МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕЛЕКОМУНІКАЦІЙ ТА ІНФОРМАТИЗАЦІЇ
(назва інституту (факультету))

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
(повна назва кафедри)

ПЛАН КОНСПЕКТ ЛЕКЦІЙ

з дисципліни «Моделювання та проектування програмного забезпечення»

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр та повна назва напряму (спеціальності))

Спеціалізації_____

Укладач(і): старший викладач Гаманюк І.М.
(науковий ступінь, вчене звання, П.І.Б. викладача)

Конспект лекцій розглянуто та схвалено
на засіданні кафедри інженерії програмного забезпечення
(повна назва кафедри).

Протокол № ____ від «___»_____20__р.

Завідувач кафедри В.В.ОНИЩЕНКО

Лекція № 18

Тема лекції:

Архітектура.

План лекції

Вступ.
1. Опис архітектури
2. Ознайомлення з представленнями архітектури
3. Логічна архітектура
4. Рівні
5. Програмна архітектура системи
6. Діаграми пакетів
7. Проектування на основі шаблона Layers
8. Рівні і розділи
9. Принцип Model-View Separation

Заключна частина.

Література

1. Г. Буч, Д. Рамбо, І. Якобсон Язык UML. Руководство пользователя 2-е издание / Пер. с агнл. – ДМК издательство, 2006 – 496 с.
2. К. Ларман. Применение UML 2.0 и шаблонов проектирования. Практическое руководство 3-е издание / Пер. с агнл. – Издательский дом “Вильямс”, 2013. – 736 с.
3. Р. Мартин, М. Мартин. Принципы, паттерны и методики гибкой разработки на языке C# / Пер. с агнл. – Издательство “Символ-Плюс”, 2014. – 768 с.
4. Руководство Microsoft по проектированию архитектуры приложений. 2е издание.
5. Обзор проектирования архитектуры // Режим доступу: https://msdn.microsoft.com/ru-ru/hh144976.aspx
6. Проектирование программного обеспечения // Режим доступу: https://ru.wikipedia.org/wiki/Проектирование_программного_обеспечения

Вступ

Процес проектування архітектури програмного забезпечення включає в себе збір вимог клієнтів, їх аналіз та створення проекту для компонента програмного забезпечення у відповідність до вимог. Успішна розробка ПЗ повинна забезпечувати баланс неминучих компромісів внаслідок суперечок вимог; відповідати принципам проектування та рекомендованим методам, вироблених з часом; і доповнювати сучасне обладнання, мережі і системи управління. Надійна архітектура програмного забезпечення вимагає значного досвіду в теоретичних і практичних питаннях, а також уяви, необхідного для перетворення бізнес-сценаріїв і вимог, які можуть здаватися неясними, в надійні і практичні робочі проекти.

1. Опис архітектури.

Архітектура програмного забезпечення включає в себе визначення структурованого рішення, відповідного всім технічним і робочим вимогам, одночасно оптимізуючи загальні атрибути якості, такі як продуктивність, безпеку і керованість. Сюди входить серія рішень, заснованих на широкому діапазоні факторів, і кожне з цих рішень може значно впливати на якість, продуктивність, підтримуваний і загальний успіх програмного забезпечення.

Сучасне програмне забезпечення рідко буває автономним. Як мінімум, в більшості випадків воно буде взаємодіяти з джерелом даних, наприклад корпоративною базою даних, що надає інформацію, з якою працюють користувачі програмного забезпечення. Зазвичай сучасне програмне забезпечення також має взаємодіяти з іншими службами і мережевими функція для виконання перевірки автентичності, отримання та публікації інформації та надання інтегрованих середовищ для роботи користувачів. Без відповідної архітектури може бути складно, якщо взагалі можливо, здійснити розгортання, експлуатацію, обслуговування і успішну інтеграцію з іншими системами; крім того, вимоги користувачів не будуть дотримані.

