення найбільш зовнішніх характеристик об'єкта, що вирізняють його серед об'єктів інших видів.

Інкапсуляція — ізоляція інтерфейсу об'єкта від внутрішніх елементів об'єкта, що визначають його устрій і поводження, тобто його внутрішнє середовище.

Модульність — можливість декомпозиції системи на модулі, не пов'язані між собою.

Ієрархія — впорядкована за рівнями система абстракцій. Ієрархія за номенклатурою — це структура класів, а ієрархія за складом — структура об'єктів.

Типізація — здатність об'єктів перебувати в активному чи пасивному стані та розрізняти активні і пасивні об'єкти.

Стійкість — існування об'єкта у часі і просторі незалежно від процесу, який його створив.

Основні поняття об'єктно-орієнтованого підходу — *об'єкт* і *клас*.

Об'єкт — відчутна реальність, тобто предмет або явище, що має чітко визначені стан, поводження та індивідуальність.

Клас — множина об'єктів, пов'язаних спільністю структури та поводження. Об'єкт — представник класу.

З поняттям класу пов'язані поняття поліморфізму і успадкування.

Поліморфізм — здатність класу належати декільком типам.

Успадкування — можливість побудови нових класів на основі вже наявних додаванням або перевизначенням даних і методів.

Ця властивість забезпечує можливість еволюції системи, визначення нових функціональних можливостей, створення нових похідних класів — потомків базових класів, які успадковують характеристики батьківських класів без їх змін і додавання нових структур даних та методів роботи з ними.

Об’єктний підхід дає змогу узгоджувати моделі функціонування підприємства з моделями проектованої системи протягом усього життєвого циклу від першого етапу до останнього.

1.7. Зіставлення і взаємозв’язок структурного та об`єктно-орієнтованого підходів

У більшості людей поняття «проектування» асоціюється зі структурним проектуванням за методом «згори донизу» на основі функціональної декомпозиції, відповідно до якої вся система в цілому подається як одна велика функція і розбивається на підфункції, котрі, у свою чергу, розбиваються на підфункції, і т.д. Ці функції подібні до варіантів використання в об’єктно-орієнтованій системі, що відповідають діям, виконуваним системою в цілому.

Головний недолік структурного підходу такий: процеси і дані існують окремо один від одного (як у моделі діяльності організації, так і в моделі програмної системи), причому проектування ведеться від процесів до даних. Таким чином, крім функціональної декомпозиції існує також структура даних, що перебувають на другому плані.

В об’єктно-орієнтованому підході основна категорія об’єктної моделі — клас — поєднує в собі на елементарному рівні як дані, так і операції, що з ними виконуються (методи). Саме з цього погляду зміни, пов’язані з переходом від структурного до об’єктно-орієнтованого підходу, є найбільш помітними. Поділ процесів і даних подолано, однак лишається проблема подолання складності системи, що зважується із застосуванням механізму компонентів.

Дані порівняно з процесами є більш стабільною і менш змінюваного частиною системи. Звідси випливає головна позитивна властивість об’єктно-орієнтованого підходу, про яку Граді Буч сказав так: *Об’єктно-орієнтовані системи більш відкриті і легше піддаються внесенню змін, оскільки їхня конструкція базується на стійких формах. Це дає змогу системі розвиватися поступово і не приводить до повної її переробки навіть у випадку істотних змін вихідних вимог.*

Буч відзначає ще низку переваг *об’єктно-орієнтованого підходу:*

1. Об’єктна декомпозиція дає змогу створювати програмні системи меншого розміру використанням загальних механізмів, що забезпечують необхідну економію виражальних засобів. Застосування об’єктного підходу істотно підвищує рівень уніфікації розробки та придатність для повторного використання не лише програм, а й проектів, що зрештою веде до створення середовища розробки й переходу до складального створення ПЗ. Системи найчастіше виходять більш компактними, аніж їхні структурні еквіваленти, що означає не тільки зменшення обсягу програмного коду, але й здешевлення проекту за рахунок використання попередніх розробок.

2. Об’єктна декомпозиція зменшує ризик створення складних систем ПЗ, оскільки вона припускає еволюційний шлях розвитку системи на базі щодо невеликих

підсистем. Процес інтеграції системи розтягується на ввесь час розробки, а не перетворюється на одноразову подію.

