**Лекція №22. Компоненти повторного використання**

Однією з характерних рис інженерної діяльності є використання готових рішень або деталей. Однак усі, хто працює над створенням реальних систем, знають, що промислове використання готових рішень у програмній інженерії ще не стало повсякденною практикою. Якщо у світі працюють майже 8 мільйонів програмістів, то приблизно 80% з них працюють над створенням програм обліку й організаційного управління на кількох рівнях: окремого підрозділу фірми, окремого аспекту діяльності фірми, фірми в цілому, корпорації, галузі і, нарешті, держави. Це, переважно, задачі розрахунків, статистики, допомоги в прийнятті рішень при управлінні різними ресурсами − кадровими, фінансовими тощо.

За оцінками експертів, 75% цих робіт у світі дублюються − на тисячах підприємств створюються програми складського обліку, нарахування зарплати, розрахунку витрат на виробництво продукції, складання маршрутів деталей на виробничому конвеєрі тощо. Хоча більшість із цих програм типові, але кожного разу знаходяться особливості, що не дозволяють застосувати розроблену раніше програму. Тому нині активно розвивається напрям, водночас науковий та інженерний, який названо повторним використанням або компонентним розробленням програм. Компонентне програмування дозволяє уникнути цих проблем. Воно є подальшим розвитком ООП, заснованим на повторному використанні, специфікації компонентів і їхніх інтерфейсів, композиції та конфігурації компонентів. Зв’язки між компонентами містять у собі підтипи й еквівалентність, а об'єктні зв’язки — класи і суперкласи.

***Компонентне розроблення*** (*component development)* − це метод побудови програмного забезпечення з конструкцій за каталогом − як композиції готових компонент. Йдеться не лише про порції програмного коду або програмні модулі.

***Повторне використання*** *(reuse)* − це використання для нових розробок будь-яких порцій формалізованих знань, здобутих під час реалізації завершених розробок програмних систем.

Під ***компонентом***розуміють самостійний продукт, що підтримує об'єктну парадигму, реалізує окрему предметну область і може взаємодіяти з іншими компонентами через інтерфейси. Об'єкти розглядаються на логічному рівні проектування ПС, а компоненти – це безпосередня фізична, тобто програмна реалізація об'єктів. Співвідношення між об'єктами і компонентами неоднозначне. Один компонент може бути реалізацією декількох об'єктів або навіть деякої частини системи, отриманої при проектуванні. Зворотне співвідношення, тобто компонент – об’єкт, як правило, не виконується.

Перехід до компонентів відбувався еволюційно (табл. 1.): від підпрограм, модулів і функцій. При цьому удосконалилися елементи, методи їхньої композиції і накопичення для подальшого використання.

Компоненти конструюються самостійно, як деяка абстракція, що містить у собі інформаційну частину й артефакт (специфікація, код, каркас і ін.).

В *інформаційній частині* задаються відомості: призначення, дата виготовлення, умови застосування (ОС, середовище, платформа і т.п.).

Таблиця 1. Схема еволюції повторних елементів компонентного типу



*Артефакт* – це реалізація (implementation), інтерфейс (interface) і схема розгортання (deployment) компонента.

*Реалізація* – це код, що буде виконуватися при зверненні до операцій, визначених в інтерфейсах компонента. Компонент може мати кілька реалізацій залежно від операційного середовища, моделі даних, СКБД і ін.

Для опису компонентів, як правило, застосовуються мови об’єктно-орієнтованого програмування, зокрема, Java, у якій поняття інтерфейсу і класу – базові, використовуються в моделях Javabeans і Enterprise Javabeans і в об'єктній моделі CORBA[[1]](#footnote-1).

*Інтерфейс* відображає операції звертання до реалізації компонента, описується в мовах IDL або APL, містить у собі опис типів і операції передачі аргументів і результатів для взаємодії компонентів. Компонент, як фізична сутність, може мати множину інтерфейсів.

*Розгортання* – це виконання фізичного файлу відповідно до конфігурації (версії), з урахуванням параметрів запуску системи.

