Лекція 1. Архітектура, архітектори, класифікація архітектурних парадигм

<u>Слайд 2.</u> **Зміст лекції**

1.1 Архітектура ПЗ основні визначення.
1.2 Роль архітектора.
1.3 Визначення якості архітектурного рішення.
1.4 Класифікація архітектурних парадигм

<u>Слайд 3.</u> Список стандартів, які регламінтіруют опис архітектури та проектної документації в загальному вигляді:

- IEEE 1016-1998 Recommended Practice for Software Design Descriptions;

- IEEE 1471-2000 Recommended Practice for Architectural Description of Software-Intensive Systems.

1.1 Архітектура ПЗ основні визначення

<u>Слайд 4.</u> Трохи історії. Основною вважається книга **Design Patterns: Elements of Reusable Object-Oriented Software** (Прийоми об'єктно-орієнтованого проектування. Паттерни проектування) - книга 1994 року про інженерію програмного забезпечення, що описує рішення деяких частих проблем в проектуванні програмного забезпечення. Автори книги: Еріх Гамма (Erich Gamma), Річард Хелм (англ.) Рос. (Richard Helm), Ральф Джонсон (англ.) Рос. (Ralph Johnson), Джон Вліссідс (англ.) Рос. (John Vlissides). Колектив авторів також відомий як «**Банда чотирьох**», Gang of Four [1], GoF. Автор передмови Граді Буч (Grady Booch). Книга складається з двох частин, в перших двох розділах розповідається про можливості і недоліки об'єктно-орієнтованого програмування, а в другій частині описані 23 класичних шаблону проектування. Приклади в книзі написані на мовах програмування C ++ і Smalltalk.

У стандарті IEEE +1471 дається наступне визначення: «Архітектура - це базова організація системи, яка описує зв'язки між компонентами цієї системи (і зовнішнім середовищем), а також визначає принципи її проектування і розвитку». Однак багато інших визначення архітектури визнають не тільки структурні елементи, але і їх композиції, а також інтерфейси і інші сполучні ланки.

RUP (Rational Unifying Process), взявши визначення IEEE, визначає архітектуру як "найбільш високорівневу концепцію системи в певному оточенні. Архітектура програмної системи (в конкретний момент часу) представляє собою організацію чи структуру важливих компонентів, що взаємодіють за допомогою

інтерфейсів; компоненти, в свою чергу, складаються з менших компонентів і інтерфейсів. "

Слайд 5. Ральф Джонсон про визначення архітектури: Я був редактором стандарту IEEE, в якому дано визначення, і я марно намагався довести, що воно абсолютно безглузде. Не існує такого поняття, як найбільш Високорівнева концепції системи. Користувачі і розробники дивляться на систему з різних точок зору. Користувачам абсолютно наплювати на структуру важливих компонентів. Так що, можливо, архітектура є найбільш високорівневою концепцією системи в певному оточенні з точки зору розробників. Давайте поки забудемо про розробників, які розуміють лише малу частину системи. Архітектура є найбільш високорівневою концепцією з точки зору розробників-експертів. А що робить компонент важливим? Він важливий, тому що розробники-експерти вважають його таким.

Так що більш відповідним буде наступне визначення архітектури: "в найбільш успішних проектах, розробники-експерти, що працюють над ним, мають загальним розумінням дизайну системи. Це загальне розуміння називається" архітектурою ". Це розуміння включає те, як система розділена на компоненти, і як ці компоненти взаємодіють один з одним за допомогою інтерфейсів. ці компоненти зазвичай складаються з більш дрібних компонентів, але архітектура включає в себе лише компоненти і інтерфейси, зрозумілі всім розробникам. "

Це визначення краще, оскільки дає зрозуміти, що архітектура є соціальною концепцією (ну, як і саме програмне забезпечення, але в архітектурі соціальна складова проявляється сильніше), оскільки вона стосується не всієї програми, а лише до тієї її частини, яку всі вважають важливою .

Існує ще одне визначення архітектури, яке звучить приблизно так: "архітектура - це набір проектних рішень (design decision), які повинні бути прийняті на ранніх етапах роботи над проектом". На що я можу заперечити, що якщо архітектура - це набір рішень, які ви б хотіли, щоб були прийняті вірно на початку проекту, то як ви можете гарантувати, що ймовірність правильності цих рішень вище, ніж у будь-яких інших рішень.

