Лекція 2. Архітектура ПЗ. Технології програмування. компонентний підхід. Зразки проектування.

1) Архітектура ПЗ
2) Розробка і оцінка архітектури
3) UML. Види діаграм UML
4) Зразки проектування і архітектурні стилі

1. Архітектура ПЗ

Під архітектурою ПЗ зазвичай розуміють набір внутрішніх структур ПЗ, що складаються з компонентів, зв'язків і можливих взаємодій між ними, а також видимих ззовні властивостей цих компонентів.

Під компонентом в цьому визначенні мається на увазі досить довільний структурний елемент ПО, який можна виділити шляхом визначення інтерфейсу взаємодії між цим компонентом і всім, що його оточує. Термін «компонент» в розробці ПЗ найчастіше (далі, під час обговорення UML і технології EJB) має дещо інший, більш вузький зміст - це одиниця складання системи, її розгортання і конфігураційного управління, то, що не може бути розділене на більш дрібні елементи при розгортанні або постачання системи. Там, де можливі непорозуміння, буде вказано, в першому, широкому або в другому, вузькому сенсі вживається цей термін.

Архітектура ПЗ схожа на набір карт певної території - карти мають різні масштаби, на них показані різні елементи (адміністративно-політичний поділ, рельєф і тип місцевості - ліс, степ, пустеля, болотиста місцевість і ін., Економічна діяльність і зв'язку), але вони об'єднуються тим, що всі відомості, представлені на них, співвідносяться з географічним положенням.

Точно так же архітектура ПЗ являє собою набір структур або уявлень, що мають різні рівні абстракції (аналог масштабу географічних карт) і показують різні аспекти (структуру класів ПО, структуру розгортання, тобто прив'язки компонентів ПО до фізичних машинам, можливі сценарії взаємодій компонентів і пр.), що об'єднуються прив'язкою всіх представлених даних до структурних елементів ПЗ.

Архітектура важлива перш за все тому, що саме вона визначає більшість характеристик якості ПЗ в цілому. Архітектура служить також основним засобом спілкування між розробниками, а також і між усіма особами, зацікавленими в даному ПЗ.

Вибір архітектури визначає спосіб реалізації вимог на високому рівні абстракції. Саме архітектура майже повністю визначає такі характеристики ПЗ як надійність, переносимість і зручність супроводу. Архітектура значно впливає і на зручність використання і ефективність ПЗ, які визначаються також і реалізацією окремих компонентів. Значно менше вплив архітектури на функціональність - зазвичай для реалізації заданої функціональності можна використовувати різні архітектури.

Тому вибір між тією або іншою архітектурою визначається перш за все саме нефункціональними вимогами і необхідними властивостями ПЗ в аспектах зручності супроводу і переносимості. При цьому для побудови гарної архітектури

треба враховувати можливі протиріччя між вимогами до різних характеристиках і вміти вибирати компромісні рішення, що дають прийнятні значення за всіма показниками.

Так, для підвищення ефективності в загальному випадку вигідніше використовувати монолітні архітектури, в яких виділено невелике число компонентів (в межі - єдиний компонент) - цим забезпечується економія як пам'яті, оскільки кожен компонент зазвичай має свої дані, а тут число компонентів мінімально, так і часу роботи, оскільки можливість оптимізувати роботу алгоритмів обробки даних є також, зазвичай, тільки в рамках одного компонента.

З іншого боку, для підвищення зручності супроводу, навпаки, краще розбивати систему на велике число окремих компонентів, з тим, щоб кожен з них вирішував свою невелику, але чітко визначену частину спільного завдання. При цьому, якщо виникають зміни у вимогах або проект, їх зазвичай можна звести до зміни однієї-кількох таких підзадач, і, відповідно, змінювати тільки відповідають за вирішення цих підзадач компоненти.