Компоненти зберігаються у вигляді класів компонентів і використовуються в компонентній моделі, композиції й у каркасі інтегрованого середовища. Керування компонентами проводиться на архітектурному або інтерфейсному рівні, між ними існує взаємний зв'язок.

Компонент описується мовою програмування, не залежить від операційного середовища (наприклад, від середовища віртуальної машини Java) і від реальної платформи (наприклад, від платформ у моделі CORBA), де він буде функціонувати.

В основі створення ПВК лежить ідея виявлення так званих родових знань про проблеми, які розв'язуються програмними системами. Цим терміном позначаються знання, котрі сприяють вирішенню кожного з певного кола завдань (таке коло визначає певну проблемну галузь). Родові знання виявляються як результат абстракції артефактів програмного забезпечення. Під артефактами мають на увазі будь-які продукти діяльності фахівців з розробки програмного забезпечення (ПЗ).

Типи артефактів при цьому не обмежуються фрагментами коду. Вони можуть бути структурами проекту, структурами реалізації модулів, документації, трансформації і т.д.

За величезної різноманітності форм ПВК, які використовуються у комп'ютерній практиці, є безперечна спільність у техніці їхнього застосування. Всі вони конструюються в процесі абстрагування і застосовуються шляхом виконання процесів селекції, спеціалізації й інтеграції. Перелічені чотири процеси визначають, так би мовити, чотири виміри, в аспектах яких ПВК можуть абстрагуватися, класифікуватися, добиратися і зіставлятися. Розглянемо ці процеси докладніше.

*Абстракція*. Абстракція є визначальною властивістю ПВК, кожна абстракція - продукт аналізу ознак зібрання сутностей, для котрого виділяються суттєві спільні ознаки й відкидаються несуттєві з певного погляду. Внаслідок цього визначається абстрактна або родова сутність, що матеріалізується як ПВК.

Кожна абстракція може розглядатися як така, що має дві частини - видиму та приховану. Видима частина є специфікацією тих знань, які необхідні для використання ПВК. Прихована частина містить деталі реалізації ПВК, невидимі на рівні її специфікації.

Видиму частину, у свою чергу, доцільно розглядати як сукупність стабільної або фіксованої та змінної складових.

Змінна частина являє собою змінні властивості ПВК, її здатність адаптуватися до якогось спектра застосувань, фіксована частина - інваріанти. Тобто, специфікація абстракції із змінною частиною відображається в множину альтернативних реалізацій.

Розподіл абстракції на видиму фіксовану і змінну та приховану частини є не внутрішньою властивістю абстракції, це передовсім проектні рішення розробника абстракції, які він приймає, даючи відповіді на такі запитання:

* яка інформація потрібна користувачу? Потрібна користувачу інформація подається як специфікація (видима частина);
* які з властивостей абстракції користувачу доведеться змінювати? Такі властивості доцільно подати у змінній частині специфікації.

*Добір або селекція*. Ключовою особливістю технологій використання ПВК є та обставина, що, приступаючи *до* нової програмної розробки, користувач не має відомостей про готові компоненти, тому у ПВК має бути чітка і виразна специфікація абстракції, котра *давала* б користувачеві можливість ефективно переглядати, розуміти, зіставляти і добирати підходящі для нього ПВК із числа наявних.

Подібно до бібліотечного пошуку, на допомогу користувачу може бути надано спеціальні каталоги й класифікатори зібрань компонент, призначених здійснювати навігацію пошуку і полегшувати розуміння сутності ПВК.

*Спеціалізація*. Як було зазначено вище, зазвичай ПВК є матеріалізацією певної абстракції або родового артефакту, який неможливо безпосередньо використати. Для його використання необхідно спеціалізувати або конкретизувати його, тобто провести операцію, обернену до абстракції, зміст котрої полягає в тому, щоб до родових ознак, властивих кожному з "представників роду", додати ознаки конкретного застосування - параметри, трансформації, обмеження тощо. Родовий артефакт є, по суті, абстракцією зі змінною частиною. Спеціалізація родового артефакту - це отримання реалізації абстракції на основі довизначення змінної частини ознаками конкретного застосування.

