Методика побудови архітектури і дизайну

Зміст

Вступ

Для опрацювання та створення прототипу майбутньої архітектури існує ітеративна техніка. Вона допоможе звести до єдиного цілого ключові рішення: по параметрам якості, по архітектурним стилям, типам застосувань, технологіям і сценаріям розгортання.

Ітеративний процес допомагає відпрацювати можливі варіанти рішень, які в подальшому відпрацьовуються в ході ітерацій і, забезпечують створення дизайну архітектури, який найбільше відповідає застосуванню, що розробляється.

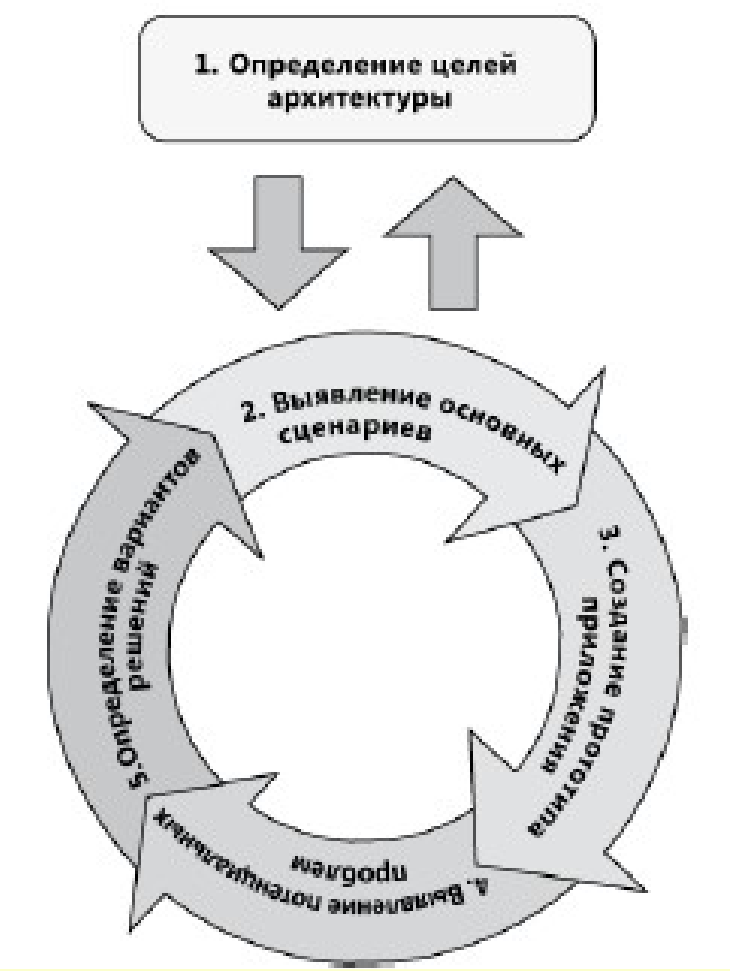
Архітектура може багаторазово переглядатися в ході життєвого циклу. Ця методика підходить до подальшої доробки архітектури.

Вхідні та вихідні данні та етапи проектування

Вхідні дані проектування допомагають формалізувати вимоги і обмеження, які має реалізувати створювана архітектура. Зазвичай вхідними даними є варіанти використання і сценарії поведінки користувача, функціональні вимоги, нефункціональні вимоги (включаючи параметри якості, такі як продуктивність, безпека, надійність і інші), технологічні вимоги, цільове середовище розгортання та інші обмеження.

В ході процесу розробки створюється список значущих з точки зору архітектури варіантів використання, аспектів архітектури, які потребують спеціальної уваги, і можливих архітектурних рішень, які задовольняють вимогам і обмеженням, виявленим в процесі проектування. Загальною технікою поступового доопрацювання дизайну до тих пір, поки він не буде задовольняти всім вимогам і обмеженням, є ітеративна методика, що включає п'ять основних етапів.

