УДК 004.4’2

к.т.н., доц. **Муляр І.В.** (ХмНУ)
Гнатюк В.В. (ХмНУ**)**
Солодєєва Л.В. (ВІКНУ)

МЕТОД ВИЗНАЧЕННЯ ШАБЛОНУ ПРОЕКТУВАННЯ ВИХІДНОГО КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У статті проведено аналіз доступних способів проектування, та запропоновано набір кроків для вибору шаблону проектування вихідного коду програмного забезпечення.

На основі результатів дослідження визначено основні типи шаблонів проектування, особливість кожного з яких полягає в завданнях які вони вирішують, а саме: дати чітке пояснення проблеми, та її вирішення у відповідній галузі застосування.

Запропоновано метод та розроблено алгоритм визначення шаблону проектування вихідного коду програмного забезпечення, який дозволятиме обирати найбільш підходящий шаблон проектування, що на ранніх етапах розробки програмного забезпечення вирішуватиме проблему структуризації вихідного коду.

Ключові слова: проектування, вихідний код, програмне забезпечення.

Вступ. Близько двадцяти років тому з’явився новий напрямок досліджень, яке називають архітектурою підприємства. На початку цей напрям був призначений для вирішення наступних проблем:

- Складність системи – у підприємств постійно збільшувались витрати на створення IT-систем;
- Неефективна організація бізнесу - незважаючи на постійне збільшення вартості IT-систем, підприємствам з великими зусиллями вдавалось підтримувати їхню відповідність до вимог бізнесу ;

В результаті замовник отримував високі затрати та низьку ефективність. В силу цих обставин, за останні двадцять років було розроблено безліч методологій побудови архітектури підприємств, найпопулярнішими з яких є:

- Структура Захмана для архітектури підприємств;
- TOGAF (The Open Group Architectural Framework);
- Архітектура федеральної організації;
- Методологія Gartner;

З часом ці методології почали використовувати не лише для глобальних задач, таких як побудова IT-систем, а й для розробки якісного програмного забезпечення.

За сучасних умов, коли за короткий проміжок часу, потрібно створювати якісний та ефективний програмний продукт, однією з вимог до розробників ПЗ, стала вимога структуризації програмного коду.

Постановка проблеми. Більшість завдань, які часто зустрічаються розробникам, вже давно вирішені іншими розробниками. Різні методи структуризації коду, або ж шаблони проектування – той засіб, з допомогою якого програмісти можуть ділитись один з одним накопиченим досвідом. Як тільки шаблон стає загальнодоступним, він збагачує мови програмування та дозволяє поділитись з іншими розробниками, новими ідеями проектування та їх результатами. З допомогою шаблонів проектування легко виділити загальні задачі, визначити перевірені рішення та описати можливі результати.

Але відколи процес розробки програмного забезпечення не обходиться без процедури структуризації програмного коду та використання шаблонів проектування, було винайдено безліч програмних рішень для вирішення тої чи іншої задачі. Шаблони проектування досягли того розвитку, що стали поділятись на декілька типів, а кожний тип мав безліч рішень та розширювався на спектр різних задач. Тому з’явилась проблема визначення підходящого, для конкретного завдання, шаблону проектування.

У зв’язку з цим, важливо організувати механізм, який би визначав необхідний розробнику шаблон проектування, в залежності від задачі.

Виклад основного матеріалу досліджень. Управління будь-яким проектом припускає його поділ на окремі блоки, які є самостійними об'єктами планування, обліку й координування, тобто побудови структури проекту. В залежності від поставлених задач, шаблони проектування можна поділити на три групи:

- Породжуючі шаблони (Creational patterns);
- Структурні шаблони (Structural);
- Шаблони поведінки (Behavioral patterns);

