З.С.Сейдаметова
Національний педагогічний університет ім. М.П.Драгоманова

Основна проблематика навчальної дисципліни «Технологія проектування»

Розробка програмних систем є непростим процесом. Програмний продукт з одного боку, повинен бути високоякісним, а з іншого, швидким і економічним по відношенню до ресурсів системи. Тому дуже важливо при підготовці фахівців в галузі комп'ютинга враховувати ці особливості, а також навчати студентів володінню сучасними методами і технологіями розробки програмного забезпечення.

Одним з важливих принципів розробки програмного забезпечення є «проектування, орієнтоване на зручність використання». Саме цей принцип береться за основу систематичного підходу до процесу проектування. Результатом застосування цього принципу є програмний продукт, відповідний реальним запитам клієнтів і користувачів. Цей підхід дозволяє за допомогою простих і дієвих моделей описувати роботу користувача з системою, дії, які користувач або система можуть здійснювати, а також передбачати інші підтримуючі дії системи. Одним з інструментів цього підходу є уніфікована мова моделювання UML, використовувана при об'єктно-орієнтованому програмуванні і проектуванні.

Ці питання розглядаються в рамках університетських дисциплін «Технологія проектування», «Програмна інженерія», які зазвичай включені в навчальні плани спеціальностей, пов'язаних з комп'ютингом.

Теоретичний базис розробки програмних систем складає програмна архітектура і моделі програмних систем. Є велика кількість літератури, присвяченої цій тематиці, але в основному вона англомовна. У даній статті приводиться короткий опис сучасних концепцій, моделей і підходів стосовно програмної архітектури, а також чотири найбільш часто використовувані представлення програмних систем. Ми сподіваємося, що представлений в даній статті матеріал буде корисний викладачам вищих навчальних закладів для формування теоретичних основ методики навчання технологій розробки програмних продуктів.

Процес розробки програмного забезпечення, крім навичок програмування і проектування, вимагає терпіння і старанності в роботі і обов'язкової уваги до всіх без виключення деталей проекту – цього терпіння і цієї уваги теж треба навчати студентів в рамках навчальної дисципліни «Технологія проектування» (а також у навчанні інших професійно-орієнтованих дисциплін).

Програмна архітектура. Основоположним моментом в розробці програмного забезпечення є програмна архітектура. Програмна архітектура є структурою і принципами взаємозв'язків компонентів програмної системи, розробкою і аналізом її властивостей. У програмну архітектуру вкладається задум розробників щодо структури і функціонуваня системи. Не дивлячись на розвиток програмної архітектури впродовж останнього десятиліття, до цих пір немає чіткого визначення поняття «Програмна архітектура». Наприклад, на сайті Software Engineering Institute (SEI) [1] наведено кілька означень цього поняття. У статті Дюейна Перрі і Олександра Вольфа [2], присвяченій основам вивчення програмної архітектури, запропонована (за аналогією з архітектурою будівель) формула «програмна архітектура = {елементи, форми, логічні обгрунтування/обмеження}». Для спеціалістів поняття «елементи» в цій формулі означають «компоненти і коннектори (з'єднувачі)».

Таким чином, програмна архітектура – це безліч архітектурних елементів, що мають певну форму.

Виділяють три різні класи архітектурних елементів:

- Оброблювальні елементи (processing elements);
- Елементи даних (data elements);
- З'єднуючі елементи (connecting elements).

Оброблювальні елементи – це компоненти, за допомогою яких забезпечується перетворення елементів даних; у свою чергу, елементи даних містять відомості, що використовуються і перетворюються; з'єднуюч елементи (час від часу вони можуть бути оброблювальними елементами або елементами даних) – це «клей», який скріплює різні частини архітектури. Виклики процедур, спільно використовувані дані, повідомлення – різні приклади з'єднуючих елементів, які «склеюють» архітектурні елементи.

У статті Д. Перрі і О. Вольфа [2] для ілюстрації класів архітектурних елементів наведений як метафора наступний приклад з ватерполо. Плавці – це оброблювальні елементи, м'яч – елемент даних, а вода – первинний з'єднуючий елемент («клей»). Якщо розглядати ватерполо, поло і футбол, то вони всі мають подібну «архітектуру», але відрізняються «клеєм», тобто вони мають подібні елементи, контури і форми, але відрізняються головним чином змістом гри та способами, якими елементи пов'язані один з одним.

