Лекция 3. Проектирование архитектуры

Основные понятия проектирования архитектуры ПО

1. Понятие архитектуры ПО и рамочная модель по IEEE 1471
2. Архитектурные стили и архитектурные слои
3. Процесс проектирования архитектуры

Понятие архитектуры ПО

Архитектура [IEEE 1471] - базовая организация системы, воплощенная в:

- компонентах, объединенных для выполнения определенных функций,
- их отношениях между собой и с окружением, определяющим ход и обстоятельства влияний на систему, а также
- принципы, определяющие проектирование и развитие системы.

Является реализацией нефункциональных требований к системе, в то время как проектирование ПО — реализация функциональных требований.

Рамочная модель разработки архитектуры по IEEE 1471

Участник (заинтересованная сторона)

- любой человек, правомочно заинтересованный в проекте. *Например, заказчики, разработчики, обслуживающий персонал, руководство*.

Точка зрения

- шаблон или обобщение представления, задает правила для представлений. Точки зрения подбираются в соответствии с соображениями участников. *Например, точка зрения развертывания, точка зрения защиты безопасности, точка зрения надежности.*

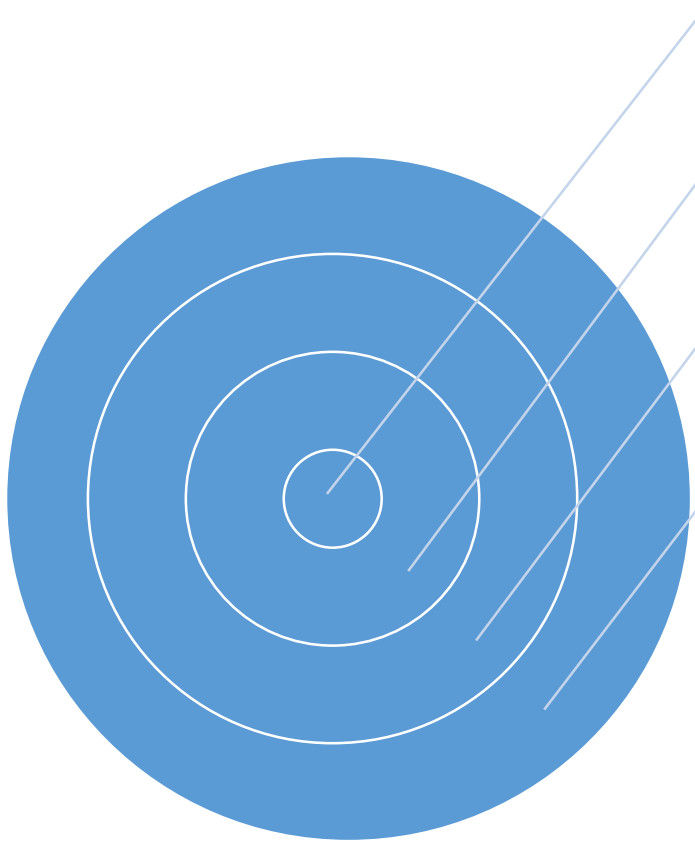
Представление

- это отображение целой системы через призму определенных взглядов. Представление можно рассматривать как экземпляр точки зрения. Представление состоит из одной или нескольких моделей.

Модель

- это абстракция или упрощенное отображение системы или ее части: UML (универсальный язык моделирования), диаграммы сущностей и связей (ER-диаграммы), нестандартизированные системы обозначений (блок-схемы, стрелочные диаграммы).

Причины проектирования архитектуры



Программы читаю чаще, чем пишут

Разработка не может быть окончательной

Разработчику необходимо понимать как общую концепцию, так и детали

Стоимость модификации старых программ многократно превосходит стоимость создания новых

«Экономика — двигатель проектирования программ»

(Эдвард Йордон, 1979г.)