3. Об'єктна модель цілком природна, оскільки передусім зорієнтована на людське сприйняття світу, а не на комп'ютерну реалізацію.

4. Об'єктна модель дає змогу повною мірою використовувати виражальні можливості об'єктних і об'єктно-орієнтованих мов програмування.

До *недоліків об'єктно-орієнтованого підходу* належать деяке зниження продуктивності функціонування ПЗ і високі початкові витрати. Об'єктна декомпозиція істотно відрізняється від функціональної, тому перехід на нову технологію пов'язаний як із подоланням психологічних труднощів, так і з додатковими фінансовими витратами. Безумовно, об'єктно-орієнтована модель найбільш адекватно відбиває реальний світ, що являє собою сукупність об'єктів, яківзаємодіють (за допомогою обміну повідомленнями). Але на практиці триває формування стандарту мови об'єктно-орієнтованого моделювання UML, і кількість CASE-засобів, які підтримують об'єктно-орієнтований підхід, невелика порівняно з тими, що підтримують структурний підхід. Крім того, діаграми, що відбивають специфіку об'єктного підходу (діаграми класів і т. ін.), значно менш наочні і важко сприймаються непрофесіоналами. Тому одна з головних цілей упровадження CASE-технології, а саме постачання всіх учасників проекту (зокрема і замовника) спільною мовою «для передання розуміння», забезпечується сьогодні тільки структурними методами.

У разі переходу від структурного підходу до об'єктного, як і за будь-якої зміни технології, необхідно вкладати кошти у придбання нових інструментальних засобів. Тут варто врахувати й витрати на навчання (оволодіння методом, інструментальними засобами і мовою програмування). Для деяких організацій ці обставини можуть стати серйозними перешкодами.

Об'єктно-орієнтований підхід не дає негайної віддачі. Ефект від його застосування починає позначатися після розробки двох-трьох проектів і нагромадження повторно використовуваних компонентів, що відбивають типові проектні рішення у відповідній галузі. Перехід організації на об'єктно-орієнтовану технологію — це зміна світогляду, а не просте вивчення нових CASE-засобів і мов програмування.

Таким чином, структурний підхід, як і раніше, зберігає свою значущість і доволі широко використовується на практиці. На прикладі мови UML добре видно, що його автори запозичали те раціональне, що можна було взяти зі структурного підходу: елементи функціональної декомпозиції в діаграмах варіантів використання, діаграми станів, діаграми діяльностей і т. ін. Однак очевидно, що у конкретному проекті виконати декомпозицію складної системи одночасно двома способами неможливо. Можна почати декомпозицію яким-небудь одним способом, а потім, використовуючи здобуті результати, спробувати розглянути систему з іншого боку.

Тепер перейдемо до розгляду взаємозв'язку між структурним і об'єктно-орієнтованими підходами. Основою взаємозв'язку є спільність низки категорій і понять обох підходів (процес і варіант використання, сутність і клас щодо). Цей взаємозв'язок може виявлятися в різних формах. Так, одним із можливих підходів є використання структурного аналізу як основи для об'єктно-орієнтованого

проектування. Такий підхід доцільний з огляду на значне поширення CASE-засобів, що підтримують структурний аналіз. Його можна вважати занадто прагматичним, але в деяких ситуаціях інший підхід неможливий. При цьому структурний аналіз варто припиняти, тільки-но діаграми потоків даних почнуть відбивати не тільки діяльність організації (предметну область), а й систему ПЗ.

Після виконання структурного аналізу і побудови діаграм потоків даних разом зі структурами даних та інших продуктів аналізу можна різними способами приступити до визначення класів і об'єктів. Так, якщо взяти яку-небудь окрему діаграму, то кандидатами в об'єкти можуть бути зовнішні сутності і нагромаджувачі даних, а кандидатами у класи — потоки даних.

Іншою формою прояву взаємозв'язку можна вважати інтеграцію об'єктної та реляційної технологій. Реляційні СУБД є сьогодні основним засобом реалізації великомасштабних баз даних і сховищ даних. Причини цього очевидні: реляційна технологія використовується доволі довго, освоєна численними користувачами й розроблювачами, стала промисловим стандартом, у неї вкладено значні кошти і створено безліч корпоративних БД у всіляких галузях, реляційна модель проста і має строгу математичну підставу; існує велика кількість промислових засобів проектування, реалізації й експлуатації реляційних БД. Унаслідок цього реляційні БД здебільшого використовуються для зберігання та пошуку об'єктів у так званих об'єктно-реляційних системах.