Кожна з груп шаблонів виконують задачу структуризації коду, в залежності від поставленого завдання та результатів, які очікуються від готового компонента програми або продукту. Завдання, програмна реалізація яких, потребує створення нових об'єктів, є однією з найпоширеніших задач, які доводиться виконувати розробникам програмних систем. Породжуючі шаблони проектування призначені для створення об'єктів, дозволяючи системі залишатись незалежною як від самого процесу створення, так і від типів створених об'єктів. Це можна показати на прикладі гри де можуть бути воїни трьох типів: піхота, стрільці та кіннота. Кожний з цих типів має свої відмінності, такі як зовнішній вигляд, бойова міць, швидкість пересування та ступінь захисту. Але незважаючи на це, у всіх них є загальні властивості які притаманні всім типам одночасно. Наприклад, всі вони можуть пересуватись по ігровій карті в різних напрямках. Або кожна бойова одиниця має свій рівень здоров'я, і якщо він рівний нулю, воїн гине. Для забезпечення можливості розвитку такої гри, потрібно на початку проектування зробити гру максимально незалежною від конкретних типів персонажів, що дозволить в майбутньому додавати нових воїнів. Для цього достатньо використовувати наступну ієрархію класів:

```
class Warrior
{
public:
virtual void info() = 0;
virtual ~Warrior() {}
};
class Infantryman: public Warrior
{
public:
void info() { cout << "Infantryman" << endl; }
};
class Archer: public Warrior
{
public:
void info() { cout << "Archer" << endl; }
};
class Horseman: public Warrior
{
public:
void info() { cout << "Horseman" << endl; }
};
```

Поліморфний базовий клас Warrior визначає загальний інтерфейс, а похідні від нього класи Infantryman, Archer і Horseman реалізують особливості кожного з видів воїна. Складність полягає в тому, що якщо код створення кожного з персонажів хаотично розміщений по всьому додатку, додавання нових типів персонажів буде значно ускладнений. Тому в таких випадках на допомогу приходить «фабрика об'єктів », яка локалізує створення об'єктів.

«Фабрика об'єктів» дозволяє створювати об'єкти потрібних класів, не вказуючи напряму їх типів. В найпростішому випадку, для цього використовується ідентифікатори типів. Наступний приклад демонструє найпростіший варіант «фабрики об'єктів »

```
enum Warrior_ID { Infantryman_ID=0, Archer_ID, Horseman_ID };
 Warrior * createWarrior( Warrior_ID id  )
{
   Warrior * p;
   switch (id)
   {
      case Infantryman_ID:
         p = new Infantryman();
         break;
      case Archer_ID:
         p = new Archer();
         break;
      case Horseman_ID:
         p = new Horseman();
         break;
      default:
         assert( false);
   }
   return p;
}
```

Тепер, програмний код створення об'єктів різних типів ігрових персонажів, знаходиться в одному місці, а сама, в фабричній функції createWarrior(). Ця функція отримує в якості параметра тип об'єкта, який потрібно створити, створює його та повертає відповідний вказівник на базовий клас.

Не менш важливою властивістю програмного коду, є його здатність до повторного використання в одному або в кількох проектах, незалежність один від одного його компонентів, групування схожих об'єктів в деревовидні структури, розширення функціональності об'єктів, створення багаторівневого уніфікованого інтерфейсу до набору інтерфейсів деякої підсистеми, тощо. Ці та інші завдання компоновки системи на основі класів та об'єктів вирішують структурні шаблони проектування. При цьому можуть використовуватись наступні механізми:

- Наслідування, коли базовий клас визначає інтерфейс, а підкласи – реалізацію. Структури на основі наслідування статичні.;

- Композиція, коли структури будуються шляхом поєднання об'єктів деяких класів. Композиція дозволяє отримувати структури, які можна змінювати під час виконання програми ;

Як приклад можна розглянути структурний шаблон проектування «Adapter», який вирішує одну з проблем, яка з'являється в нових проектах, та полягає в неможливості повторного використання вже існуючого програмного коду. Наприклад, класи які вже є в наявності можуть мати потрібну функціональність, але при цьому, мати несумісні інтерфейси.