Архітектуру програмного забезпечення можна розглядати як зіставлення між метою компонента ПЗ і відомостями про реалізацію в коді. Правильне розуміння архітектури забезпечить оптимальний баланс вимог і результатів. Програмне забезпечення з добре продуманою архітектурою буде виконувати зазначені завдання з параметрами вихідних вимог, одночасно забезпечуючи максимально високу продуктивність, безпеку, надійність і багато інших чинників.

На найвищому рівні проект архітектури повинен надавати структуру системи, але приховувати деталі реалізації; охоплювати всі випадки застосування і сценарії; намагатися враховувати вимоги всіх зацікавлених осіб; і задовольняти настільки, наскільки можливо, всім функціональним вимогам і вимогам до якості.

2. Ознайомлення з представленнями архітектури.

Архітектура програмного забезпечення починається з набору вимог. Вони можуть бути виражені у формі діаграм, блок-схем процесу, моделей або документованих списків завдань експлуатації, які має виконувати програмне забезпечення. Зазвичай клієнт або партнер також висловлює менш точні вимоги, такі як зовнішній вигляд, або спосіб роботи певних інтерфейсів користувача для завдань,

що часто зустрічаются. Вимоги також повинні включати в себе інформацію про існуюче програмне забезпечення, системи, обладнання та мережі, з якими буде взаємодіяти нове програмне забезпечення; та інші фактори, такі як план розгортання і обслуговування, і, звичайно ж, доступний бюджет проекту.

Розробник архітектури програмного забезпечення повинен враховувати потреби клієнта. Однак загальний термін «клієнт» зазвичай складається з трьох суперечних областей відповідальності: бізнес-вимоги, вимоги користувача і системні вимоги. Бізнес-вимоги зазвичай визначають діапазон таких факторів, як бізнес-процеси, фактори продуктивності (такі як безпека, надійність і пропускна здатність), а також бюджетні обмеження і обмеження на витрати. Вимоги користувача включають в себе дизайн інтерфейсу, виробничі можливості і простоту використання програмного забезпечення. Системні вимоги: можливості та обмеження обладнання, мережі та середовища виконання. На рис. 1 показано, як можуть відрізнятися ці різні вимоги, тому розробнику необхідно створити архітектуру, яка підходить для областей, що перекриваються.

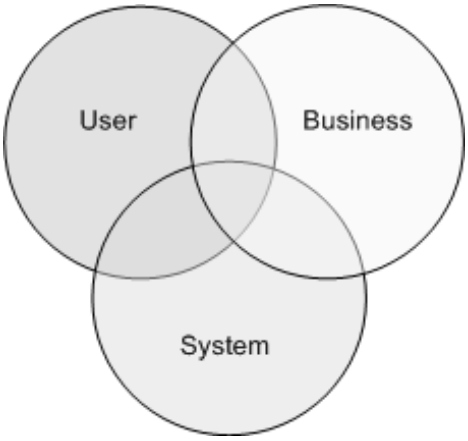


Рис. 1. - конфліктні вимоги типового клієнта

Кожен архітектор програмного забезпечення має власний підхід до збору й аналізу вимог, а також визначенню архітектури. Проте їм часто доводиться відповідати на такі питання: «Як користувачі будуть працювати з додатком?»; «Як додаток буде розгорнуто у виробничому середовищі і управлятися?»; «Які вимоги атрибутів якості для програми, такі як безпека, продуктивність, паралелізм, інтернаціоналізація і конфігурація?»; «Яка повинна бути архітектура програми для забезпечення гнучкості і зручності в обслуговуванні з плином часу?» І «Які архітектурні тенденції можуть впливати на додаток зараз і після його розгортання?».

Останнє питання одночасно важливе і цікаве. Хороша архітектура програмного забезпечення не тільки відповідає поточним вимогам клієнтів, але й буде відповідати таким вимогам в осяжному майбутньому Це впливає на рішення, що приймаються розробником архітектури щодо обладнання, компонентів, платформ, середовищ виконання, програмних систем управління і безлічі інших функцій, вбудованих в програмне забезпечення або з якими воно має інтегруватися.