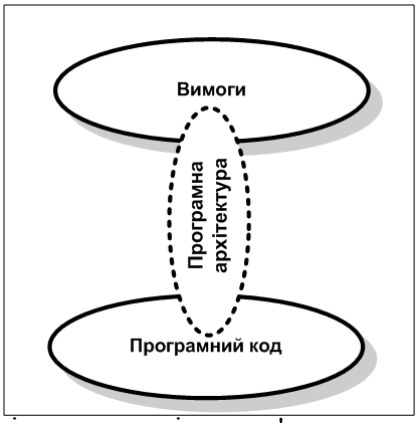

Рис. 1. Програмна архітектура як міст, що з'єднує вимоги і програмний код.

Девід Гарлан [3] представляє роль програмної архітектури як міст між вимогами до програмного забезпечення і програмним кодом (рис. 1).

У 1995 році компанія Rational [4] запропонувала модель програмної архітектури «4+1», яка частково була використана в стандарті Rational Unified Process. У цій моделі архітектура програмного забезпечення пов'язується з абстракцією, декомпозицією і композицією, із стилем і естетикою. Модель «4+1» складається з п'яти уявлень або перспектив (див. рис. 2):

Логічне представлення є моделлю об'єкта проекту (коли використовується метод об'єктно-орієнтованого проектування);

Представлення процесу включає як паралельне, так і послідовне виконання етапів; використовуються конструкції, що задають синхронізацію і розгалуження;

Фізичне представлення відображає картографію програмного забезпечення на апаратні засоби і відображає його розподілений аспект;

Представлення розвитку описує статичну організацію програмного забезпечення в навколишньому середовищі розвитку.

Опис архітектури, ухвалені рішення можуть бути організовані навколо цих чотирьох представлень, потім ілюстровані кількома вибраними випадками використання (use case) або *сценаріями*, які стають п'ятим представленням моделі «4+1».

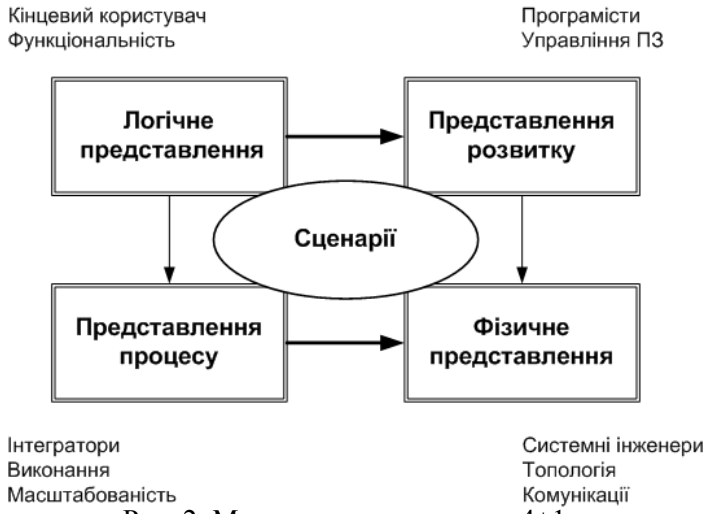

Рис. 2. Модель представлення «4+1».

У 1995 році також були сформульовані і чотири представлення програмної архітектури [5] компанії Siemens (див. рис. 3):

Концептуальна структура, що описує систему в термінах головних елементів проектування і зв'язків між ними;

Модульна структура, що охоплює дві ортогональні структури: функціональну декомпозицію і шари;

Виконавча структура, що описує динамічну структуру системи;

Структура програмного коду, що описує яким чином вихідний код, бінарність і бібліотеки організовані в середовищі розвитку.

Корпорації Siemens, Nokia, Phillips, Nortel, Lockheed Martin, IBM і інші крупні розробники програмного забезпечення звернули увагу на значення програмної архітектури для створення якісного софтверного продукту і внесли внесок до її розвитку.

У 1999 році група Open Group на базі мови XML створила мову Architecture Description Markup Language. В той же час виникли і сформувалися методи SAAM, BAPO, ATAM і стандарти RM-ODP и IEEE 1471 [6]. Сьогодні в крупних компаніях–розробниках програмного забезпечення є посади головних архітекторів, з'явилися мови опису архітектури ADL (напр., Koala, UML). Є і готова архітектура у вигляді платформ (напр., J2EE, .Net), мови сценаріїв (напр., Python, Perl).