*Інтеграція*.Для ефективної інтеграції розрізнених артефактів, втілених у ПВК, до конкретної програмної системи потрібно, щоб користувач розумів інтерфейс артефакту, тобто особливості його взаємодії з іншими артефактами або з якимсь каркасом, який певною мірою можна вважати базою інтеграції. Можна сказати, що інтерфейс артефакту - це абстракція, в якій не враховуються внутрішні властивості артефакту, специфікація котрої визначає правила взаємодії артефактів між собою.

Для оцінювання ефективності абстракції вводиться інтуїтивна міра, яку названо *когнітивною відстанню (cognitive distance).* Цим терміном позначається кількість інтелектуальних зусиль, які необхідно витратити розробникові програмної системи для того, щоб перевести її з однієї стадії життєвого циклу в іншу. Для створення програмної системи за допомогою ПВК розробник ставить за мету мінімізацію когнітивної відстані між початковою концепцією системи та її кінцевою реалізацією, готовою до виконання.

Засобами досягнення цієї мети є:

- використання фіксованих і змінних абстракцій, лаконічних і виразних водночас;

- максимізація прихованої частини абстракції;

- застосування автоматичного відображення специфікації абстракції в її реалізацію (наприклад, шляхом компіляції).

Якщо поглянути на розвиток у часі засобів створення програм з точки зору повторного використання, то історична картина виглядає як поширення застосувань комп'ютера на нові і нові домени, набуття практики вирішення окремих завдань в рамках цих доменів, узагальнення набутого досвіду як накопичення типових абстракцій та втілення їх у ПВК певної форми. Історично першим доменом застосування програмних засобів була галузь числових розрахунків, для якої було створено і перші ПВК, котрі мали вигляд спочатку типових програмних модулів (так званих підпрограм - аналог сучасних бібліотечних функцій), а згодом і мов програмування. Перше покоління мов програмування було втіленням типових абстракцій алгоритмічних обчислень (мови програмування Фортран, Алгол-60 та подібні їм).

Друге покоління з'явилось як відповідь на поширення комп'ютерних методів на домени оброблення великих обсягів економічної інформації, завдяки чому було виділено абстракції типових процесів оброблення даних, збирання програм з окремих модулів (Кобол, PL/1, Паскаль, Модула тощо).

Поява візуальних можливостей обміну інформації з комп'ютером покликала до життя наступні покоління ПВК, подані абстракціями комп'ютерної графіки, діалогової взаємодії, узагальненою концепцією інтерфейсу. Ці узагальнення втілено в мови програмування (C++, Java тощо) та багато інших форм подання узагальненого досвіду розв'язання певних проблем у вигляді ПВК.

**Типи компонентних структур.**

Розширенням поняття компонента є *шаблон* (паттерн) – абстракція, що містить у собі опис взаємодії сукупності об'єктів у загальній кооперативній діяльності, для якої визначені ролі учасників і їхня відповідальність. Шаблон є повторюваною частиною програмного елемента або схемою вирішення проблеми.

*Компонентна модель –* відбиває проектні рішення щодо композиції компонентів, визначає типи шаблонів компонентів і припустимі взаємодії між ними, а також джерело формування файлу розгортання ПС у середовищі функціонування.

*Каркас* являє собою високорівневу абстракцію проекту ПС, у якій функції компонентів відділені від задач керування ними. Каркас поєднує множину взаємодіючих між собою об'єктів у деяке інтегроване середовище для досягнення певної кінцевої мети. Залежно від спеціалізації каркас називають «білою скринькою» або «чорною скринькою».

Каркас типу «біла скринька» містить у собі абстрактні класи для зображення об'єктів і їх інтерфейсів. При реалізації ці класи трансформуються в конкретні класи з указівкою відповідних методів реалізації. Використання такого типу каркаса є характерним для OOП.

Для каркаса типу «чорна скринька» у його видиму частину виносять точки, що дозволяють змінювати входи і виходи.