Як і більшість завдань в світі проектування та розробки програмного забезпечення, розробка архітектури - це одночасно попереджувальний і ітеративний процес. Багато початкових завдань, такі як аналіз вимог, технічне дослідження і визначення цілей, зазвичай виконуються на початку процесу. Наступний етап -

визначення ключових сценаріїв для архітектури. Це основні вимоги, яким має відповідати програмне забезпечення, та обмеження, в рамках яких воно має працювати. На підставі цієї інформації розробник архітектури може створити огляд програми. Цей огляд охоплює такі високорівневі відомості, як тип програми (для Інтернету, телефонів, настільне або хмарне), архітектура розгортання (зазвичай багаторівнева архітектура, компоненти якої зв'язуються через кордони обладнання та мережі), відповідні застосовні стилі архітектури (наприклад, n-рівнева, клієнт сервер або сервіс-орієнтована) і технології реалізації, найкраще підходять для сценарію.

Після цього розробник може приступити до створення кандидатів архітектури, що задовольняють високорівневим і найбільш важливим вимогам, визначеним раніше. Після цього проект архітектури перевіряється і тестується в ключових сценаріях, часто в поєднанні з відгуками клієнтів і пробними або тестовими версіями, для забезпечення отримання оптимального результату. Це малоймовірно при першій ітерації, але при повторенні циклу буде досягнуто відповідність проекту вимогам і ключовим сценаріями. На рис. 2 показаний цей поетапний підхід.

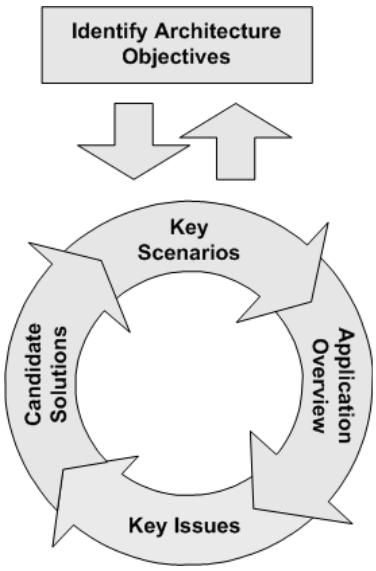


Рис. 2. - поетапний процес архітектурного проектування

У міру деталізації проекту і визначення окремих завдань і компонентів архітектор може уточнити і додати подробиці на кожному етапі. Наприклад, після визначення архітектурного стилю і підходу до розгортання архітектор може прийняти рішення про зв'язок рівнів і компонентів. Сюди може входити вибір протоколу на основі існуючих і майбутніх вимог, а також прийняття до уваги оглядів нових технологій і можливостей, визначених у майбутніх стандартах.

Остаточний продукт роботи архітектора - це зазвичай набір схем, моделей і документів, що визначають додаток з декількох точок зору, щоб при об'єднанні вони могли надати розробникам, групам тестування, адміністраторам і управлінню всю необхідну інформацію для реалізації проекту. Ця інформація буде описувати структуру і розміщення компонентів і рівнів додатків; спосіб обробки горизонтального перетину ієрархії, наприклад, протоколювання та перевірки; плани тестування і розгортання; та документацію для розробників, адміністраторів і персоналу служби підтримки.

В остаточному проекті також можуть бути вказані атрибути якості, яким має відповідати додаток. Це результат прийнятих рішень і компромісів розробника архітектури за консультаціями з клієнтом. До них відносяться визначення вимог безпеки та план реалізації безпеки, необхідна масштабованість і продуктивність при розгортанні на цільовій платформі, способи реалізації можливості обслуговування і розширюваності і функції забезпечення взаємодії з іншими системами.

Архітектурні представлення

1. Логічне представлення.

Відображає концептуальну структуру програмної системи в термінах рівнів, підсистем, пакетів, каркасів, найбільш важливих класів і інтерфейсів. В ньому узагальнюється функціональність основних програмних елементів (підсистем).

Відображає реалізацію важливих прецедентів, які ілюструють основні аспекти роботи системи.

Це представлення моделі проектування UP, побудоване з використанням позначень пакетів UML, класів і діаграм взаємодії.

2. Представлення процесів.

Відображає процеси і потоки, їх обов'язки, взаємодію і розміщення логічних елементів (рівнів, класів, підсистем і т.і.).