Вхідні та вихідні данні та етапи проектування

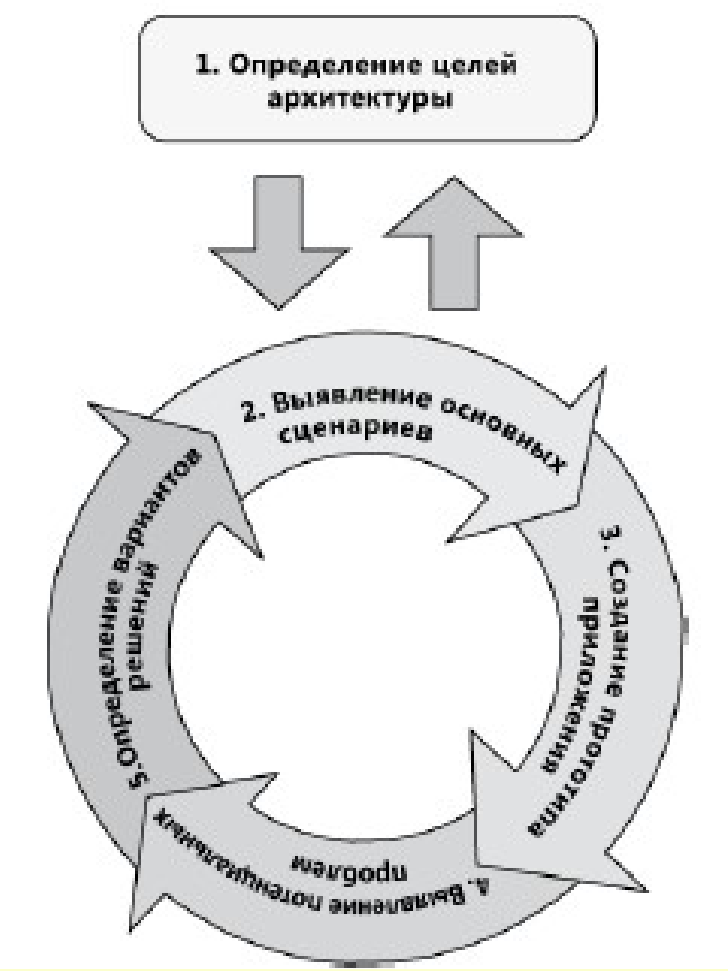


Основні етапи ітеративної методики для доопрацьовування дизайну архітектури:

1. **Визначення цілей архітектури.**
Наявність чітких цілей допоможе зосередитися на архітектурі і правильному виборі проблем для вирішення. Точно зазначені цілі допомагають визначити межі кожної фази: момент, коли завершена поточна фаза і все готово для переходу до наступної.

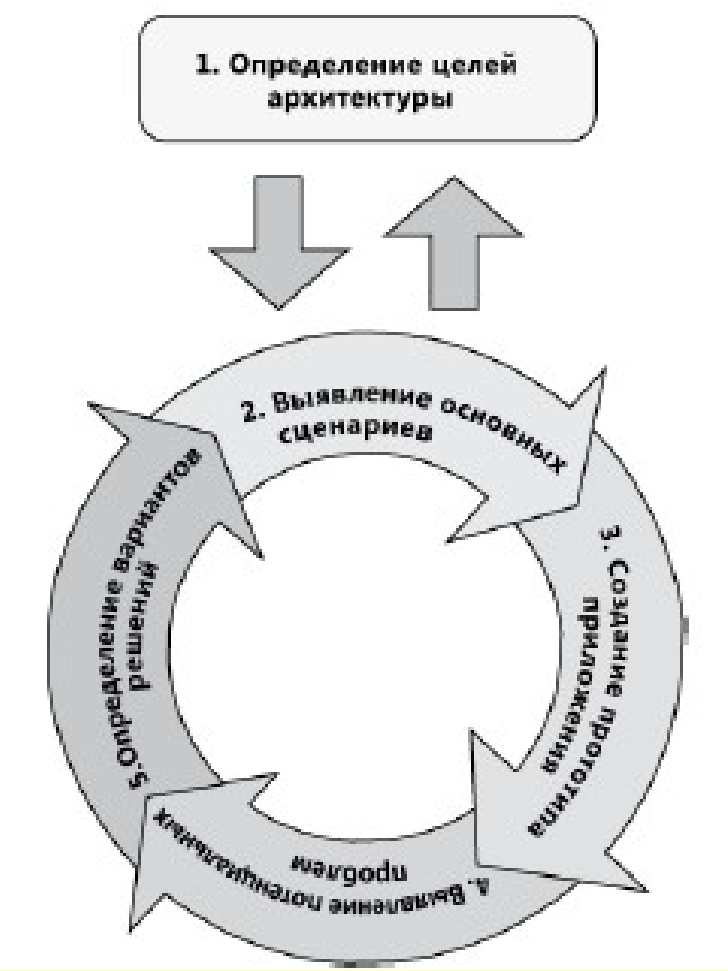
2. **Основні сценарії.**
Використовуйте основні сценарії, щоб зосередитися на тому, що має першорядне значення, і перевіряйте можливі варіанти архітектур на відповідність цих сценаріїв.

Вхідні та вихідні данні та етапи проектування



3. Загальне уявлення щодо застосування.
Визначте тип програми, архітектуру розгортання, архітектурні стилі та технології, щоб забезпечити відповідність вашого дизайну реальним умовам, в яких буде функціонувати створюване застосування.
4. Потенційні проблеми.
Виявити основні проблемні області на підставі параметрів якості і потреби в наскрізний функціональності.

Вхідні та вихідні данні та етапи проектування



5. **Варіанти рішень.**
У кожній ітерації повинен бути створений «пілот» або прототип архітектури, що є розвитком і доопрацюванням рішення. Перш ніж переходити до наступної ітерації, необхідно переконатися у відповідності цього прототипу основним сценаріям, проблемам і обмеженням розгортання

Вхідні та вихідні данні та етапи проектування

При ітеративном поході до архітектури часто є спокуса виконувати ітерації в горизонтальному напрямку, в рамках окремих шарів застосування, а не в вертикальному напрямку, що змушує думати про функціональність, що виходить за рамки шарів і яка складає окрему можливість (варіант використання), яка є значащою користувачам.

При виконанні ітерацій в горизонтальній площині є загроза витрачання цінного часу на реалізацію великого числа функцій (можливо неважливих, або неможливих до втілення з причин на інших шарах, або з причин вимог бізнес-правил) до того, як користувачі зможуть їх перевірити.