Шаблон проектування «Adapter», програмно приводить інтерфейси існуючих класів до вигляду, придатного до подальшого використання. Нехай клас, інтерфейс якого потрібно адаптувати до потрібного вигляду, називається Adaptee. Для вирішення задачі адаптації його інтерфейсу, шаблон «Adapter» вводить ієрархію класів яка показана на рисунку 1:

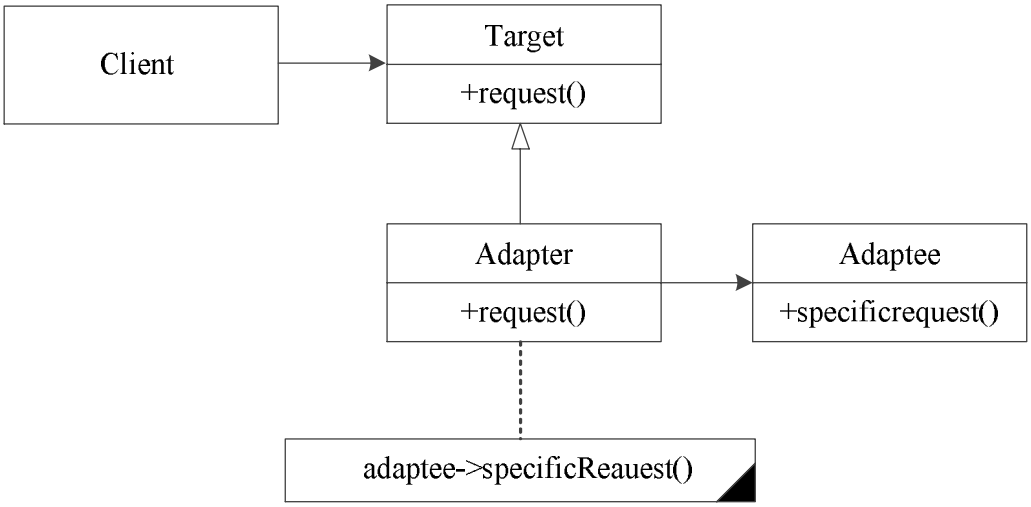

Рис. 1. Ієрархія класів для реалізації шаблону проектування «Adapter»

Віртуальний базовий клас Target – користувацький інтерфейс необхідного виду. Але цей інтерфейс доступний лише користувачу. Підклас Adapter, реалізує інтерфейс Target, та містить вказівники або посилання на екземпляр Adaptee. Шаблон «Adapter» використовує цей вказівник для пере направлення клієнтський викликів до Adaptee.

Останній тип шаблонів проектування – шаблони поведінки, які розглядають питання зв'язків між об'єктами та розподіл обов'язків між ними. Такий тип шаблонів вирішує завдання, пов'язані з інкапсуляцією взаємодії сукупності об'єктів в окремий об'єкт-посередник, отримання та збереження, за межами об'єкта, внутрішнього стану об'єкта так, щоб в майбутньому його можна було відновити в початковому стані, а також ситуації, коли необхідно реалізувати можливість зміни поведінки об'єкта, в залежності від його внутрішнього стану, тобто по суті створювати ілюзію, коли об'єкт начебто змінює свій клас.

Для прикладу буде розглянуто шаблон проектування «Iterator». Цей шаблон надає можливість послідовного доступу до всіх елементів ключового об'єкта, не розкриваючи його внутрішнього представлення. Абстракція в стандартних бібліотеках мов програмування C++, Java, C#, яка дозволяє розділити класи на колекції та алгоритми.

Складовий об'єкт, такий як список чи колекція, повинен надавати доступ до своїх елементів, приховуючи при цьому, свою внутрішню структуру. Більше того, іноді потрібно перебирати елементи списку різними способами, в залежності від конкретної задачі. Але накопичувати інтерфейс списку різноманітними операціями для різних переборів, навіть за їх необхідності дуже незручно. Крім того, іноді потрібно мати декілька активних переборів одного списку одночасно. Краще мати єдиний інтерфейс для перебору різних типів об'єктів, тобто реалізувати поліморфну ітерацію.

Шаблон проектування «Iterator» дозволяє все це робити. Ключова ідея полягає в перенесенні всієї відповідальності за доступ та перебір з основного об'єкта, на об'єкт Iterator, який буде визначати стандартний протокол перебору.