З третього боку, для підвищення надійності краще використовувати дублювання функцій, тобто зробити кілька компонентів відповідальними за вирішення однієї підзадачі. Причому, оскільки помилки в ПЗ найчастіше носять невипадковий характер (тобто вони повторювані, на відміну від апаратного забезпечення, де помилки пов'язані перш за все з випадковими змінами характеристик середовища і можуть бути подолані простим дублюванням компонентів, без зміни їх внутрішньої реалізації), краще використовувати досить сильно розрізняються способи вирішення однієї і тієї ж задачі в різних компонентах.

Список стандартів, що регламентують опис архітектури та проектну документацію взагалі, виглядає так:

- IEEE 1016-1998 Recommended Practice for Software Design Descriptions
- IEEE 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems.

2. Розробка і оцінка архітектури

При проектуванні архітектури системи на основі вимог, зафіксованих у вигляді варіантів використання, перші можливі кроки полягають у наступному.

1) Вибирається набір «основних» сценаріїв використання - найбільш істотних і часто використовуваних
2) Визначаються, виходячи з досвіду проектувальників, обраного архітектурного стилю (див. Далі) і вимог до переносимості та гнучкості, компоненти відповідають за певні дії - рішення певних підзадач - в рамках цих сценаріїв:
   - Сценарії розбиваються на послідовності обміну повідомленнями між отриманим компонентами
   - При виникненні додаткових добре виділених підзадач, додаються нові компоненти, і сценарії уточнюються
   - Для кожного компонента в результаті виділяється його інтерфейс - набір повідомлень, які він приймає від інших компонентів і посилає їм

- Розглядаються «неосновні» сценарії, які так само розбиваються на послідовності обміну повідомленнями з використанням, по можливості, вже визначених інтерфейсів
- Якщо інтерфейси недостатні - вони розширюються
- Якщо інтерфейс компонента занадто великий або компонент відповідає за занадто багато - він розбивається на більш дрібні
- Там, де це необхідно в силу вимог ефективності або зручності супроводу, кілька компонентів можуть бути об'єднані в один
- Все це робиться до тих пір, поки не виконаються наступні умови
- Всі сценарії використання реалізуються у вигляді послідовностей обміну повідомленнями між компонентами в рамках їх інтерфейсів
- Набір компонентів достатній для забезпечення всієї необхідної функціональності, досить зручний для супроводу і з точки зору переносимості і не викликає помітних проблем з ефективністю
- Кожен компонент має невеликий, чітко визначене коло вирішуваних завдань і чітко визначений, збалансований за розміром інтерфейс

На основі можливих сценаріїв несанкціонованих змін до системи можливий також аналіз характеристик архітектури та оцінка її придатності або порівняльний аналіз декількох архітектур. Це так званий метод аналізу архітектури ПЗ (Software Architecture Analysis Method, SAAM). Основні його кроки наступні.

1. Визначити набір сценаріїв дій користувачів або зовнішніх систем, або сценаріїв використання деяких можливостей, які можуть вже бути в вістеме або бути новими. Сценарії повинні бути значущими для конкретних зацікавлених осіб, будь то користувач, розробник, відповідальний за супровід, представник контролюючої організації та ін. Чим повніше набір сценаріїв, тим вищою буде якість аналізу. Можна оцінити частоту появи, важливість сценаріїв.

2. Визначити архітектуру (або кілька порівнюваних архітектур). Це повинно бути зроблено в зрозумілою всім учасникам оцінки формі.

3. Класифікувати сценарії. Для кожного сценарію з набору повинно бути визначено, чи підтримується він даної архітектурою або потрібно вносити в неї зміни, щоб цей сценарій став виконаємо. Сценарій може підтримуватися, тобто його виконання не вимагає внесення змін ні в один з компонентів, або ж не підтримуватися, якщо його виконання вимагає змін в описі поведінки одного або несолько компонентів або змін в їх інтерфейсах. Підтримка сценарію означає, що особа, зацікавлена в його виконання оцінює ступінь підтримки як достатню, а необхідні при цьому дії як досить зручні.