Це представлення моделі проектування UP, побудоване з використанням позначень пакетів UML, класів, діаграм взаємодії, а також процесів і потоків.

3. Представлення розгортання.

Відображає фізичний розподіл процесів і компонентів по процесорам і фізичну конфігурацію мережі.

Це представлення моделі розгортання UP, побудоване з використанням позначень діаграм розгортання UML. В цьому представлені зазвичай відображається вся система, а не її підмножина.

4. Представлення даних.

Відображає схему побудови бази даних і перетворення об'єктного представлення даних в реляціонне, описує збережені процедури і тригери бази даних.

Це представлення моделі даних UP, побудоване з використанням позначень діаграм класів UML.

Потоки даних можуть відображатися на діаграмах видів діяльності UML

5. Представлення безпеки.

Містить короткий опис схем забезпечення безпеки.

Може бути представлено у вигляді моделі розгортання UP, забезпеченою діаграмами розгортання, на яких виділено ключові моменти забезпечення безпеки.

6. Представлення реалізації.

"Модель реалізації" представляє вихідний код і код системи, що виконується.

В представлення реалізації включають короткий опис організації вихідного і виконуваного коду системи.

Це представлення моделі реалізації UP, яке складається із текстового опису і діаграм пакетів і компонентів UML.

7. Представлення розробки.

Містить необхідну інформацію про середовище розробки. Наприклад, в цьому представленні описується організація файлів в каталозі, засіб контролю версій і т.і.

8. Представлення прецедентів.

Короткий опис найбільш архітектурно важливих прецедентів і їх нефункціональних вимог. Реалізація таких прецедентів торкається багатьох архітектурно значимих елементів системи.

Це представлення моделі прецедентів, побудоване з використанням позначень діаграм прецедентів UML.

Кожне представлення включає не тільки діаграми, але і пояснюючий текст.

Іноді при складанні документа SAD забувають описати мотиви тих чи інших рішень. Опис мотивування часто є більш важливим, ніж інші розділи документу, коли виникає питання про внесення змін до архітектури.

Відповідно до Г. Буч, Д. Рамбо, І. Якобсон “Язык UML”:

Для моделювання архітектури необхідно:

- Ідентифікувати представлення, які ви будете використовувати для моделювання архітектури, а саме представлення:

варіантів використання (К.Ларман “Представлення прецедентів”);

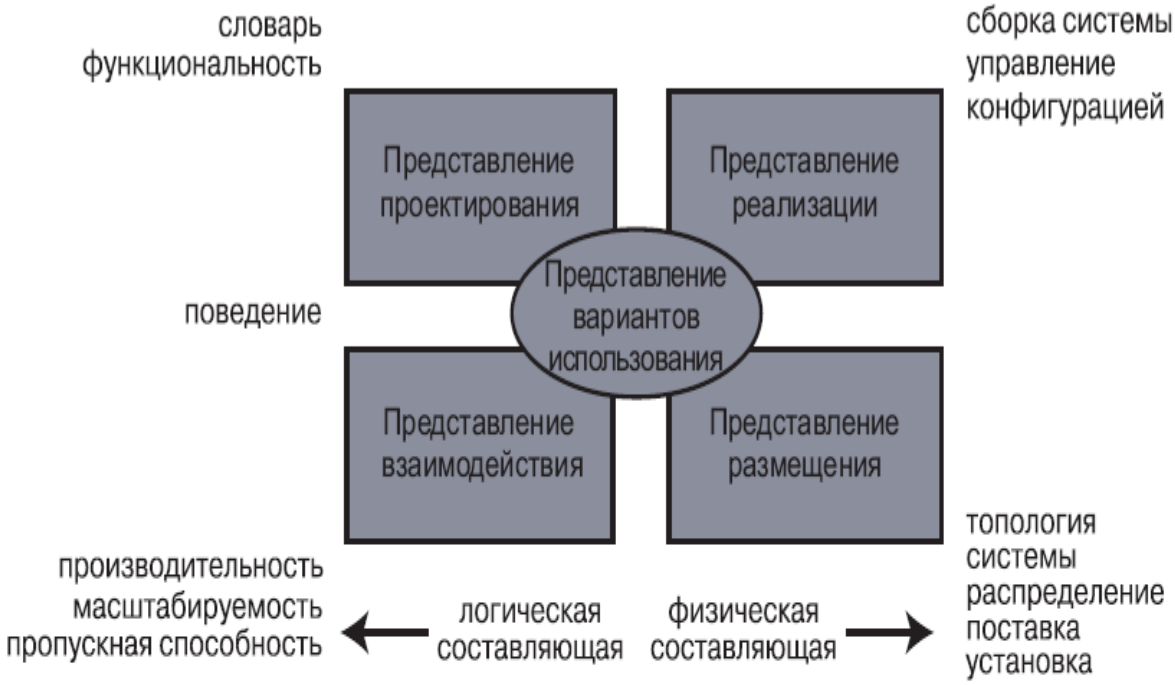
проектування (К.Ларман “Представлення процесів, даних”);