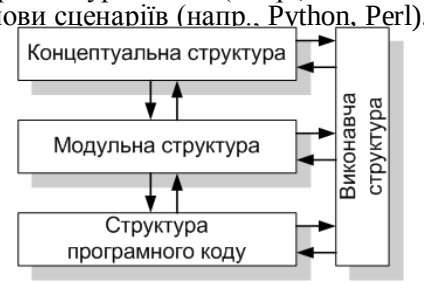

Рис. 3. Категорії структур програмного забезпечення і зв'язки між структурами.

Класифікація архітектури. Американські вчені Мері Шоу і Давид Гарлан [7] вивчили зразки проектування для різної архітектури і класифікували їх таким чином:

1. Архітектура потоків даних.
   Послідовні пакети.
   Канали і фільтри.
2. Незалежні компоненти.
   Паралельні взаємодіючі процеси.
   Клієнт-серверні системи.

Системи, керовані подіями.
3. Віртуальні машини.
Інтерпретатори.
Системи, засновані на правилах.
4. Репозіторні архітектури.
Бази даних.
Гіпертекстові системи.
Дошки оголошень.
5. Рівнева архітектура.

Всі ці архітектури описани в статті Мері Шоу і Поля Клемента [11]. Вони можуть бути використані для вирішення широкого спектру проблем, пов'язаних із створенням програмних систем.

Моделі програмних систем. У енциклопедії Wikipedia в розділі «Програмна інженерія» (http://en.wikipedia.org/wiki/Software_engineering) процес розробки програмного забезпечення представлений схематично карикатурою. Незважаючи навіть на іронічність малюнка, представленого в Wikipedia, можна зрозуміти, що процес створення програмних систем не є простою справою. При розробці програмних систем слід мати на увазі не тільки розв'язування поточних завдань, але і враховувати потреби клієнта, які можуть виникнути в процесі експлуатації. Заздалегідь спрогнозувати ці потреби складно, тому в системі, що розробляється, необхідно закласти можливість додавання нових модулів.

Існує багато моделей програмних систем, що є основою для розробки програмних продуктів. Ці моделі можуть бути налагоджені під конкретні завдання, як користувачем, так і розробником, також їх можна змінювати, оновлювати, додавати до них елементи різного призначення. Знання структурних моделей дозволяє при проектуванні програмного продукту передбачити розширення його функціональних можливостей і налагодження на предметну галузь користувача.

Враховуючи особливості структур існуючих програмних систем, їх можна умовно поділити на чотири моделі, як це зроблено в статті [9]:

- Класична модель;
- Подієво-орієнтована модель;
- Відкрита програмна система (існують наступні реалізації цієї моделі: програмовані управляючі алгоритми, взаємозамінні модулі; інтерфейс програмування);
- Змішана (гібридна) модель.

Основу *класичної моделі* складають три базові елементи (блок введення даних, блок опрацювання даних і блок виведення даних), які можуть функціонувати паралельно. Через блок введення даних приймаються дані від користувача через призначений для користувача інтерфейс або від зовнішнього джерела даних і готуються для подальшого опрацювання, після перевірки введених даних на коректність, вони передаються до блоку опрацювання. Блок опрацювання даних забезпечує виконання алгоритму, передбаченого програмою. При необхідності виконуються операції управління блоками введення і виведення даних для відображення отриманих результатів. Через блок виведення даних перетворить дані, отримані від блоків введення і опрацювання, у вигляді, зручному для перегляду користувачем, або готуються дані для коректного передавання до зовнішніх пристроїв.

Подієво-орієнтована модель найчастіше використовується в системах з графічним інтерфейсом користувача. Кожен елемент програмної системи розглядається як незалежний об'єкт, що інформує програму про зміну свого стану за допомогою події (кожна подія опрацьовується за деяким алгоритмом). Ця модель складається з трьох базових підсистем: інтерфейсу користувача, диспетчера подій і блоку реакції на події. У блоці реакції на події закладені алгоритми, що виконуються у відповідь на кожну з можливих подій. При цьому дотримується обов'язкова умова незалежності: кожен блок реакції на певну подію не залежить від блоків реакції на інші події.