4. Оцінити сценарії. Для кожного непідтримуваного сценарію треба визначити необхідні зміни в архітектурі - внесення нових компонентів, зміни в існуючих, зміни зв'язків і способів взаємодії. Якщо є можливість, варто оцінити трудомісткість внесення таких змін.

5. Виявити взаємодію сценаріїв. Визначити які компоненти потрібно змінювати для непідтримуваних сценаріїв, компоненти, які потрібно змінювати для підтримки декількох сценаріїв - такі сценарії називають взаємодіючими, оцінити ступінь смислової пов'язаності взаємодіючих сценаріїв.

Мала зв'язаність за змістом між взаємодіючими сценаріями означає, що компоненти, в яких вони взаємодіють, виконують слабо пов'язані між собою завдання та їх варто декомпозировать.

Компоненти, в яких взаємодіють багато сценаріїв також є можливими проблемними місцями.

6. Оцінити архітектуру в цілому (або порівняти кілька заданих архітектур). Для цього треба використовувати оцінки важливості сценаріїв і ступінь їх підтримки архітектурою.

3. UML. Види діаграм UML

Для уявлення архітектури (точніше різних входять до неї структур) зручно використовувати графічні мови. На даний момент найбільш опрацьованим і найбільш широко використовуваним з них є уніфікована мова моделювання (Unified Modeling Language, UML), хоча на високому рівні абстракції архітектуру системи зазвичай описують просто набором іменованих прямокутників, з'єднаних лініями і стрілками, що представляють можливі зв'язки.

Деякі такі мови закріплені у вигляді стандартів, наприклад

• IEEE 1320.1-1998 (R2004) Standard for Functional Modeling Language - Syntax and Semantics for IDEF0

Цей стандарт описує мову ієрархічних діаграм потоків даних.

• IEEE 1320.2-1998 (R2004) Standard for Conceptual Modeling Language - Syntax and Semantics for IDEF1X97 (IDEFobject)

Цей стандарт описує мову опису струтурірованних даних на основі понять сутності, зв'язку і атрибута сутності, використовуваний, зокрема, для моделювання баз даних.

UML пропонує використовувати для опису архітектури 8 видів діаграм. Не всяка діаграма на UML описує архітектуру - 9-й вид діаграм, діаграми варіантів використання, не відносяться до архітектурних уявленням, крім того, і інші види діаграм можна використовувати для опису внутрішньої структури компонентів або сценаріїв дій користувачів і інших елементів, до архітектури не відносяться .

• Статичні структури, що відображають постійно присутні в системі суті і зв'язку між ними, або сумарну інформацію про сутності і зв'язках, або сутності та зв'язки, що існують в якийсь момент часу

o Діаграми класів. Показують типи сутностей системи, атрибути типів (поля і операції) і можливі зв'язки між ними, а також відносини типів між собою по спадкоємства.

Найбільш часто використовуваний вид діаграм.

o Діаграми об'єктів. Показують об'єкти системи та їх зв'язку, в деякому конкретному стані або сумарно.

Використовуються рідко.

o Діаграми компонентів. Це компоненти у вузькому сенсі, компоненти «фізичного» уявлення системи - файли з вихідним кодом, динамічно подгражаемие бібліотеки, HTML-сторінки тощо. Вони визначають розбиття системи на набір сутностей, розглядаються як атомарні з точки зору її збірки і конфігураційного управління.

Використовуються рідко.

о Діаграми розгортання. Показують прив'язку (в певний момент часу або постійну) компонентів системи (у вузькому сенсі) до фізичних пристроїв - машинам, процесорам, принтерів, маршрутизаторів тощо.

Використовуються рідко.

• Динамічні структури, що описують відбуваються в системі процеси

о Діаграми діяльностей. Показують набір процесів-діяльностей і потоки даних, що передаються між ними, а також можливі їх синхронізації один з одним.

Використовуються досить часто.

о Діаграми сценаріїв. Показують можливі сценарії обміну повідомленнями / викликами в часі між різними компонентами системи (в широкому сенсі). Ці діаграми є підмножиною іншого графічного мови - мови діграмм послідовностей повідомлень (Message Sequence Charts, MSC).