*Компонентне середовище* – розширення класичної моделі клієнт – сервер з урахуванням специфіки побудови і функціонування програмних компонентів, а також результатів практичних реалізацій і їхньої апробації. Основа компонентного середовища – множина серверів компонентів (часто їх називають сервери застосувань – application servers). Усередині сервера розгортаються компоненти-контейнери. Для кожного сервера може існувати довільна кількість контейнерів.

*Контейнер* – це оболонка, усередині якої реалізується функціональність компонента. Взаємодія контейнера із сервером строго регламентована і здійснюється через стандартизовані інтерфейси. Контейнер керує породжуваними компонентами і їхніми екземплярами з відповідною функціональністю. У загальному випадку усередині нього може існувати довільна кількість екземплярів-реалізацій, кожна з яких має унікальний ідентифікатор.

**Інтерфейс компонентів.**

З компонентами у складі контейнера зв'язані два типи інтерфейсів: один для взаємодії з іншими компонентами, а другий **–** інтерфейс системних сервісів, необхідних для функціонування самого контейнера і реалізації спеціальних функцій, наприклад, підтримка розподілених транзакцій, у яких беруть участь кілька компонентів.

Перший тип інтерфейсу — домашній (Home interface) забезпечує керування екземплярами компонента з обов'язковими реалізаціями методів пошуку, створення і видалення окремих екземплярів.

До другого типу належать функціональні інтерфейси (fuction interface), що забезпечують доступ до реалізації компонента. Фактично з кожним екземпляром зв'язаний свій функціональний інтерфейс.

Екземпляри компонентів контейнера можуть взаємодіяти за допомогою системних сервісів, розміщених в інших компонентах. Самі компоненти можуть розміщатися як усередині одного сервера, так і в різних серверах для різних платформ. Така взаємодія забезпечується унікальними іменами компонентів і екземплярів, а також регламентована інтерфейсами і системними функціями.

Інтерфейс відображає перелік сервісів, вхідні та вихідні параметри сервісів та їхні типи, переду- і постумови функціонування компонента, а також перелік інших компонентів, сервіси яких потрібні для здійснення своїх дій як сервісів.

Архітектура компонентного середовища може складатися з наступних типів об'єктів:

– сервери компонентів;

– контейнери компонентів;

– реалізації функцій, подані як екземпляри усередині контейнерів;

– реалізація компонентних моделей, об'єктів, що задовольняють установку і конфігурування окремих компонентів для деякої комп'ютерної платформи;

– клієнтські компоненти і інтерфейси, що забезпечують кінцевого користувача, реалізовані у вигляді різних типів клієнтів (веб-клієнти, повноцінні реалізації графічного інтерфейсу і т.д.);

– компонентне застосування, як сукупність компонентів.

Кожний з типів об'єктів може реалізовуватися окремо, оскільки для кожного з них існують свої специфікації і вимоги, а також правила взаємодії з іншими елементами середовища компонентного програмування. Всі елементи разом узяті утворюють ланцюжок, що визначає порядок реалізації компонентної ПС. Кожен тип елементів може реалізовуватися окремим розробником і відповідно до цього визначається його роль у процесі створення компонентної системи.

Згідно до розподілені об'єктів на типи і з огляду на визначене місце кожного з них у процесі створення компонентної програми, варто зробити висновок, що ЖЦ компонентної системи значно складніше, ніж ЖЦ в інших підходах до програмування. Фактично мова йде про декілька окремих ЖЦ для кожного типу компонентів.

Композиція (інтеграція) компонентів і розгортання не залежать від ЖЦ розробки компонентів, заміна будь-якого компонента на новий компонент не повинна призвести до перекомпіляції або перенастроювання зв'язків у ПС.

Інтерфейс компонента може бути визначений у вигляді специфікації точок доступу до компонента, що обумовлюють його варіантність, і використовуючи які клієнт одержує сервіс у клієнт-серверному середовищі. Виходячи з того, що інтерфейс не надає реалізацію операцій, можна змінювати реалізацію без зміни інтерфейсу і, таким чином, покращувати функціональні властивості компонента без перебудови ПС у цілому, а також додавати нові інтерфейси (і реалізацію) без зміни існуючої реалізації усієї ПС.