Наприклад, в програмі, що реалізовує текстовий редактор, є дві залежні події «відкривання файлу» і «збереження файлу». Принцип незалежності блоків вимагає, щоб блок реакції на подію «збереження файлу» не перевіряв, чи був раніше відкритий файл, що зберігається. Забезпечення такої незалежності покладається на диспетчер подій, який забезпечує передавання управління до блоку реакції на події при настанні події. Подієво-орієнтована модель легко реалізується за допомогою візуальних середовищ розробки.

Відкрита програмна система є системою, в якій містяться засоби зміни її функціональних призначень самим користувачем, що не вимагає втручання у вихідний код. Відкрита програмна система складається з інтерфейсу користувача, блоку опрацювання даних, засобів конфігурації. Основу цієї моделі складає управляючий алгоритм. У відкритій програмній системі розробник надає засоби реалізації управляючого алгоритму клієнтові у вигляді набору сервісів, за допомогою яких розв'язується певне коло завдань з використанням того або іншого управляючого алгоритму.

Змішані (або гібридні) *модеіь* застосовуються у великих програмних системах, і в них в основному використовується подієво-орієнтований підхід. Як приклад можна розглянути систему MS Word, в основу якої покладена подієво-орієнтована модель. Підтримка словників і тезаурусів в MS Word організована на основі технології взаємозамінних модулів. Розширення функціональних призначень MS Word у вигляді підсистеми макросів, описаних мовою високого рівня Visual Basic, організоване на основі технології програмованих управляючих алгоритмів.

У даній статті представлена проблематика програмної архітектури як основи процесу розробки програмних продуктів, а також основні моделі, які часто використовуються як каркаси програмних систем. Ця проблематика є стрижнем для навчальної дисципліни «Технологія проектування», введення якої в навчальній план спеціальності «Інформатика» нам представляється необхідним.

Необхідно підкреслити, що засвоєння теоретичних положень розробки програмних систем вимагає великої кількості індивідуальної роботи студента, а також виконання колективних проектів. Ці питання розглянуті автором в навчальному посібнику [10] і статті [11].

Надалі планується детальніший опис програми навчальної дисципліни «Технологія проектування», практичних і лабораторних занять, самостійної і індивідуальної роботи, а також софтверних програм, яки використовуються як програмне забезпечення даної дисципліни.

ЛІТЕРАТУРА

1. Сайт Software Engineering Institute (SEI) – http://www.sei.cmu.edu/architecture/definitions.html
2. Perry D., Wolf A. Foundations for the Study of Software Architecture. // ACM Software Eng. Notes, 1992, vol. 17, no 4. – pp. 40–52.
3. Garlan D. Software Architecture: a Roadmap. // The Future of Software Engineering, A. Finkekstein (Ed), ACM Press, 2000. – 9 p.
4. Kruchten P. The 4+1 View Model of Architecture. // IEEE Software, 1995, vol. 12, no. 6. – pp. 42–50.
5. Soni D., Nord R., Hofmeister C. Software Architecture in Industrial Applications. // Proc. 17th Int'l Conf. Software Eng. (ICSE-17), ACM Press, 1995. – pp. 196–207.
6. IEEE 1471-2000. IEEE Recommended Practice for Architecture Description of Software-Intensive Systems. – NY: IEEE Press, 2000. – 29 p.
7. Garlan D., Shaw M. Software Architecture: Perspectives on an Emerging Discipline. – NJ: Prentice Hall, 1996. – 242 p.
8. Shaw M., Clements P. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. // Proc. COMPSAC97, 21st Int'l Computer Software and Applications Conference, August 1997. – pp. 6-13.
9. Цытович П.Л. Структурные модели программных систем. // Компьютеры+программы, № 11, 2003. – http://www.cpp.com.ua.
10. Сейдаметова З.С. Технология проектирования. – Сімферополь: Кримське навчально-педагогічне державне видавництво, 2006. – 76 с.
11. Сейдаметова З.С. Формирование профессиональных навыков в компьютерных науках: командный и проектный подходы. // Проблеми інженерно-педагогічної освіти. Зб. наук. праць – Харків: УІПА – 2006, № 13. – С. 22–27.