Використовуються майже так само часто, як діаграми класів.

о Діаграми взаємодії. Показують ту ж інформацію, що і діаграми сценаріїв, але прив'язують обмін повідомленнями / викликами ні до часу, а до зв'язками між компонентами.

Використовуються рідко.

о Діаграми станів. Показують можливі стану окремих компонентів або системи в цілому, переходи між ними у відповідь на будь-які події і виконувані при цьому дії.

Використовуються досить часто.

Приклади діаграм UML для простої системи ведення банківських рахунків.

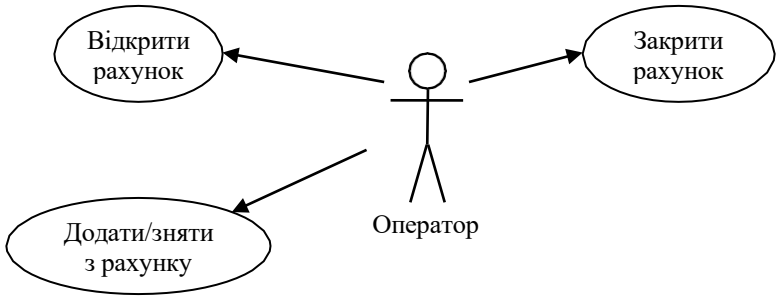

Рисунок 1. Діаграма варіантів використання (не архітектурна).

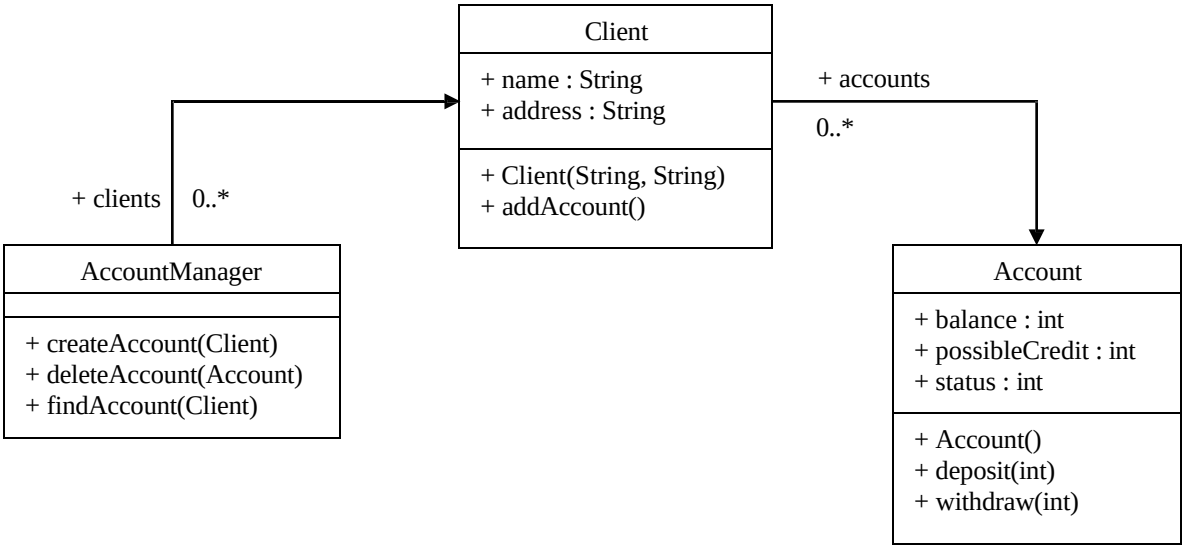

Рисунок 2. Діаграма класів.