Вхідні та вихідні данні та етапи проектування

Не прагніть створити архітектуру за одну ітерацію.
Кожна ітерація повинна розкривати додаткові деталі. Але не угрузайте в деталях, зосередьтеся на основних етапах і створюйте інфраструктуру, на якій може бути заснована ваша архітектура і дизайн.

Визначення цілей архітектури

Цілі архітектури - це завдання і обмеження, що окреслюють архітектуру і процес проектування, що визначають обсяг робіт і допомагають зрозуміти, коли необхідно зупинитися.

Розглянемо ключові моменти у визначенні цілей архітектури:
- Початкове визначення завдань архітектури. Від цих завдань буде залежати час, що витрачається на кожну фазу проектування архітектури: створюєте прототип, проводите тестування можливих варіантів реалізації або виконуєте тривалий процес розробки архітектури для нового застосування.
- Визначення споживачів архітектури.
Чи буде конструкція, що розробляється використовуватися іншими архітекторами, або вона призначається для розробників і тестувальників, IT-фахівців і керівників.
- Визначення обмежень.
Вивчіть всі опції і обмеження технології, що застосовується, обмеження використання та розгортання. Повністю розберіться з усіма обмеженнями на початку роботи, щоб не витрачати час або не стикатися з сюрпризами в процесі розробки програми.

Визначення цілей архітектури

Перерахуємо можливі цілі:
- Створення повного дизайну застосування.
- Створення прототипу.
- Визначення основних технічних ризиків.
- Тестування можливих варіантів реалізації.
- Створення загальних моделей для полегшення розуміння системи.

Час та об'єм робіт

Виходячи зі свого розуміння цілей, оцінюйте необхідну кількість часу і сил для кожного етапу, це допоможе прийти до бачення результату і чіткому визначенню цілей і пріоритетів архітектури.

Ключові сценарії

Варіант використання (use case) - це опис ряду взаємодій між системою і одним або більше дійовими особами (або користувачем, або іншою системою).

Сценарій - це більш широкий і всеосяжний опис взаємодії користувача з системою, ніж гілка варіанти використання.

Основною метою при продумуванні архітектури системи повинно бути виявлення декількох ключових сценаріїв, що допоможе при прийнятті рішення про архітектуру.

Завдання - знайти баланс між цілями користувача, бізнесу і системи.

Ключові сценарії - це найбільш важливі сценарії для успіху створюваного додатка. Ключовий сценарій відповідає одному або більше з таких критеріїв:
- Він представляє проблемну область - значну невідому область або область значного ризику.
- Він посилається на істотний для архітектури варіант використання.
- Він представляє взаємодію параметрів якості з функціональністю.
- Він є компромісом між параметрами якості.

Важливі з точки зору архітектури варіанти використання

До важливих з точки зору архітектури варіантів використання відносяться:
- Бізнес-критичний (Business Critical).
Варіант використання, що має високий рівень використання або особливу важливість для користувачів або інших зацікавлених сторін, в порівнянні з іншими функціями, або передбачає високий ризик.

- Хто має великий вплив (High Impact).
Варіант використання охоплює і функціональність, і параметри якості, або представляє наскрізну функцію, що має глобальний вплив на шари і рівні програми.
Прикладами можуть служити особливо вразливі з точки зору безпеки операції Create, Read, Update, Delete (CRUD).

Важливі з точки зору архітектури варіанти використання

Використовуйте ці сценарії і варіанти використання для тестування свого дизайну і виявлення можливих проблем.

При продумуванні варіантів використання і сценаріїв зверніть увагу на таке:
- На ранніх етапах розробки дизайну скоротіть ризик шляхом створення варіанту архітектури, що підтримує важливі з точки зору архітектури наскрізні сценарії, що зачіпають всі шари архітектури.
- Використовуючи модель архітектури як керівництво, вносьте зміни в архітектуру, дизайн і код для реалізації сценаріїв, функціональних вимог, технологічних вимог, параметрів якості та обмежень.
- Створіть модель архітектури на підставі відомих на даний момент відомостей і складіть список питань, відповіді на які повинні бути дані в наступних сценаріях та ітераціях.
- При внесені істотних змін в архітектуру і дизайн, створіть варіант використання, який буде відображати і застосовувати ці зміни.

Загальне представлення застосування

Створіть загальне уявлення того, як буде виглядати готове застосування.
Це загальне уявлення дозволить зробити архітектуру більш відчутною, зв'яже її з реальними обмеженнями і рішеннями:

1. Визначення типу застосування.

Чи буде це мобільне застосування, насичений клієнт, насичене Інтернет-застосування, сервіс, Веб-застосування або деяке сполучення цих типів?

2. Визначення обмежень розгортання.

Необхідно врахувати корпоративні політики та процедури, а також середовище, в якому планується розгортання застосування.
Якщо цільове середовище фіксоване або негнучке, конструкція застосування повинна відображати існуючі в цьому середовищі обмеження.
Також в конструкції застосування повинні бути враховані нефункціональні вимоги (Quality-of-Service, QoS), такі як безпека і надійність.
Іноді необхідно поступитися чимось в дизайні або через обмеження в підтримуваних протоколах або через топологію мережі. Виявлення вимог і обмежень, присутніх між архітектурою застосування і архітектурою середовища на ранніх етапах проектування дозволяє вибрати відповідну топологію розгортання і вирішити конфлікти між застосуванням і цільовим середовищем.