Архітектура відображає важливі речі, якими б вони не були.

Слайд 6. ***1.2 Роль архітектора***

Отже, якщо архітектура - це щось важливе, тоді архітектор - це людина (або люди), які турбуються про важливі речі. І ось тут ми стикаємося з принциповою відмінністю між видом архітектора, представленому в Матриці Перезавантаження і архітектором, прикладом якого є Дейв Райс.

Архітектус Релоадус (Architectus Reloadus) - це людина, що приймає всі важливі рішення. Він робить це, оскільки повинен бути єдиний розум, що забезпечує концептуальну цілісність системи, а може тому, що він не думає, що в команді є ще хтось з достатнім досвідом для прийняття цих рішень. До того ж, як правило, ці рішення повинні прийматися рано, щоб у всіх був план, з яким потрібно слідувати.

Архітектус Орізус (Architectus Oryzus) відноситься до іншого виду тварин (якщо не можете здогадатися, чому він так називається, то подивіться тут - www.nd.edu/~archives/latgramm.htm). Цей вид архітектора повинен бути максимально уважним до того, що відбувається на проекті, вишукуючи важливі проблеми і вирішуючи їх до того, як вони приведуть до серйозних неприємностей. Коли я бачу архітектора такого роду, то легко помітити, що найбільш суттєва частина його роботи полягає в тісному спілкуванні з іншими людьми. Вранці архітектор програмує з одним з розробників, намагаючись розібратися з настирливою проблемою блокувань. Після обіду він бере участь в мітингу зі збору вимог, пояснюючи бізнес-користувачам технічні наслідки їх ідей в нетехнічних формі, як, наприклад, за допомогою поняття вартості розробки.

Архітектор програмного забезпечення (ПЗ) - проектна роль в розробці ПЗ, професія, можливо - позиція / посаду.

Ключовий обов'язок архітектора - проектування архітектури ПЗ, тобто прийняття ключових проектних рішень щодо внутрішнього устрою програмної системи і її технічних інтерфейсів.

Слайд 7. У проектування архітектури ПЗ входять наступні завдання:

– визначення архітектурного шаблону / парадигми
– розбиття на технічні підсистеми / шари / компоненти / модулі
– визначення мовної парадигми для кожного з них
– вибір засобів виконання
– розробка ключових технічних сценаріїв взаємодії компонентів
– визначення протоколів взаємодії компонентів (проектування технічних інтерфейсів)
– визначення форматів зберігання і передачі даних
– підбір технічних засобів та шаблонів для реалізації підсистем.

Слайд 8. Крім того, до повноважень Архітектора ПЗ входить:

– рецензування вимог
– розробка функціональних вимог
– участь в нарадах з замовником
– стратегічне планування технічного розвитку системи
– реінжиніринг ПЗ
– архітектурний нагляд розробки
– поточне консультування команди
– технічний аудит сторонніх / знову придбаних систем
– регламентація процедури внесення змін
– розробка стандартів кодування / проектування
– написання технічного проекту.

Слайд 9. ***1.3 Визначення якості архітектурного рішення***

Одним з головних відмінностей архітектури будівель від програмної архітектури є те, що багато рішень, прийнятих при будівництві складно змінити. Дуже складно (хоча і можливо) взяти і змінити фундамент будівлі.

Але теоретичних причин, чому було б складно змінити щось в програмній системі, не існує. Якщо взяти один будь-який аспект програми, то можна зробити так, щоб його було легко змінити в майбутньому. Проблема в тому, що ми не знаємо, як зробити так, щоб легко було змінити будь-який аспект системи. Коли ми робимо деякий аспект системи простим в зміні, то вся система при цьому стає трохи складніше. Якщо ж ми зробимо будь-який аспект системи простим в зміні, то це призведе до неймовірного ускладнення всієї системи. Саме складність перешкоджає модифікації наших систем. Складність і дублювання.

Програмне забезпечення не обмежена законами фізики, як будівлі. Воно обмежене нашою уявою, дизайном і організацією. Коротше кажучи, воно обмежене властивостями людей, а не властивостями реального світу.

А як визначити якість архітектурного рішення? Щоб зрозуміти, як далеко деякий рішення проникло (або проникне) в різні частини системи, я задаю собі таке питання: «А що якщо я помилився і мені доведеться змінити це рішення в майбутньому? Які будуть наслідки? ». Якщо деякий рішення розмазане рівним шаром по всьому додатком, то вартість його зміни буде величезною.