Для маніпуляції колекцією, клієнт використовує відкритий інтерфейс класу Collection. Але доступ до елементів колекції інкапсулюється додатковим рівнем абстракції, який буде називатись Iterator. Кожний похідний від Collection клас знає, який похідний від Iterator клас потрібно створювати і повертати. Після цього клієнт використовує інтерфейс, який визначений в базовому класі Iterator. Діаграма класів шаблону «Iterator» показана на рис. 2.

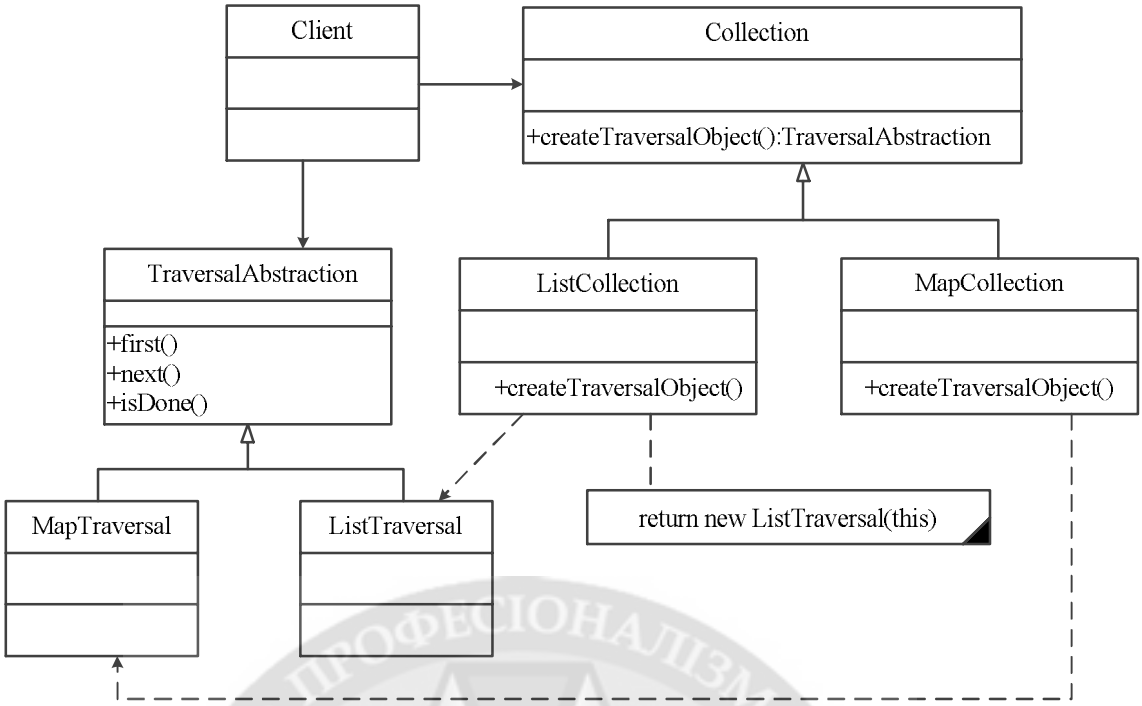

Рис. 2. Ієрархія класів для реалізації шаблону проектування «Iterator»

Більш детальне дослідження кожного типу шаблонів проектування, та визначення конкретних категорій завдань, дозволить створити послідовну систему кроків для виділення най підходящого серед безлічі можливих та відомих рішень для побудови нового програмного продукту. Під системою кроків мається на увазі метод, який буде працювати в декілька стадій, кожна з яких буде давати певний результат, та який впливатиме на результат наступної стадії. Умовно на даному етапі, весь процес вибору шаблону можна поділити на три основних стадії. На першій стадії, проводиться аналіз поставленого завдання та виділення критеріїв за допомогою яких можна поставити основні вимоги до шаблону проектування. Це дозволить серед трьох основних типів виділити один, най підходящий тип шаблону проектування.