Загальне представлення застосування

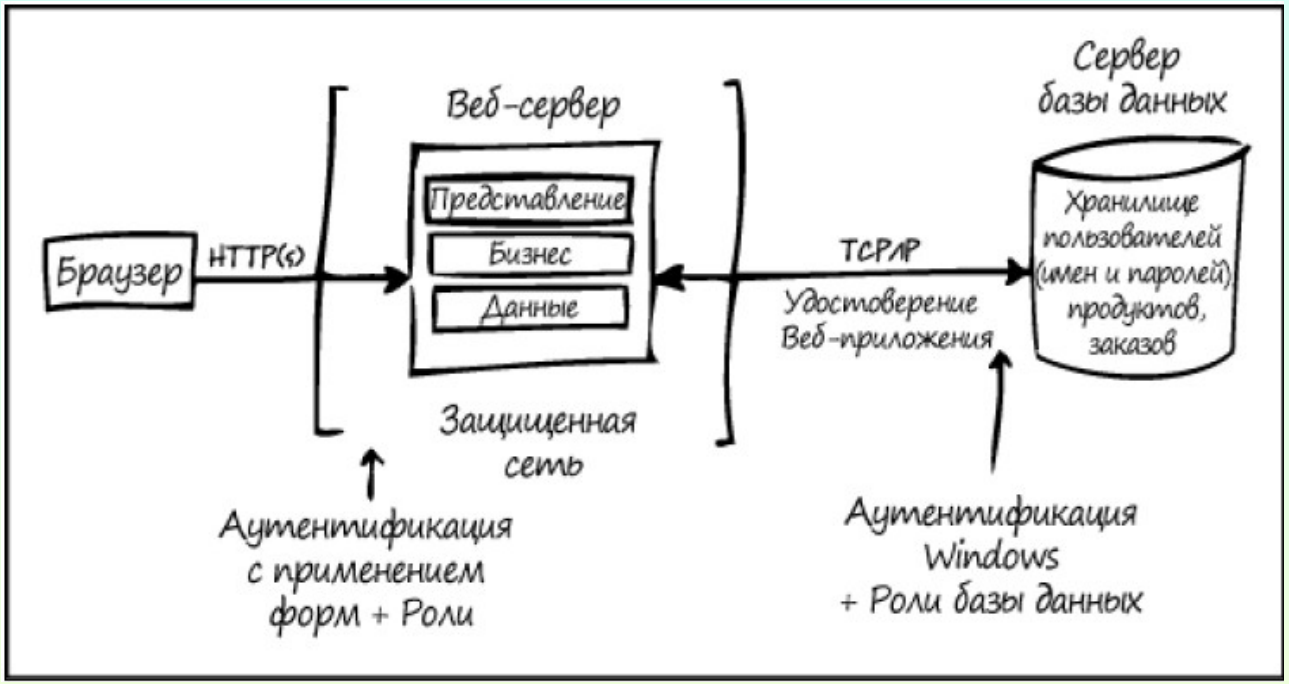
3. Визначення значущих архітектурних стилів проектування.

4. Вибір відповідних технологій.
На підставі типу застосування, стилів, основних параметрів якості та інших обмежень (політик організації, обмежень середовища, кваліфікації штату і т.інше) вибираємо відповідні технології, які будуть використовуватися в майбутній системі.