взаємодії (К.Ларман “Представлення процесів”);

реалізації (К.Ларман “Представлення реалізації”);

розміщення (К.Ларман “Представлення розгортання”).

- Специфікувати контекст системи, який включає діючих осіб.



Представлення варіантів використання:

описує поведінку системи, яка представляється кінцевим користувачам, аналітикам, тестувальникам.

Для моделювання статичних аспектів використовується діаграма варіантів використання, а для моделювання динамічних – діаграми взаємодії, станів та діяльності.

Представлення реалізації:

Має бути представлено інформацію щодо артефактів, які використовуються для збірки та випуску фізичної системи.

Для моделювання статичних аспектів використовуються діаграми артефактів.

Для моделювання динамічних аспектів – діаграми взаємодії, станів та діяльності.

Представлення розміщення:

Має бути представлено інформацію щодо вузлів, які формують топологію апаратних засобів, на яких працює система.

Для моделювання статичних аспектів використовуються діаграми розміщення.

Для моделювання динамічних аспектів – діаграми взаємодії, станів та діяльності.

3. Логічна архітектура

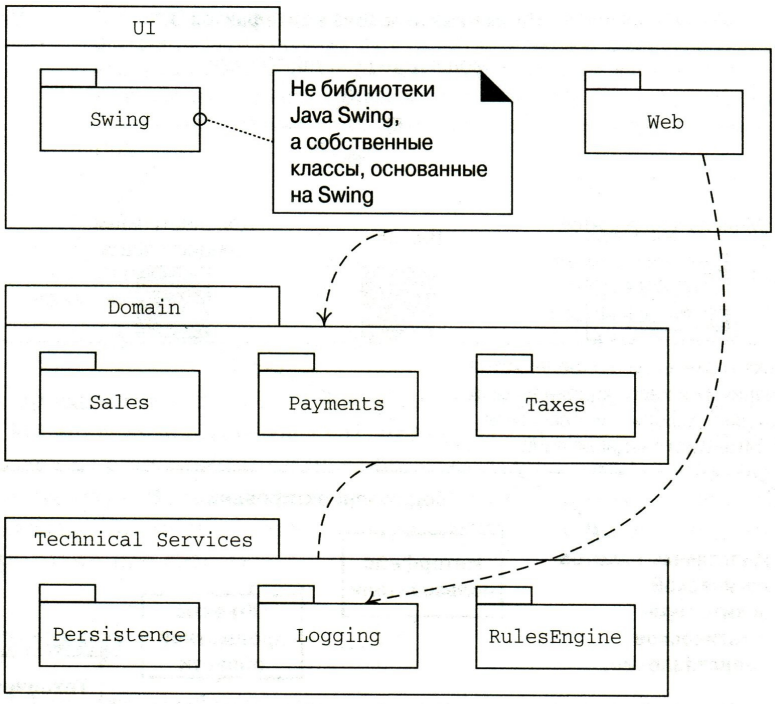
Логічна архітектура описує систему в термінах її принципової організації у вигляді рівнів, пакетів (чи просторів імен), програмних класів і підсистем. Вона називається логічною, оскільки не визначає способи розгортання цих елементів в різних операційних системах чи на фізичних комп'ютерах в мережі (це відноситься до архітектури розгортання).