В добре спроектованій системі ми повинні намагатися ізолювати свої помилки мінімальною кількістю модулів. Хороший дизайн дозволяє відкласти багато «важливі архітектурні рішення» на більш пізню стадію проекту, коли ймовірність їх вірного прийняття істотно вище.

Слайд 10. ***1.4 Класифікація архітектурних парадигм***

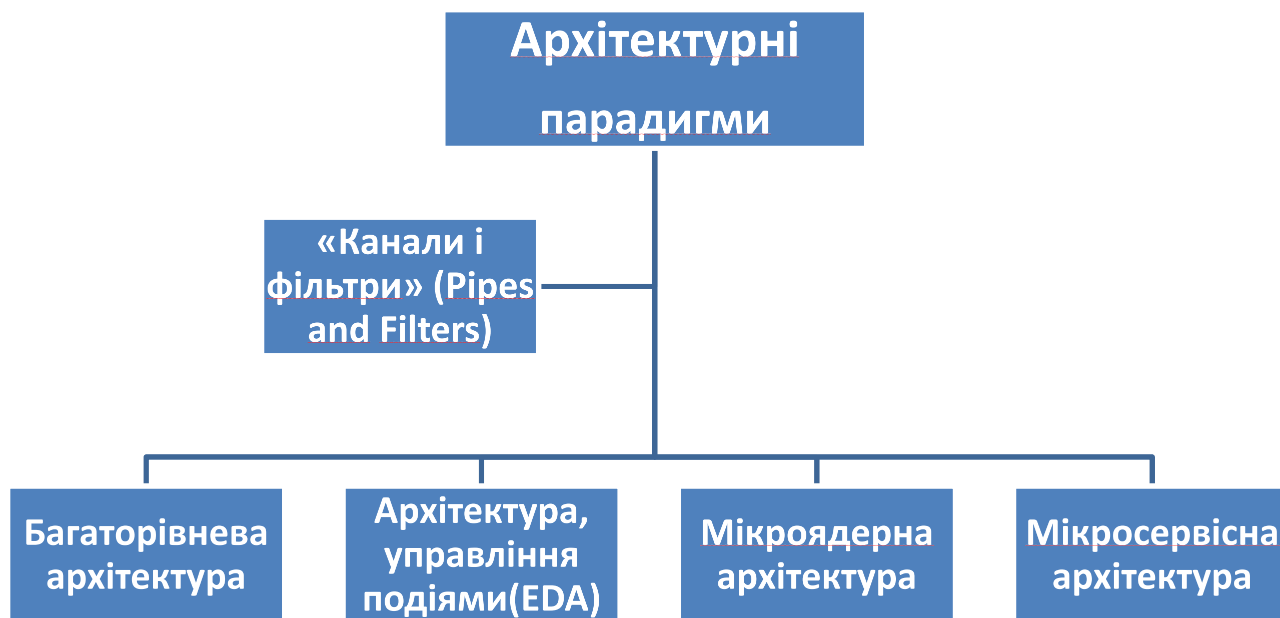

Рисунок 1 Класифікація архітектурних парадигм

Слайд 11.

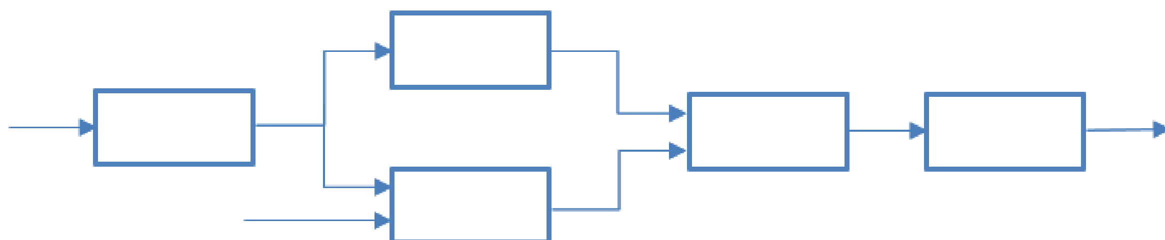

Рисунок 1.1 «Канали і фільтри» (Pipes and Filters)

Цей вид архітектури підходить в тому випадку, якщо процес роботи програми розпадається на кілька кроків, які можуть виконуватися окремими обработчиками. Основними компонентами є «фільтр» (filter) і «канал» (pipe). Іноді додатково виділяють «джерело даних» (data source) і «споживач даних» (data sink).