Підходящі технології

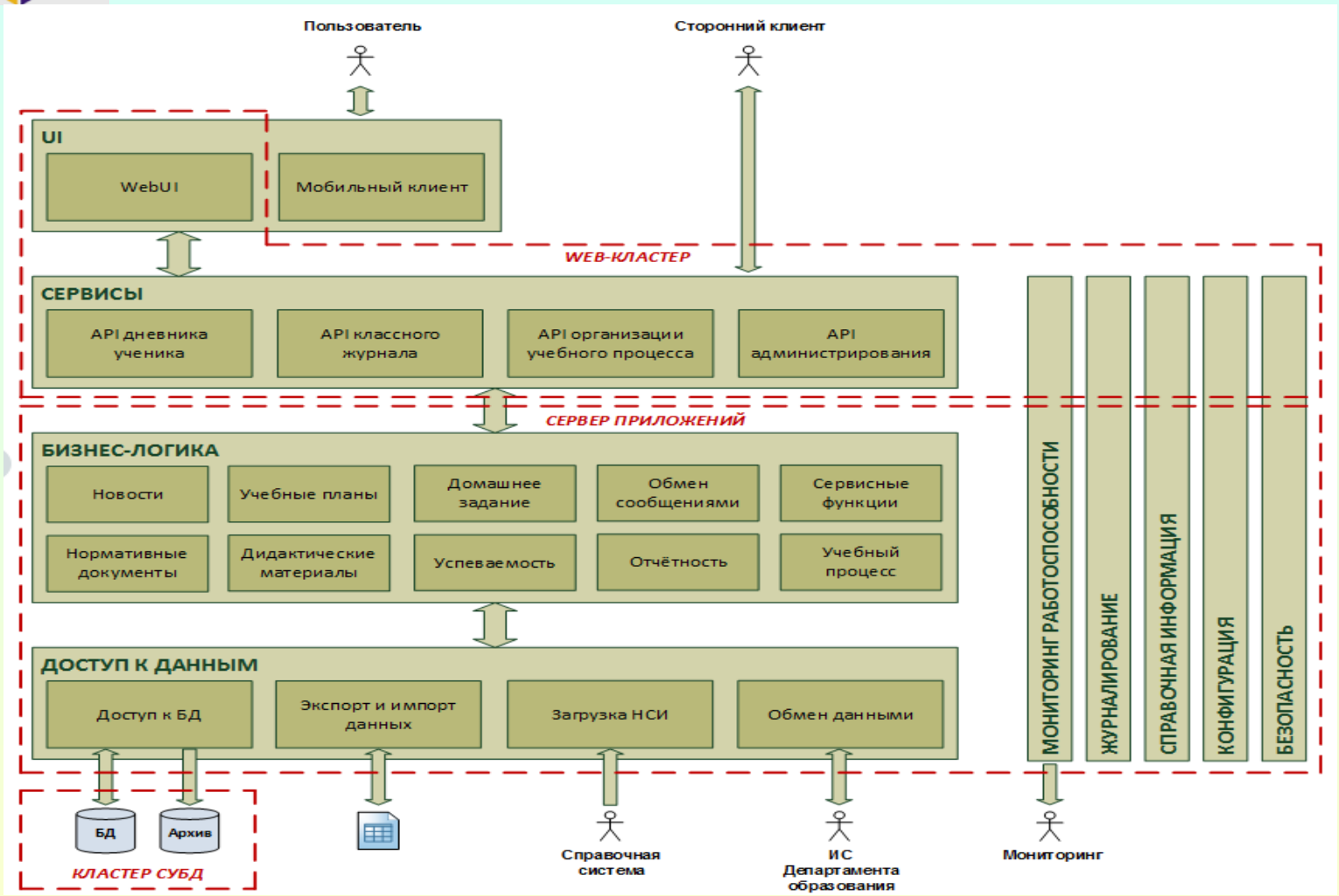
- Мобільні додатки.
Можуть використовуватися технології шару представлення: .NET Compact Framework, ASP.NET для мобільних пристроїв і Silverlight для мобільних пристроїв.
- Насичені клієнтські програми.
З насиченими UI, які розгортаються і виконуються на клієнті, може бути використано сполучення технологій шару представлення Windows Presentation Foundation (WPF), Windows Forms і XAML Browser Application (XBAP).
- Насичені клієнтські Інтернет-застосування (RIA).
Для насичених UI в рамках Веб-браузера можуть використовувати модуль Silverlight або Silverlight в поєднанні з AJAX.
- Веб-застосування.
Для створення Веб-застосувань можуть використовуватися ASP.NET WebForms, AJAX, Silverlight, ASP.NET MVC і ASP.NET Dynamic Data.
- Сервісні програми.
Для створення сервісів, що надають функціональність зовнішнім споживачам систем і сервісів, можуть використовуватися Windows Communication Foundation (WCF) та ASP.NET Web services (ASMX).

Графічне представлення архітектури



Важливо, графічно представити розроблювану архітектуру.
Показати основні обмеження і прийняті рішення для того, щоб позначити межі і почати обговорення.
Це має подвійну цінність:
- Якщо ви не можете наочно уявити архітектуру, значить, ви не повністю розумієте її.
- Якщо ви змогли зобразити чітку і коротку діаграму, вона буде зрозумілою іншим, і буде набагато простіше пояснювати деталі.

Графічне представлення архітектури



https://habrahabr.ru/post/257677/

Основні проблеми

Визначте основні потенційні проблеми архітектури свого застосування, щоб зрозуміти області, в яких найбільш ймовірне виникнення помилок.

До потенційних проблем відносяться поява нових технологій і критично важливі бізнес-вимоги.

Наприклад,
«Чи можу я переходити з одного сервісу стороннього виробника до іншого?»,
«Чи можу я додати підтримку нового типу клієнта?»,
«Чи можу я швидко змінювати бізнес-правила оплати послуг?» І
«Чи можу я перейти до нової технології для компонента X? ».

Незважаючи на те, що це вкрай узагальнені аспекти, як правило, при реалізації вони (і інші зони ризику) проектуються в параметри якості та наскрізну функціональність

Параметри якості

Параметри якості - це загальні властивості архітектури, які впливають на поведінку під час виконання, на дизайн системи і на взаємодію з користувачем.
Та ступінь, з якою застосування забезпечує необхідне сполучення параметрів якості, таких як зручність і простота використання, продуктивність, надійність і безпеку, визначає успішність дизайну і загальну якість програмного продукту.

Важливість або пріоритетність кожного з параметрів якості для різних систем різна. Наприклад, для бізнес-застосувань (line-of-business, LOB) продуктивність, масштабованість, безпека і зручність використання будуть більш важливі, ніж можливість взаємодії з іншими системами.
Для коробкового застосування така можливість буде мати більше значення, ніж для LOB-застосування.