Типічні рівні ОО системи:

інтерфейс користувача;

рівень застосування чи об'єктів предметної галузі;

рівень технічних служб. Об'єкти і підсистеми, які забезпечують підтримку взаємодії з базою даних чи журналами реєстрації помилок. Ці служби зазвичай незалежні від застосування, і їх можна повторно використовувати в декількох системах.



4. Рівні

В жорстко структурованої багаторівневої архітектурі об'єкти кожного рівня можуть викликати служби тільки рівня, який знаходиться безпосередньо під ними.

В інформаційних системах зазвичай використовується гнучка багаторівнева архітектура, при якій об'єкти можуть звертатися до декількох рівнів, які розташовані нижче.

Наприклад: об’єкти рівня інтерфейсу користувача можуть звертатися до елементів нижнього рівня технічних служб.

5. Програмна архітектура системи.

Архітектура – це набір важливих рішень, які стосуються організації програмної системи, вибору структурних елементів і їх інтерфейсів, які зачіпають поведінку і взаємодію цих елементів, їх групування до більш крупних підсистем і архітектурний стиль застосування.

Незалежно від визначення, архітектура системи включає крупномасштабні елементи – основні ідеї організації, шаблони, способ розподілення обов’язків, взаємодію і мотивацію поведінки системи і її основних підсистем.

6. Діаграми пакетів.

Діаграми пакетів UML використовуються для ілюстрації логічної архітектури застосування – рівней, підсистем, пакетів.

Кожний рівень можна представити у вигляді пакету.

Діаграма пакетів забезпечує один із способів групування елементів.

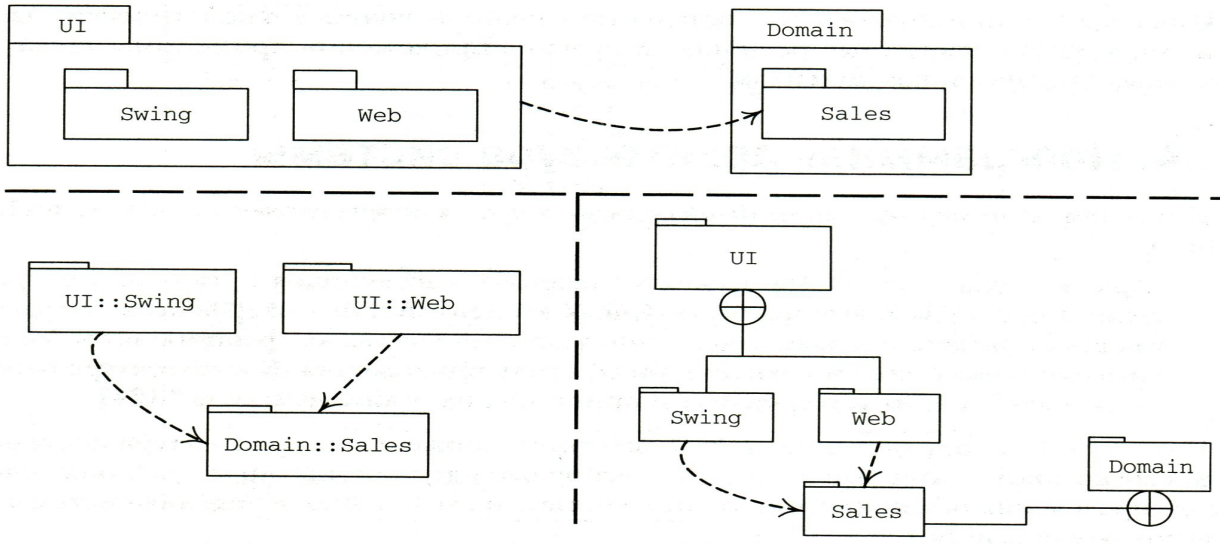
В одному пакеті UML можуть об’єднуватися різні елементи: класи, інші пакети, прецеденти і т.і.