Кожен потік обробки даних - це серія чергуються фільтрів і каналів, що починається з джерела даних і закінчується їх споживачем. Канали забезпечують передачу даних і синхронізацію. Фільтр ж приймає на вхід дані і обробляє їх, трансформуючи в якесь інше уявлення, а потім передає далі.

Слайд 11. Наприклад, один з фільтрів може реалізовувати **шифр Цезаря** - шифр підстановки, в якому кожен символ в тексті замінюється символом, що знаходяться на деякій постійній числі позицій ліворуч або праворуч в алфавіті. Одна з варіацій коду Цезаря - це ROT13, що має крок рівний 13.

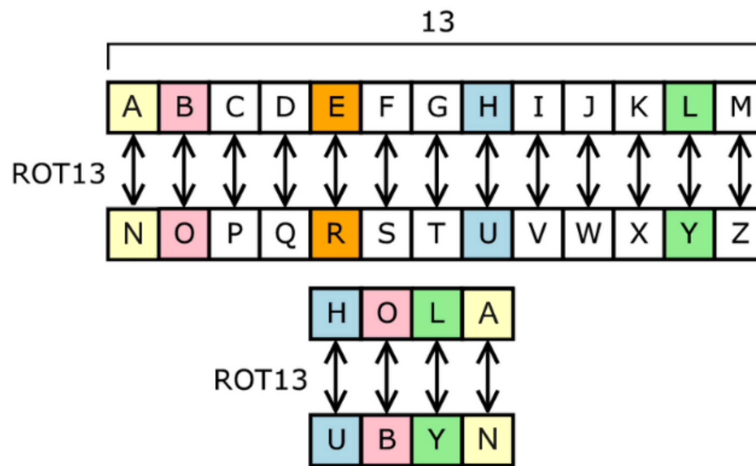

Рисунок 2 – Принцип ROT13

Слайд 13. Фільтри можна легко замінювати, використовувати повторно, переставляти місцями, що дає можливість реалізовувати безліч функцій на основі обмеженого набору компонентів. Більш того, активні фільтри можуть працювати паралельно, що призводить до значного підвищення продуктивності на багатопроцесорних системах. Однак є й недоліки, наприклад, фільтри часто витрачають більше часу на перетворення вхідних даних, ніж на їх обробку.

Як приклад використання цієї архітектури може служити оболонка UNIX Shell, одним з дизайнерів якої був Дуглас Макілрой (Douglas McIlroy). Іншим прикладом може стати архітектура компілятора, якщо розглядати її як послідовність фільтрів: Лексера, парсеру, семантичного аналізатора і генератора коду.

Слайд 14. Суть цієї парадигми декомпозиція задачі, яка виконує складну обробку на ряд дискретних елементів, які можуть бути використані повторно. Ця модель може поліпшити продуктивність, масштабованість і можливість багаторазового використання, дозволяючи використовувати елементи завдання, які виконують обробку для розгортання і масштабуються незалежно один від одного.

Слайд 15. Розглянемо невеликий приклад.

Проблема

Додаток може знадобитися для виконання різноманітних завдань різної складності базуючись на інформації, яку воно обробляє. Нескладний, але негнучкий підхід до реалізації цієї програми це виконання обробки в якості монолітного модуля. Проте, цей підхід, ймовірно, зменшить можливості для рефакторінга коду, його оптимізації, або повторного використання, якщо частини однієї і тієї ж обробки потрібні в іншому місці в межах додатку.

Слайд 16.