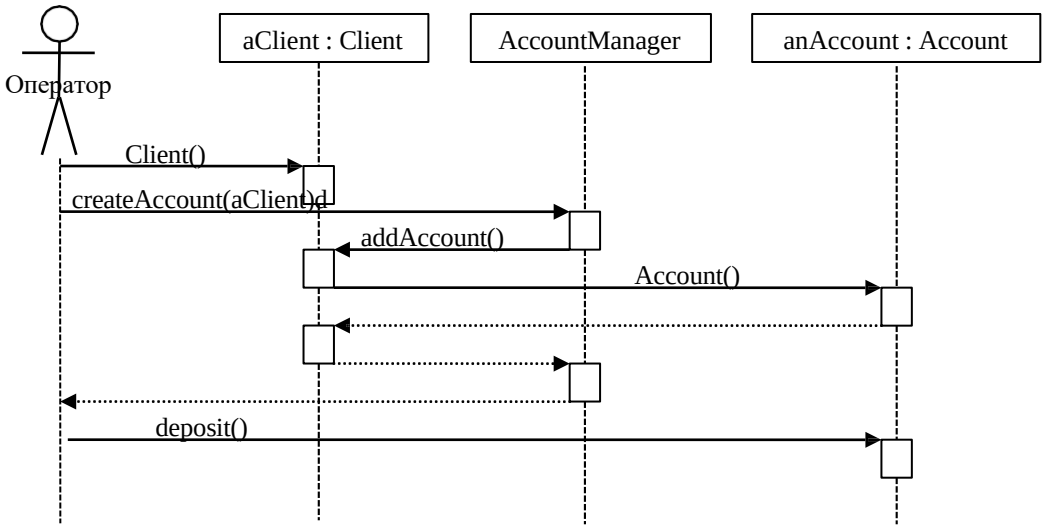

Рисунок 3. Діаграма сценарію відкриття рахунка.

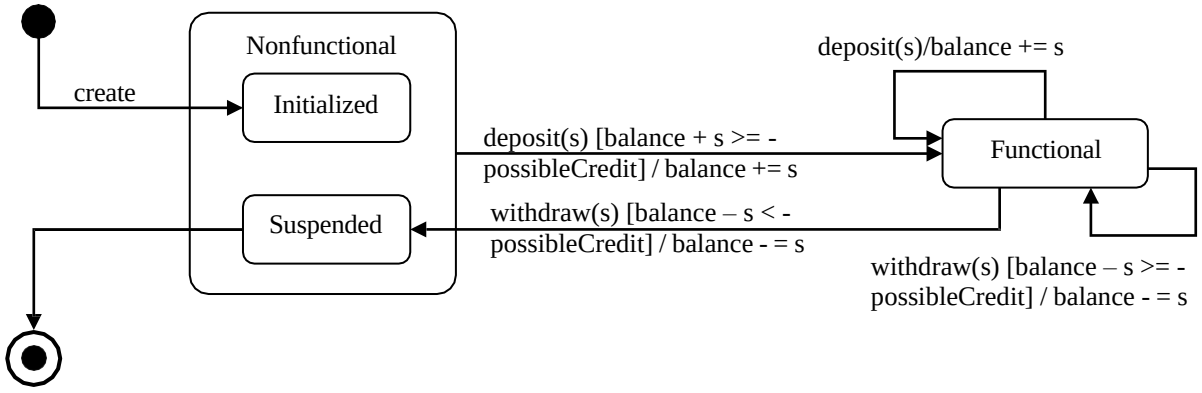

Рисунок 4. Діаграма станів рахунку.

4. Зразки проектування і архітектурні стилі

На основі наявного досвіду дослідниками і практиками розробки ПО вироблено деякий безліч типових архітектур, знайомство з якими дозволяє не винаходити велосипед для вирішення досить відомих задач. Подібні типові рішення на рівні архітектури називаються архітектурними стилями. Точніше, архітектурний стиль визначає набір типів компонентів системи і набір шаблонів їх взаємодій з передачі даних або управління. Різні архітектурні стилі підходять для вирішення різних завдань в плані забезпечення не функціональних вимог, хоча одну і ту ж функціональність можна реалізувати, використовуючи різні стилі.