Об'єктно-орієнтоване проектування має точки дотику з реляційним проектуванням. Наприклад, як уже зазначалося, класи в об'єктній моделі можуть деяким чином відповідати сутностям (якщо, скажімо, ретельно проаналізувати всі подібності і розбіжності діаграм «сутність-зв'язок» і діаграм класів). Як правило, така відповідність існує тільки на ранній стадії розробки системи — стадії формування вимог. Надалі, зрозуміло, цілі об'єктно-орієнтованого проектування (адекватне моделювання предметної сфери в термінах взаємодії об'єктів) і розробки реляційної БД (нормалізація даних) розбігаються. Таким чином, єдино можливим засобом подолання утворюваного розриву є побудова відображення між об'єктно-орієнтованою і реляційною технологіями, що в основному зводиться до відображення між діаграмами класів і реляційною моделлю.

Одним із прикладів практичної реалізації взаємозв'язку між структурним і об'єктно-орієнтованим підходом є програмний інтерфейс (міст) між структурним CASE-засобом Silverrun і об'єктно-орієнтованим CASE-засобом Rational Rose, розроблений російською компанією «Аргуссофт». Цей міст створює *діаграми класів* Rational Rose на *основі RDM-моделі* (Relational Data Model — реляційна модель даних) Silverrun і навпаки. Аналогічні інтерфейси існують також між CASE-засобами ERwin (з одного боку), Rational Rose і Paradigm Plus (з іншого боку).

Тенденції розвитку сучасних інформаційних технологій призводять до постійного зростання складності інформаційних систем (ІС), створюваних у різних галузях економіки.

Вони сприяли появі програмно-технологічних засобів спеціального класу — CASE-засобів, що реалізують CASE-технологію створення і супроводження ІС. Термін «CASE» (Computer Aided Software Engineering) використовується нині в дуже

широкому сенсі. Первісне значення терміна «CASE», обмежене питаннями автоматизації розробки самого лише програмного забезпечення, тепер набуло нового трактування, що охоплює процес розробки складних ІС у цілому. Тепер під CASE-засобами розуміють програмні засоби, що підтримують процеси створення і супроводження ІС, включаючи аналіз і формулювання вимог, проектування прикладного ПЗ (додатків) і баз даних, генерацію коду, тестування, документування, забезпечення якості, конфігураційне керування і керування проектом, а також інші процеси. CASE-засоби разом із системним ПЗ і технічними засобами утворять повне середовище розробки ІС.

CASE-технологія являє собою методологію проектування ІС, а також набір інструментальних засобів, що дають змогу в наочній формі моделювати предметну сферу, аналізувати цю модель на всіх етапах розробки й супроводження ІС і розробляти додатки відповідно до інформаційних потреб користувачів. Більшість наявних CASE-засобів ґрунтується на методологіях структурного (переважно) чи об'єктно-орієнтованого аналізу і проектування, що використовують специфікації у вигляді діаграм чи текстів для опису зовнішніх вимог, зв'язків між моделями системи, динаміки поводження системи й архітектури програмних засобів.

Відповідно до огляду передових технологій (Survey of Advanced Technology), складеного фірмою Systems Development Inc. у 1996 році за результатами анкетування понад 1000 американських фірм, CASE-технологія потрапила до розряду найбільш стабільних інформаційних технологій (її використовувала половина всіх опитаних користувачів більш ніж у третині своїх проектів, з них 85 % завершилися успішно).

Контрольні питання

1. *Поняття програмного виробу.*
2. *Основні шляхи розвитку програмування.*
3. *Поняття технологічності створення програм.*
4. *Мета автоматизації програмування.*
5. *Складності автоматизації проектування програмних виробів.*
6. *У чому полягають основні принципи об'єктно-орієнтованого підходу?*
7. *Які переваги і недоліки має об'єктно-орієнтований підхід?*
8. *Які принципові розбіжності і що спільного між структурним та об'єктно-орієнтованим підходами?*