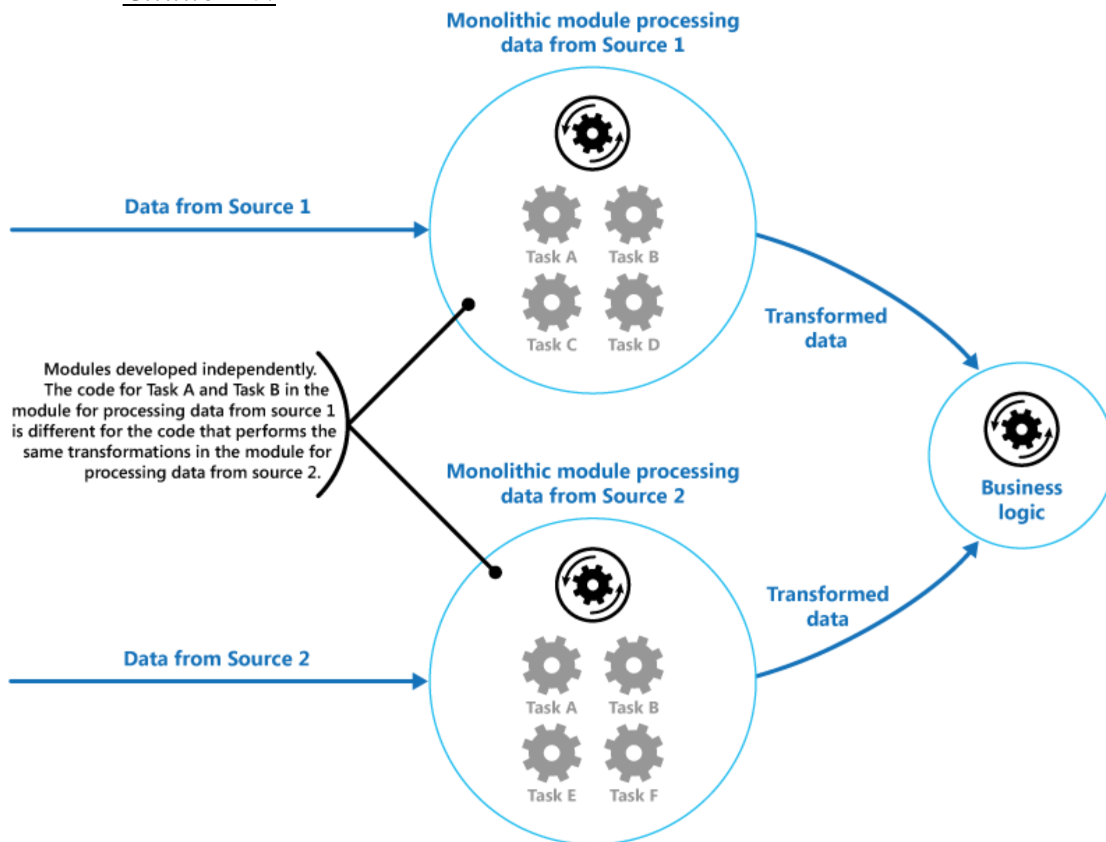

Рисунок 3 Монолітний підхід при проблемі з обробкою даних

На рис. 3 показані проблеми з обробкою даних, використовуючи монолітний підхід. Додаток отримує і обробляє дані з двох джерел. Дані від кожного джерела обробляються в окремому модулі, який виконує ряд завдань, щоб перетворити ці дані, перш ніж передати результат в бізнес-логіку програми.

Слайд 17. Рішення

Розкласти обробку даних, необхідну для кожного потоку в набір дискретних компонентів (або фільтрів), кожен з яких виконує одну задачу. Стандартизуючи формат вхідних та вихідних даних, ці фільтри можуть бути об'єднані разом в потік обробки даних. Це допомагає уникнути дублювання коду, і його можна видалити, замінити або інтегрувати додаткові компоненти, якщо вимоги до обробки змінюються. На рис.4 показаний приклад такої структури.

Слайд 18.