Семантика інтерфейсу компонента може бути представлена за допомогою *контрактів*, що визначають зовнішні обмеження і підтримують інваріант, який містить у собі правила встановлення взаємозв’язків властивостей компонента або умови його життєздатності. Крім того, для кожної операції компонента контракт може визначати обмеження, що повинні бути враховані клієнтом перед викликом операції (передумова), і постумови перевірки правильності функціонування компонента після завершення операції. Перед- і постумова визначають специфікацію поведінки компонента і залежать від стану компонента, а також інтерфейсу і зв'язаним з ним набором інваріантів.

Контракти й інтерфейс зв'язані між собою, але їхні сутності різні. Інтерфейс являє собою колекцію операцій або функціональних властивостей специфікації сервісів, що підтримує компонент.

Контракт задає опис поведінки компонента, націлений на взаємодію з іншими компонентами і відбиває семантику функціональних властивостей компонента. Таким чином, модель специфікації семантики компонента визначає його інтерфейс і обмеження. Кожен інтерфейс складається з набору операцій (сервісів, що він пропонує або потребує). З кожною операцією зв'язаний набір перед- і пост- умов.

**Структури з компонентів.**

Композиція компонентів може бути таких типів:

* компонент з компонентом зв’язані через інтерфейс на рівні застосування;
* каркас з компонентом зв’язані через інтерфейси на системному рівні;
* компонент з каркасом взаємодіють через інтерфейси на сервісному рівні;
* каркас з каркасом взаємодіють через інтерфейси на мережному рівні.

Компоненти запам'ятовуються в репозитарії компонентів, а їхні інтерфейси – в репозитарії інтерфейсів. Компоненти і їхні композиції, як правило, запам'ятовуються в репозитарії компонентів, а їхні інтерфейси також в репозитарії інтерфейсів.

*Повторне використання* в компонентному програмуванні – це застосування готових порцій формалізованих знань, здобутих під час попередніх реалізацій ПС, у нових розробках.

*Компоненти повторного використання (КПВ)* – це готові компоненти, елементи оформлених знань (проектні рішення, функції, шаблони й ін.), що використовуються у ході розроблення не тільки самими розробниками, а й іншими користувачами шляхом адаптації їх до нової ПС, що спрощує і скорочує терміни її розробки. В Інтернеті у даний момент є багато різних бібліотек, репозитаріїв, що містять у собі КПВ, які можна використовувати в нових проектах.

При створенні компонентів, орієнтованих на повторне використання, їхні інтерфейси повинні містити операції, що забезпечують різні способи застосування компонентів. КПВ повинні відповідати визначеним вимогам, мати характерні властивості і структуру, а також механізми звертання до них.

Головною перевагою створення ПС із компонентів є зменшення витрат на розробку за рахунок вибору готових компонентів з подібними функціями, придатними для практичного використання, і пристосування їх до нових умов, на що витрачається менше зусиль, ніж на аналогічну розробку нових компонентів.

Пошук готових компонентів ґрунтується на методах класифікації і каталогізації. Методи класифікації призначені для подання інформації про компоненти з метою швидкого пошуку і добору. Методи каталогізації – для їхнього фізичного розміщення в репозитаріях із забезпеченням доступу до них у процесі інтеграції в компонентну систему.

**Методологія компонентної розробки ПС.**

Створення компонентної системи починається з аналізу предметної області і побудови концептуальної моделі, на основі якої створюється компонентна модель (рис. 1). Вона містить у собі проектні рішення щодо композиції компонентів, використання різних типів шаблонів, зв'язків між ними й операцій розгортання ПС у середовищі функціонування.