Архітектурні стилі є зразками проектування на рівні архітектури. Зразок проектування (design pattern) - це шаблон рішення часто зустрічається завдання проектування, який можна використовувати всякий раз, коли ця задача виникає. Зразки проектування поділяються залежно від масштабу рішень на архітектурні, що визначають можливу декомпозицію системи в цілому або великих підсистем, області відповідальності підсистем і правила їх взаємодії, проектні, що визначають шаблон взаємодій групи компонентів, зазвичай в рамках деякої підсистеми, для вирішення деякої загальної задачі проектування в повторюється контексті, і ідіоми, що визначають спосіб використання мовних конструкцій для вирішення подібних завдань.

Стиль або зразок	***Контекст використання***	***Приклади***
Конвейер обробки даних (data flow)	**Система видає вихідні дані в результаті обробки добре певних вхідних, при цьому процес обробки не залежить від часу, застосовується багаторазово, однаково до будь-яких даних на вході.** **Важливою властивістю є чітко визначена структура даних і підтримка можливості інтеграції з іншими системами**	
• Пакетна обробка	Один висновок виробляється на основі читання деякого набору даних на вході, проміжні перетворення послідовні	Виконання тестів
• Канали і фільтри	Потрібно забезпечити перетворення безперервних потоків даних, перетворення Інкрементальний, наступне може бути розпочато до закінчення попереднього, можливо додавання додаткових перетворень	Утиліти UNIX, компілятори
• Замкнений цикл управління	Потрібно забезпечити постійне управління в умовах погано передбачуваних впливів оточення, особливо, якщо система повинна реагувати на зовнішні фізичні події	Системи управління рухом
«Виклик-повернення» (call-return)	**Порядок виконання дій досить визначено, компонентам нема чого витрачати час на очікування звернення від інших**	
• Процедурна декомпозиція	Дані незмінні, процедури роботи з ними можуть трохи змінюватися, можуть виникати нові	Основна схема побудови програм для мов

		C, Pascal, Ada
• Абстракні типи даних	Важливі можливості внесення змін та інтеграції з іншими системами, в системі багато даних, структура яких може змінюватися	Бібліотеки компонентів
• Багаторівнева система	Важливі переносимість і можливість багаторазового використання, є природне розшарування системи на специфічні тільки для неї функції та функції загального характеру, специфічні для платформи	Протоколи (модель OSI и реальні)
Незалежні компоненти	**Можливо розпаралелювання роботи і використання декількох машин, система природно розбивається на слабо пов'язані невеликі компоненти, робота яких може бути організована майже незалежно**	
• Клієнт-сервер	Завдання, які вирішуються природно розподіляються між ініціаторами і оброблювачами запитів, можлива зміна зовнішнього представлення даних і способів їх обробки	Основна модель бізнес-застосунків
• Розподілені об’єкти	Можливість використання розподіленої архітектури і численні дані з мінливою структурою	
Інтерактивні системи	**Необхідність досить швидко реагувати на дії користувача, мінливість призначеного для користувача інтерфейсу**	
• Дані-представлення-обробник	Зміни в зовнішньому поданні досить вірогідні, одна і та ж інформація може надаватися по-різному в декількох місцях, система повинна швидко реагувати на зміни даних	Document-View в MFC (Microsoft Foundation Classes)
• Представлення-абстракція-управління	Інтерактивна система на основі агентів, що мають власні стану і призначений для користувача інтерфейс, можливо додавання нових агентів	
Системы на основе хранилища данных	**Основні функції системи пов'язані зі зберіганням, обробкою і представленням великої кількості даних**	
• Репозиторій	Порядок роботи визначається потоком зовнішніх подій і цілком визначений	Середовища розробки і CASE-системи
• Дошка оголошень	Спосіб вирішення завдання в цілому невідомий або занадто трудомісткий, але відомі методи, частково вирішують завдання, композиція яких здатна видавати прийнятні результати, можливо додавання нових споживачів даних або обробників	Системи розпізнавання тексту