Ім’я пакета розташовується на корінці, якщо на діаграмі відображається внутрішній вміст пакету, чи на самому позначені пакету.

Часто зображають залежності між пакетами, щоб розробникам був зрозумілим взаємозв’язок елементів системи. Для цього використовується лінія залежності UML, яка представляє собою пунктирну лінію зі стрілкою, яку направлено в бік незалежного пакету.

Пакет UML представляє собою простір імент, отдже, іноді використовують повну кваліфікацію імен для опису вкладеності пакетів.

Іноді вкладені пакети графічно розміщуються всередині загального пакету.



7. Проектування на основі шаблона Layers.

Основні принципи шаблону Layers:

Організувати крупномасштабні структурні елементи системи в окремі рівні із взаємозв'язаними обов'язками таким чином, щоб на нижньому рівні розташовувались низькорівневі служби і служби загального призначення, а на більш високих рівнях – об'єкти рівня логіки застосування.

Взаємодія і зв'язування рівнів відбувається зверху до низу. Необхідно уникати зв'язування об'єктів знизу вгору.

Використання шаблону Layers дозволяє вирішити проблеми:

Зміна вихідного коду тягне за собою коректування всіх елементів системи, оскільки всі частини системи тісно пов'язані одна з іншою.

Логіка застосування переплітається з інтерфейсом користувача, тому в застосуванні неможливо змінити інтерфейс чи принципи реалізації логіки застосування.

Загальні технічні служби тісно зв'язані з бізнес логікою застосування, тому їх не можна використовувати повторно, розповсюдити на інші системи чи змінити їх реалізацію.

Із-за високого ступеню зв'язування складно модифікувати функції застосування, масшабувати систему чи переходити на нові технології.

Переваги використання шаблону Layers:

У цілому, цей шаблон забезпечує поділ різних аспектів, високорівневих служб від низькорівневих, спеціалізованих функцій до загальних. Це знижує рівень зв'язування і залежності в застосуванні, підвищує ступінь зачеплення, збільшує потенціал повторного використання і вносить додаткову ясність.

Складні елементи піддаються декомпозиції і підлягають інкапсуляції.

Деякі рівні замінюються новими реалізаціями. В загальному випадку це не можливо для низькорівневих технічних служб і базового рівня, проте цілком реально для рівней інтерфейсу, застосування і предметної галузі.

Нижні рівні містять функції, які використовуються повторно.

Деякі рівні (особливо рівні предметної галузі і технічних служб) можуть бути розподіленими.

Логічна сегментація забезпечує можливість роботи над застосуванням групи розробників.

8. Рівні і розділи.

Архітектурні рівні представляють ділення системи по вертикалі, а розділи – по горизонталі на паралельні підсистеми в рамках одного рівня.

Наприклад, рівень служб можна розділити на розділи, які відповідають за виконання вимог безпеки і формування звітів.

Більшість систем спирається на зовнішні ресурси чи служби, такі як бази даних.

В логічному представлені архітектури можна використовувати не тільки рівні, але і фізичні реалізації компонентів.

В термінах логічного представлення архітектури доступ до конкретної бази даних відображається як пакет рівня предметної галузі. При цьому для доступу до бази даних можна використовувати пакет Persistence (Сховище) розділу технічних служб.

9. Принцип Model-View Separation.

Цей принцип можна сформулювати так:

Не зв'язуйте об'єкти інтерфейсу користувача з об'єктами інших рівнів напряму.

Не використовуйте імена об'єктів логіки застосування в методах об'єктів інтерфейсу користувача. Об'єкти інтерфейсу користувача повинні лише ініціалізувати відповідні елементи інтерфейсу, отримати повідомлення про події (наприклад: про переміщення миші чи натисканні на кнопку) і делегувати запити об'єктам рівня логіки застосування.

Використання принципу MVS надає можливості:

Підтримує високий рівень зчеплення спеціальних об'єктів і дозволяє сконцентрувати увагу на процесах предметної галузі, а не на питаннях інтерфейсу.