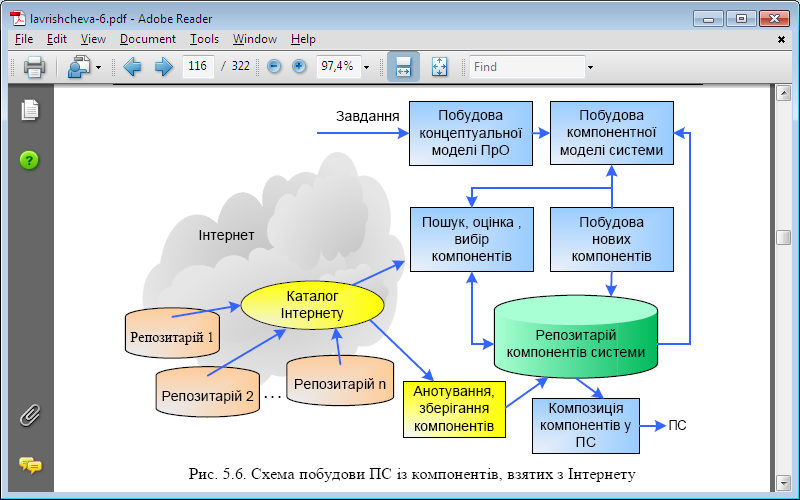


Рис. 1. Схема побудови ПС із компонентів, взятих з Інтернету

Готові компоненти беруться, наприклад, з репозитаріїв у Інтернеті і використовуються при створенні програмних систем, технологія побудови яких описується наступними етапами ЖЦ.

1. *Пошук, вибір КПВ* і розроблення нових компонентів, виходячи із системи класифікації компонентів і їхньої каталогізації, формалізоване визначення специфікацій інтерфейсів, поводження і функціональності компонентів, а також їхнього анотування і розміщення в репозитарії системи або в Інтернеті.

2. *Розроблення вимог* (Requirements) до ПС – це формування й опис функціональних, нефункціональних і інших властивостей ПС.

3. *Аналіз поведінки* (Behavioral Analysis) ПС полягає у визначенні функцій системи, деталей проектування і методів їхнього виконання.

4. *Специфікація інтерфейсів і взаємодій компонентів* (Interface and Interaction Specification) віддзеркалює розподіл ролей компонентів, інтерфейсів, їхню ідентифікацію і взаємодію компонентів через потоки дій або робіт (workflow).

5. *Інтеграція набору компонентів і КПВ* (Application Assembly and Component Reuse) у єдине середовище ґрунтується на підборі й адаптації КПВ, визначенні сукупності правил, умов інтеграції і побудові конфігурації каркаса системи.

6. *Тестування компонентів і середовища* (Component Testing) ґрунтується на методах верифікації і тестування для перевірки правильності як окремих компонентів і КПВ, так і інтегрованої з компонентів програмної системи.

7. *Розгортання (System Deployment)* містить у собі оптимізацію плану компонентної конфігурації з урахуванням середовища, розгортання окремих компонентів і створення цільової компонентної конфігурації для функціонування ПС.

8. *Супровід ПС* (System Support and Maintenance) складається з аналізу помилок і відмов при функціонуванні ПС, пошуку і виправлення помилок, повторного її тестування й адаптації нових компонентів до вимог і умов інтегрованого середовища.

Накопичений досвід розроблення систем програмного забезпечення може бути зафіксовано в різних формах, починаючи від конкретних параметризованих програмних модулів і закінчуючи програмними архітектурами та середовищами. Таким чином, ПВК - цеелементи знань про минулий досвід розроблення систем програмування, якщо:

а) їх можуть використовувати не лише їхні розробники;

б) їх можна адаптувати для створення нових систем. Зауважимо, що при цьому потрібен каталог, з якого можна зрозуміти, які деталі є і якїх можна поєднувати в конструкцію.

Тож повторне використання має включати систематичну цілеспрямовану діяльність зі створення каталогу. Ми називаємо таке повторне використання *систематичним використанням (system reuse),* підкреслюючи цим, що йдеться не про компоненти, створені внаслідок розроблення програмної системи, після завершення якої "випадково" з'ясувалося, щоці компоненти можуть ще комусь стати в нагоді. Досвід реінженерії готових програмних продуктів показав безперспективність пошуку таких знахідок. На відміну від нього, систематичне повторне використанняє капіталомістким підходом, що передбачає наявність двох явно виділених процесів в життєвому циклі розробки програмних систем.