Параметри якості представляють функціональні області, які потенційно можуть впливати на все застосування, на всі його шари і рівні.

Параметри якості

Список, що систематизує відомості про параметри якості і допомагає зрозуміти, на які сценарії найбільш ймовірно їх вплив:

- Загальносистемні якості.

Загальні якості системи в цілому, такі як можливість технічної та тестової підтримки.

- Якості часу виконання.

Якості системи, притаманні безпосередньо під час виконання, такі як доступність, можливість взаємодії з іншими системами, керованість, продуктивність, надійність, масштабованість і безпека.

- Конструктивні якості.

Якості, що відображають дизайн системи, такі як концептуальна цілісність, гнучкість, зручність і простота обслуговування і можливість повторного використання.

- Для користувача якості.

Зручність і простота використання системи.

Наскрізна функціональність

Наскрізна функціональність - це аспекти дизайну, які можуть застосовуватися до всіх верств, компонентів і рівнів. Також це ті області, в яких найчастіше робляться помилки, які мають великий вплив на дизайн.
Приклади наскрізної функціональності:
- Аутентифікація і авторизація.
Як правильно вибрати стратегію аутентифікації і авторизації, передачі ідентифікаційних даних між шарами і рівнями і зберігання посвідчень користувачів.
- Кешування.
Як правильно вибрати техніку кешування, визначити дані, що підлягають кешуванню, де кешувати дані і як вибрати відповідну політику закінчення терміну дії.
- Зв'язок.
Як правильно вибрати протоколи для зв'язку між шарами і рівнями, забезпечення слабкого зв'язування (взаємозалежність) між шарами, здійснення асинхронного обміну даними і передачі конфіденційних даних.

Наскрізна функціональність

- Управління конфігурацією.
Як виявити дані, які повинні бути налаштованими, де і як зберігати дані конфігурації, як захищати конфіденційні дані конфігурації і як обробляти їх в рамках сервера або кластера.
- Управління винятками.
Як обробляти і протоколювати виключення і забезпечувати повідомлення в разі потреби.
- Протоколювання і інструментування.
Як вибрати дані, які підлягають протоколюванню, як зробити протоколювання налаштованим, і як визначити необхідний рівень інструментування.
- Валідація.
Як визначити, де і як проводити валідацію; як вибрати методики для перевірки довжини, діапазону, розміру і типу; як запобігти і відхилити введення неприпустимих значень; як очистити потенційно зловмисне і небезпечне введення; як визначити і повторно використовувати логіку валідації на різних шарах і рівнях застосування.

Питання, які потребують особливої уваги при проектуванні

Аналіз параметрів якості і наскрізної функціональності в зв'язку з наявними вимогами дозволяє зосередитися на конкретних функціональних областях.

Наприклад, безпека, безсумнівно, є життєво важливим фактором при проектуванні і присутній у багатьох шарах і в багатьох аспектах архітектури.

Наскрізна функціональність, що відноситься до безпеки, є орієнтиром, що вказує на області, на яких слід акцентувати увагу. Категорії наскрізної функціональності можуть використовуватися для поділу архітектури застосування для подальшого аналізу і виявлення вразливих місць застосування.

Питання, які потребують особливої уваги при проектуванні

Під час обговорення наскрізної функціональності, що відноситься до безпеки рекомендується звернути увагу на такі питання:

- Аудит і протоколювання.

Хто, що зробив і коли? Додаток функціонує в нормальному режимі? Аудит займається питаннями реєстрації подій, пов'язаних з безпекою. Протоколювання стосується того, як додаток публікує дані про свою роботу.

- Аутентифікація.

Хто ви? Аутентифікація - це процес, при якому одна сутність чітко і однозначно ідентифікує іншу сутність, зазвичай це робиться за допомогою таких облікових даних, як ім'я користувача і пароль.

- Авторизація.

Що ви можете робити? Авторизація визначає, як додаток управляє доступом до ресурсів та операцій.

- Управління конфігурацією.

В якому контексті виконується застосування? До яких баз даних підключається? Як виконується адміністрування програми? Як захищені ці налаштування? Управління конфігурацією визначає, як застосування реалізує ці операції і завдання.

Питання, які потребують особливої уваги при проектуванні

-Шифрування.
Як реалізований захист секретів (конфіденційних даних)?
Як здійснюється захист від несанкціонованого доступу даних і бібліотек (цілісності)?
Як передаються випадкові значення, які повинні бути криптографічно стійкими?
Шифрування і криптографія займається питаннями реалізації конфіденційності і цілісності.
- Обробка винятків.
Що робить застосування при невдалій спробі здійснити виклик його методу?
Наскільки повні дані про помилку воно надає?
Чи забезпечує воно зрозумілі для кінцевих користувачів повідомлення про помилки?
Чи повертає воно цінні відомості про виключення (помилку) викликаючій стороні?
Чи виконується коректна обробка події збою?
Чи надає застосування адміністраторам необхідну інформацію для проведення аналізу основних причин збою?
Обробка винятків стосується того, як виключення обробляються в застосуванні.