Дозволяє розділити процес розробки рівнів моделі і інтерфейсу користувача.

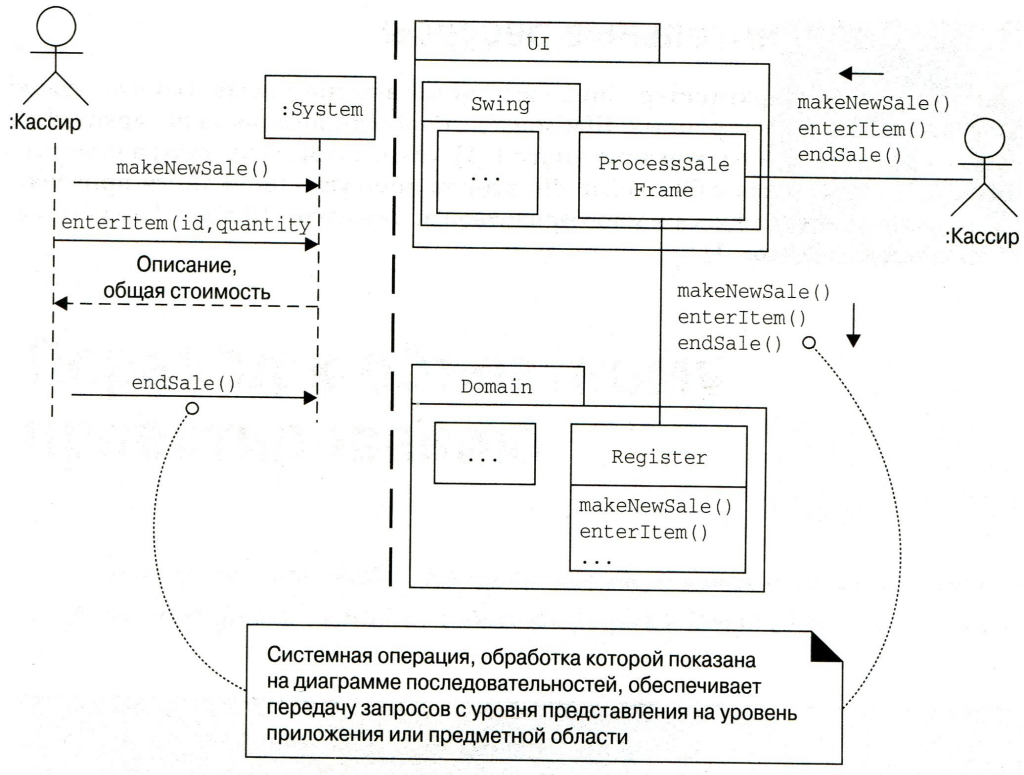
Мінімізує вплив змін інтерфейсу на спеціальні об'єкти.

Дозволяє легко підключити новий інтерфейс, не торкаючись рівня реалізації.

Дозволяє одночасно використовувати декілька різних представлень для одного і того ж спеціального об'єкта (вивести інформацію у вигляді таблиці і діаграми).

Забезпечує виконання задач рівня реалізації незалежно від інтерфейсу користувача (обробка повідомлень).

Зпрощує перехід до іншого каркасу інтерфейсу користувача.



Заключна частина

Вимоги до сучасного програмного забезпечення стають все більш складними, оскільки користувачі очікують все більше від додатків. Можливостей простих автономних настільних додатків більше не достатньо для більшості комерційних і ділових ситуацій. У світі розвиненою зв'язку додатки повинні взаємодіяти з іншими додатками і службами, а також працювати в різних середовищах, наприклад у хмарі, і на портативних пристроях. На зміну поширеним у минулому монолітних архітектур прийшло компонентне сервіс-орієнтоване програмне забезпечення, що використовує платформи, операційні системи, хост-додатки та мережі для реалізації функцій, про яких було не відомо всього кілька років тому.

Наочні посібники

Комп’ютер, мультимедійний проектор.

Завдання на самостійну роботу

1. Відпрацювати логічну архітектуру уявного застосування.

Старший викладач

І.М.ГАМАНЮК

(ініціали, прізвище)