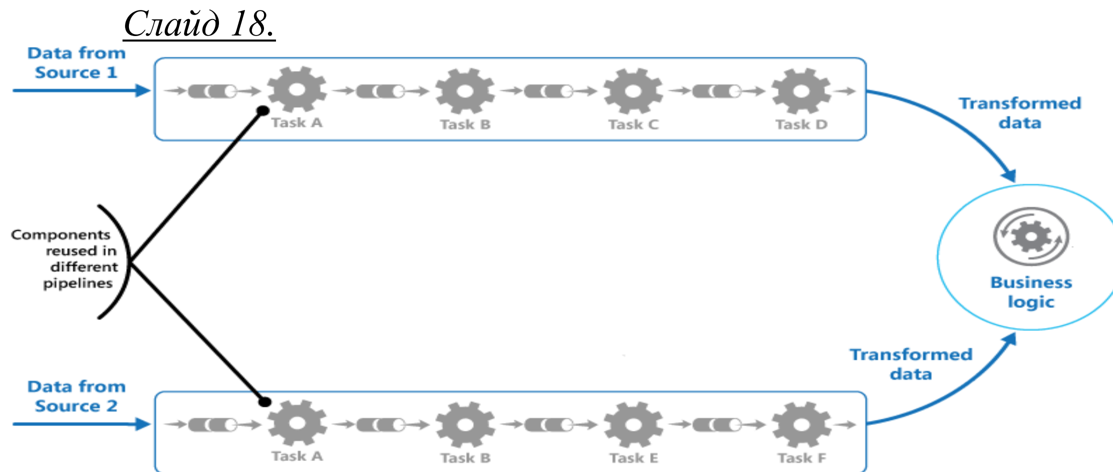

Рисунок 4 Інтеграція додаткових компонентів

Час, витрачений на обробку одного запиту залежить від швидкості самого повільного фільтра в потоці обробки даних. Цілком можливо, що один або кілька фільтрів може виявитися вузьким місцем, особливо, якщо велика кількість запитів з'являються в потоці з конкретного джерела даних. Ключовою перевагою конструкції потоку є те, що вона надає можливості для запуску паралельних примірників повільних фільтрів, що дозволяє системі розподіляти навантаження і поліпшити пропускну здатність.

Фільтри, які включаються в потік можуть працювати на різних машинах, що дозволяє їм бути масштабируются незалежно один від одного і можуть скористатися перевагами еластичності, що багато хмарні середовища забезпечують. Фільтр, який є обчислювально інтенсивним може працювати на високій продуктивності обладнання, в той час як інші, менш складні фільтри можуть бути розміщені на товарному (дешевше) апаратного забезпечення. Фільтри не повинні навіть бути в тому ж самому центрі обробки даних або географічне положення, що дозволяє кожному елементу в трубопроводі, щоб працювати в умовах, близьких до ресурсів він вимагає.

Слайд 19. **Плюси і мінуси даного підходу (шаблону, патерну):**

Ви повинні враховувати наступні моменти при ухваленні рішення про те, як реалізувати цей шаблон:

Складність. Підвищена гнучкість, що ця модель забезпечує також може ввести складності, особливо, якщо фільтри в потоці розподілені між різними серверами.

Надійність. Використовуйте інфраструктуру, яка забезпечує рух потоку даних між фільтрами і дані не будуть втрачені.

Ідемпотентність - властивість об'єкта або операції при повторному застосуванні операції до об'єкта давати той же результат, що і при одинарному. Тобто якщо фільтр не зміг обробити дані і вони обробляються дублюючим фільтром, але при цьому частина роботи завершена першим фільтром і дані вже опубліковані, таким чином дублюючий фільтр повторить ці дані.

Повторні повідомлення. Якщо фільтр зазнає невдачі після розміщення повідомлення на наступному ступені конвеєра, інший примірник фільтра може бути запущений (як описано при розгляді ідемпотентності вище), і він буде відправляти копію того ж повідомлення. Це може привести два примірники одного і того ж повідомлення, які повинні бути передані наступному фільтру. Щоб уникнути цього потік повинен виявити і уникнути дублювання повідомлень.

Запитання до Лекції 1

1. Сформулюйте основні визначення архітектури програмного забезпечення.
2. Опишіть та охарактеризуйте Список стандартів, які регламентують опис архітектури та проектної документації
3. В чому полягає роль архітектори?
4. Надайте визначення «Архітектура» за стандартом IEEE +1471.
5. Перерахуйте та охарактеризуйте завдання які входять до проектування архітектури ПЗ.
6. Яка роль архітектора та його ключові обов'язоки?
7. Опишіть та охарактеризуйте повноваження архітектора ПЗ.
8. Яким чином визначається якості архітектурного рішення?
9. Класифікація архітектурних парадигм. Замалюйте та опишіть схему даної класифікації.
10. Представте і охарактеризуйте структуру «Канали і фільтри» (Pipes and Filters).
11. Опишіть фільтр який може реалізовувати шифр Цезаря.
12. Надайте визначення Принципові ROT1.3
13. Охарактеризуйте монолітний підхід при проблемі з обробкою даних
14. Зобразіть графічно та опишіть інтеграцію додаткових компонентів з обробкою даних
15. від чого залежить час швидкості самого повільного фільтра в потоці обробки даних, витрачений на обробку одного запиту.
16. Які плюси і мінуси монолітного підходу (шаблону, патерну) Ви знаєте?
17. Які моменти Ви повинні враховувати при ухваленні рішення про те, як реалізувати шаблон? Дайте їх визначення.