Друга стадія, полягатиме у аналізі шаблонів проектування, які відповідають обраному на першій стадії, типу шаблону проектування, та поділу їх на дві умовні частини: на ті які можуть підійти, та ті які не відповідають поставленій задачі. Основою для цієї стадії, буде додатковий набір критеріїв, які будуть індивідуальними для кожного з типів шаблонів проектування та відповідати поставленій задачі

Третя стадія полягатиме в остаточному виборі одного або декількох шаблонів проектування. Можлива наявність в результаті не одного а декількох шаблонів проектування, обумовлена тим, що в залежності від складності розроблюваної програмної системи, її реалізація може вимагати комбінування різних рішень. Тому в майбутньому, планується удосконалити метод пошуку підходящого шаблону проектування для тих випадків, коли рішення може бути реалізоване з використанням кількох шаблонів проектування які будуть належати різним типам шаблонів проектування.

Висновки. На основі проведеного дослідження запропоновано метод вибору шаблонів проектування для вирішення проблем структурної організації коду програмного забезпечення. В основі методу лежить сукупність кроків які аналізують поставлене завдання та відокремлюють з всіх відомих шаблонів проектування, один єдиний, або декілька шаблонів проектування для вирішення поставленого завдання при умовах визначеної структурної організації коду програмного забезпечення.

ЛІТЕРАТУРА:
1. Слинкин А.А./Приемы объектно-ориентированного проектирования. Паттерны проектирования/ Э. Гамма, Р. Хелм, Р. Джонсон, Д.Влиссидес// СПб: «Питер», 2007. 366 с.
2. Сешнс Р./ Сравнение четырех ведущих методологий построения архитектуры предприятия/ Компания ObjectWatch,Inc, 2007. http://msdn.microsoft.com/ru-ru/library/ee914379.aspx
3. Хофстейдер Д./ Использование шаблонов для быстрой разработки/ Компания Avaya, 2008. https://msdn.microsoft.com/ru-ru/library/ee914376.aspx
4. Берштейни И.В./ Применение DDD и шаблонов проектирования. Проблемно-ориентированное проектирование приложений с примерами на C# и .NET/ Д.Нильссон// ООО «И.Д. Вильямс», 2008. – 560с.
5. Макконнелл С./ Совершенный код. Мастер-класс/ Пер. с англ. – М.: Издательство «Русская редакция», 2010. – 896 стр.: ил.

Рецензент: д.т.н., проф. Лєнков С.В., начальник науково-дослідного центру Військового інституту Київського національного університету імені Тараса Шевченка

к.т.н., доц. Муляр И.В., Гнатюк В.В., Солодеева Л.В.

МЕТОД ОПРЕДЕЛЕНИЯ ШАБЛОНА ПРОЕКТИРОВАНИЯ ИСХОДНОГО КОДА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В статье проведено анализ существующих способов проектирования, и предложено набор шагов для выбора шаблона проектирования исходного кода программного обеспечения.

На основании результатов исследования определено основные типы шаблонов проектирования, особенность каждого из которых заключается в заданиях которые они решают, а именно: точная постановка проблемы и ее решения в соответствующей сфере использования.

Предложено метод и разработано алгоритм определения шаблона проектирования исходного кода программного обеспечения, который позволяет выбрать наиболее подходящий шаблон проектирования, что на разных стадиях разработки программного обеспечения решает проблему структуризации исходного кода.

Ключевые слова: проектирование, исходный код, программное обеспечение.

Ph.D. MulyarI.V., Hnatiuk V.V., Solodeeva L.V.

METHOD OF DETERMINING DESIGN PATTERNS OF THE SOFTWARE SOURCE CODE

The article analyzes the available design methods, and proposes a set of steps for selecting a template design software source code.

Based on the research results identified the main types of design patterns, each feature of which is that they resolve problems, namely, to give a clear explanation of the problem and its solution in the relevant area of application.

The method and algorithm design pattern definition source code software that will allow to choose the most suitable design pattern in the early stages of software development will solve the problem of structuring source code.

Keywords: planning, source code, software.