Питання, які потребують особливої уваги при проектуванні

-**Валідація вхідних даних.**
Як визначити, що в додаток надходять дані дійсні і безпечні?
Чи виконується обмеження введення через точки входу і кодування виведення через точки виходу?
Чи можна довіряти таким джерелам даних, як бази даних і загальні файли?
Перевірка введення стосується питань фільтрації, очищення або відхилення даних, що вводяться в застосування перед їх додаткової обробкою.
- Конфіденційні дані.
Як застосування працює з конфіденційними даними?
Чи забезпечує воно захист конфіденційних даних користувачів і застосування?
Тут вирішуються питання обробки застосуванням будь-яких даних, які повинні бути захищені або при зберіганні в пам'яті, або при передачі по мережі, або при зберіганні в постійних сховищах.
- Управління сеансами.
Як застосування обробляє і захищає сеанси користувачів?
Сеанс - це ряд взаємозв'язаних взаємодій користувача і застосування.
Ці питання допоможуть прийняти основні проектні рішення з безпеки застосування і задокументувати їх як частину архітектури.

Питання, які потребують особливої уваги при проектуванні

Питання, які потребують особливої уваги при проектуванні

Варіанти рішень

Визначивши основні проблеми, можна приступати до створення вхідної базової архітектури і її деталізації для отримання можливого варіанту архітектури.

В ході цього процесу можна використовувати пілотні архітектури для більш докладного розгляду певних областей дизайну або для перевірки нових ідей. Потім виконується перевірка нового варіанту архітектури на відповідність основним сценаріям і заданим вимогам і знову повторюється ітеративний цикл з доопрацювання дизайну.

Важливо, особливо якщо проектування і розробка ведуться за гнучким процесом, щоб ітерація включала як проектування архітектури, так і розробку реалізації. Це допоможе уникнути масштабного проектування наперед.

Варіанти рішень

Базова архітектура і можливі варіанти архітектури
Базова архітектура описує існуючу систему, ту як вона виглядає сьогодні.

Для нового проекту вхідна базова архітектура - це перше високорівневе представлення архітектури, на підставі якого будуть створюватися можливі варіанти архітектури.

Можливий варіант архітектури включає тип програми, архітектуру розгортання, архітектурний стиль, обрані технології, параметри якості та наскрізну функціональність.

На кожному етапі розробки дизайну будьте впевнені, що розумієте основні ризики і вживаєте заходи по їх скороченню, проводьте оптимізацію для ефективної і раціональної передачі проектних відомостей і створюєте архітектуру, забезпечуючи гнучкість і можливість реструктуризації.

Можливо, архітектуру доведеться змінювати кілька разів, використовувати декілька ітерацій, можливих варіантів і безліч пілотних архітектур.

Варіанти рішень

Базова архітектура і можливі варіанти архітектури
Якщо можливий варіант архітектури є поліпшенням, він може стати базою для створення і тестування нових можливих варіантів.

Ітеративний і інкрементний підхід дозволяє позбутися від великих ризиків на початку, ітеративно формувати архітектуру і через тестування підтверджувати, що кожна нова базова архітектура є поліпшенням попередньої.

Наступні питання допоможуть протестувати новий варіант архітектури, отриманий на підставі «пілота» архітектури:
- Дана архітектура забезпечує рішення без додавання нових ризиків?
- Дана архітектура усуває більше відомих ризиків, ніж попередня ітерація?
- Дана архітектура реалізує додаткові вимоги?
- Дана архітектура реалізує важливі з точки зору архітектури варіанти використання?
- Дана архітектура реалізує аспекти, пов'язані з параметрами якості?
- Дана архітектура реалізує додаткові аспекти наскрізної функціональності?

Варіанти рішень

Пілотні архітектури
Пілотна архітектура (architectural spike) - це тестова реалізація невеликої частини загального дизайну або архітектури застосування.
Її призначення - аналіз технічних аспектів конкретної частини рішення для перевірки технічних припущень, вибору дизайну з ряду можливих варіантів і стратегій реалізації або іноді оцінка термінів реалізації.

Завдяки їх сфокусованості на основних частинах спільного проекту рішення, пілотні архітектури можуть використовуватися для вирішення важливих технічних проблем і для скорочення загальних ризиків і невизначеностей в дизайні.

Що далі?
Після завершення моделювання архітектури можна приступати до доопрацювання дизайну, планування тестів і представлення рішень іншим учасникам процесу.

Варіанти рішень

Керуйтеся наступними рекомендаціями:
- При документуванні можливих варіантів архітектури і варіантів її тестування намагайтеся не захаращувати цей документ, що забезпечить простоту його поновлення. Такий документ може включати відомості про цілі, тип програми, топології розгортання, основні сценарії і вимоги, технології, параметри якості і тести.
- Використовуйте параметри якості для визначення контурів дизайну і реалізації. Наприклад, розробники повинні знати антішаблони для виявлених архітектурних ризиків і використовувати відповідні перевірені схеми для вирішення даних проблем.
- Діліться одержуваними відомостями з учасниками групи і іншими зацікавленими сторонами. До них можуть відноситися група розробки програми, група тестування та адміністратори мережі або системні адміністратори.