*Перший процес − створення ПВК*.Він включає такі кроки:

а) вивчення спектра завдань, що вирішуються, виявлення серед них загальних підходів до вирішення;

б) побудову для них компонент, які реалізують знайдені підходи або окремі їхні елементи, котрі ми назвали повторно використовуваними компонентами;

в) побудову каталогу, націленого на пошук необхідних компонент. Для успішної реалізації цього процесу необхідно мати певний досвід у вирішенні не одного, а кількох подібних завдань, що дозволяє виявити як їхні спільні риси, так і розбіжності, щоб знайти загальне вирішення для їхньої реалізації, а також способи настроювання на характерні для кожного завдання особливості.

*Другий процес − конструювання цільових систем з готових компонент*.Він передбачає такі кроки:

а) зрозуміти, що має робити нова цільова система, для чого вона створюється і які вимоги до неї ставляться;

б) знайти у каталозі серед готових компонент ті, які вважаються підходящими, і зрозуміти, що вони роблять;

в) зіставити мету нової розробки з можливостями знайдених ПВК і прийняти рішення про доцільність використання їх;

г) застосувати відібрані ПВК й інтегрувати їх до нової розробки, забезпечивши необхідні поєднання.

Створення ПВК потребує вкладення капіталу, а використання дозволяє отримати зиск за рахунок економії трудовитрат. Як і всі інвестиції, інвестиції в повторне використання потребують дослідження й оцінювання ефективності вкладень капіталу, прогнозування термінів та обсягів повернення цих вкладень, оцінювання ризиків та інших традиційних бізнес-процесів. Інакше кажучи, бізнес повторного використання, як будь-який бізнес, потребує спеціальних зусиль для менеджменту всіх його процесів. Критеріями успіху такого бізнесу є:

а) забезпечення повторного використання меншими трудовитратами, ніж розроблення програмних систем як разових продуктів;

б) забезпечення пошуку придатних для використання компонент меншими трудовитратами, ніж нове розроблення їхніх функцій для потреб системи, що проектується;

в) забезпечення настроювання компонент на нові умови меншими трудовитратами, ніж нова розробка.

Провідні світові виробники програмного забезпечення широко інвестують дослідницькі проекти з технологій повторного використання та накопичення ПВК.

Програми розвитку повторного використання досліджуються і на загальнонаціональному рівні (відомі загальнонаціональні програми Японії, Голландії, Великобританії, Італії; американські програми департаменту оборони, NASA, військово-повітряних сил США тощо), і на рівні провідних світових корпорацій (IBM, ERICSSON, MICROSOFT, HEWLETT-PACKARD та інші).

**Таким чином**, компонентне програмування є основою економії витрат на програмування нових програм за рахунок використання готових КПВ, які можуть зберігатися у різних сховищах (бібліотеках, репозитаріях, базах знань тощо).

***Контрольні запитання для самоперевірки***.

1. Наведіть основні типи компонентів і шляхи їхнього використання.

2. Назвіть базові поняття в компонентному програмуванні.

3. Визначте основні поняття й етапи життєвого циклу у компонентному програмуванні.

4. Назвіть дві характерні властивості повторно використовуваних компонент (ПВК).

**Рекомендована література**

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.

***Для самостійного вивчення***: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

1. **CORBA** (англ. *Common Object Request Broker Architecture* — загальна архітектура брокера об'єктних запитів) — це запропонований [консорціумом OMG](https://uk.wikipedia.org/wiki/Object_Management_Group) [технологічний стандарт](https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D1%96%D1%87%D0%BD%D0%B8%D0%B9_%D1%81%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82) розробки розподілених застосунків. Завдання CORBA — інтегрувати розподілені системи, дати можливість програмам, що написані різними мовами та працюють у різних вузлах мережі, взаємодіяти одна з одною так само просто, наче вони знаходяться в адресному просторі одного процесу. [↑](#footnote-ref-1)