Аналіз архітектури

Аналіз архітектури застосування - критично важливе завдання, оскільки дозволяє скоротити витрати на виправлення помилок, якомога раніше виявити і виправити можливі проблеми. Аналіз архітектури слід виконувати часто: по завершенні основних етапів проекту та у відповідь на істотні зміни в архітектурі. Створюйте архітектуру, пам'ятаючи про основні питання, що задаються при такому аналізі, це дозволить як поліпшити архітектуру, так і скоротити час, що витрачається на кожен аналіз.

Основна мета аналізу архітектури - підтвердження придатності базової архітектури та її можливих варіантів, і також перевірка відповідності пропонованих технічних рішень функціональним вимогам і параметрам якості. Крім того, аналіз допомагає виявити проблеми та виявити області, які потребують доопрацювання.

Аналіз архітектури

Оцінки на підставі сценаріїв

Оцінки на підставі сценаріїв - це потужний метод аналізу дизайну архітектури. При такій оцінці основна увага спрямована на найбільш важливі з точки зору бізнесу сценарії і мають найбільший вплив на архітектуру.

Одна з типових методик:

- Метод аналізу архітектури ПЗ (Software Architecture Analysis Method, SAAM). Спочатку SAAM створювався для оцінки модифікованості, але пізніше був розширений для аналізу архітектури щодо показників якості, таких як модифікованість, портативність, розширюваність, інтегрованість і функціональне охоплення.
- Метод аналізу архітектурних компромісів (Architecture Tradeoff Analysis Method, ATAM). ATAM - це допрацьована і вдосконалена версія SAAM, яка дозволяє переглядати архітектурні рішення щодо вимог параметрів якості і того, наскільки добре ці рішення відповідають конкретним цільовим показниками якості.

Аналіз архітектури

Оцінки на підставі сценаріїв

- Активний аналіз конструкції (Active Design Review, ADR). ADR найбільше підходить для незавершених архітектур або архітектур, що знаходяться в процесі розробки. Основна відмінність цього методу в тому, що аналіз більш сфокусований на наборі проблем або окремих розділах, а не на проведенні загального аналізу.
- Активний аналіз проміжних конструкцій (Active Reviews of Intermediate Designs, ARID). ARID поєднує в собі підхід ADR аналізу архітектури, що знаходиться в процесі розробки, з фокусом на наборі проблем і підході методів ATAM і SAAM аналізу на підставі сценарію з основною увагою на параметрах якості.
- Метод аналізу рентабельності (Cost Benefit Analysis Method, CBAM). Метод CBAM основну увагу приділяє аналізу витрат, вигод і планування наслідків архітектурних рішень.
- Аналіз модифікованості на рівні архітектури (Architecture Level Modifiability Analysis, ALMA). ALMA оцінює модифікованість архітектури для систем бізнес-аналітики (business information systems, BIS).
- Метод оцінки сімейства архітектур (Family Architecture Assessment Method, FAAM). FAAM оцінює архітектури сімейства інформаційних систем з точки зору можливості взаємодії та розширюваності.

Представлення дизайну архітектури

Подання дизайну є дуже важливим для проведення аналізу архітектури, також це гарантує, що все реалізовано правильно.

Дизайн архітектури повинен бути представлений всім зацікавленим сторонам, включаючи групу розробки, системних адміністраторів і операторів, власників бізнесу та ін.

Один із способів подання архітектури - карта важливих рішень.
Існує кілька широко відомих методів опису архітектури для її подання:
- 4 + 1. В даному підході використовується п'ять представлень готової архітектури.
Чотири представлень описують архітектуру з різних точок зору:
- логічне представлення (наприклад, об'єктна модель),
- представлення процесів (наприклад, аспекти паралелізму і синхронізації),
- фізичне представлення (схема програмних рівнів і функцій в розподіленому апаратному середовищі) і
- представлення для розробників.
П'яте представлення показує сценарії і варіанти використання ПЗ.

Представлення дизайну архітектури

- Гнучке моделювання.
Даний підхід наслідок ідеї того, що вміст важливіше ніж представлення.
Це забезпечує простоту, зрозумілість, достатню точність і однаковість створюваних моделей. Простота документа гарантує активну участь зацікавлених сторін у моделюванні артефактів.
- IEEE 1471.
IEEE тисячу чотиреста сімдесят один - скорочена назва стандарту, формально відомого як ANSI / IEEE 1471-2000, який збагачує опис архітектури, зокрема, надаючи конкретне значення контексту, уявленням і зрізам.
- Уніфікована мова моделювання (Unified Modeling Language, UML).
Цей підхід забезпечує три представлення моделі системи. Представлення функціональних вимог (функціональні вимоги системи з точки зору користувача, включаючи варіанти використання); статичне структурне представлення (об'єкти, атрибути, відносини і операції, включаючи діаграми класів); і представлення динамічної поведінки (взаємодія об'єктів і зміни внутрішнього стану об'єктів, включаючи діаграми послідовностей, діяльностей і станів).

Заключна частина

Під час проектування архітектури і дизайну не треба забувати про паралельну реальну реалізацію частини архітектури.

Методика побудови архітектури і дизайну

Дякую за увагу