- **S общ = S разработки + S сопровождения**
- **S сопровождения = S понимания + S изменений + S тестирования + S поставки**

Архитектурные стили

- *Структура*
  - Компонентная архитектура (Component-based)
  - Монолитное приложение (Monolithic application)
  - **Расслоение (Layered)**
- *Распределенные системы*
  - Клиент-сервер (Client-server)
  - Точка-точка (Peer-to-peer)
  - Сервис-ориентированная (Service-oriented)
- *Сообщения*
  - Событийная архитектура (Event-driven)
  - Публикация/Подписка (Publish-subscribe)
  - Шина сообщений (Message bus)

Представления (Интерфейсный)

- Отображение
- Обработка событий

Сервисный (Операционный / Служб)

- Координирование операций
- Внешний API

Бизнес-логики (Предметной области / Домен)

Источников данных (Инфраструктурный)

- Файловая система
- СУБД
- Сторонние API

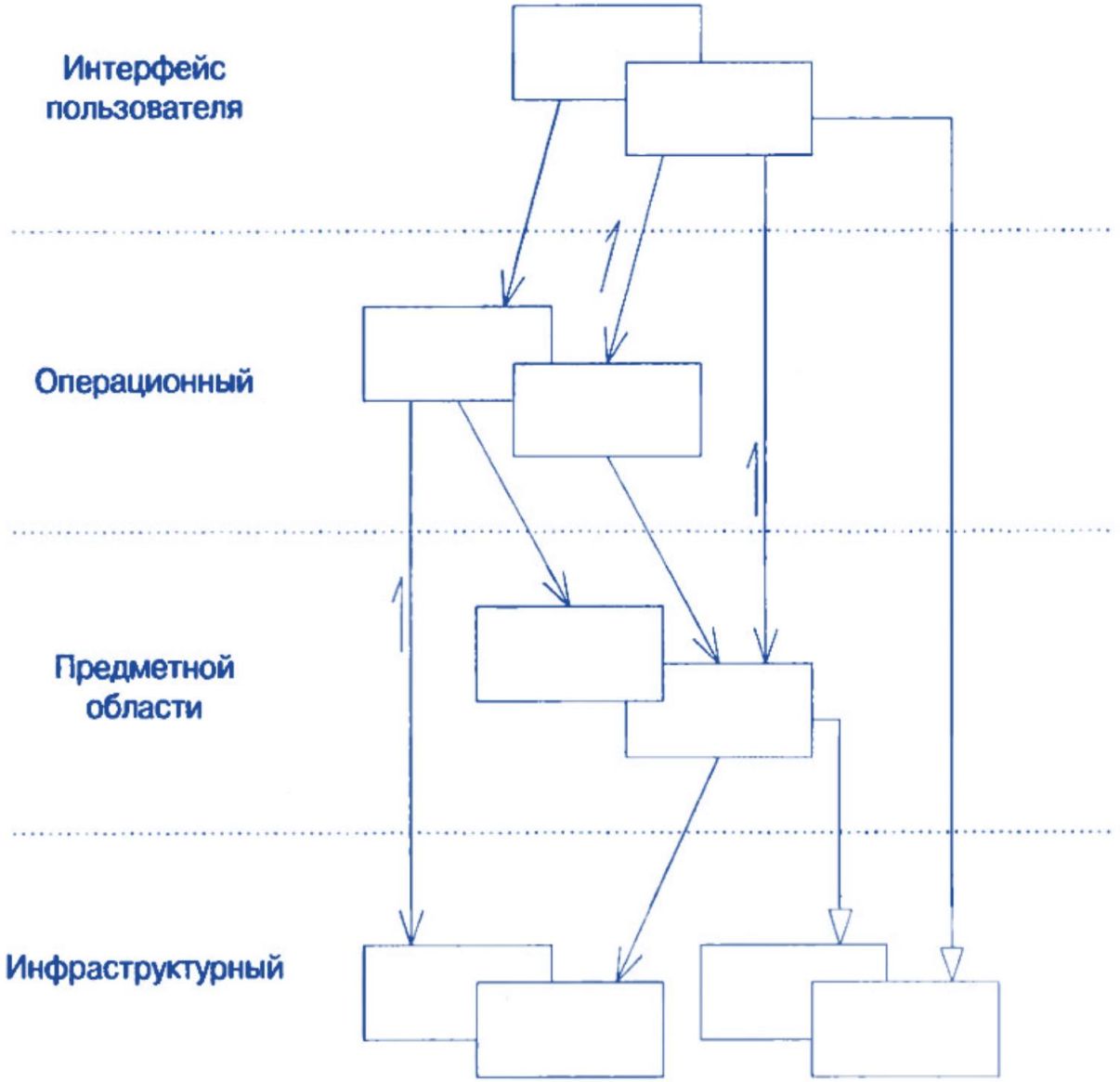
Архитектурные слои

«+»

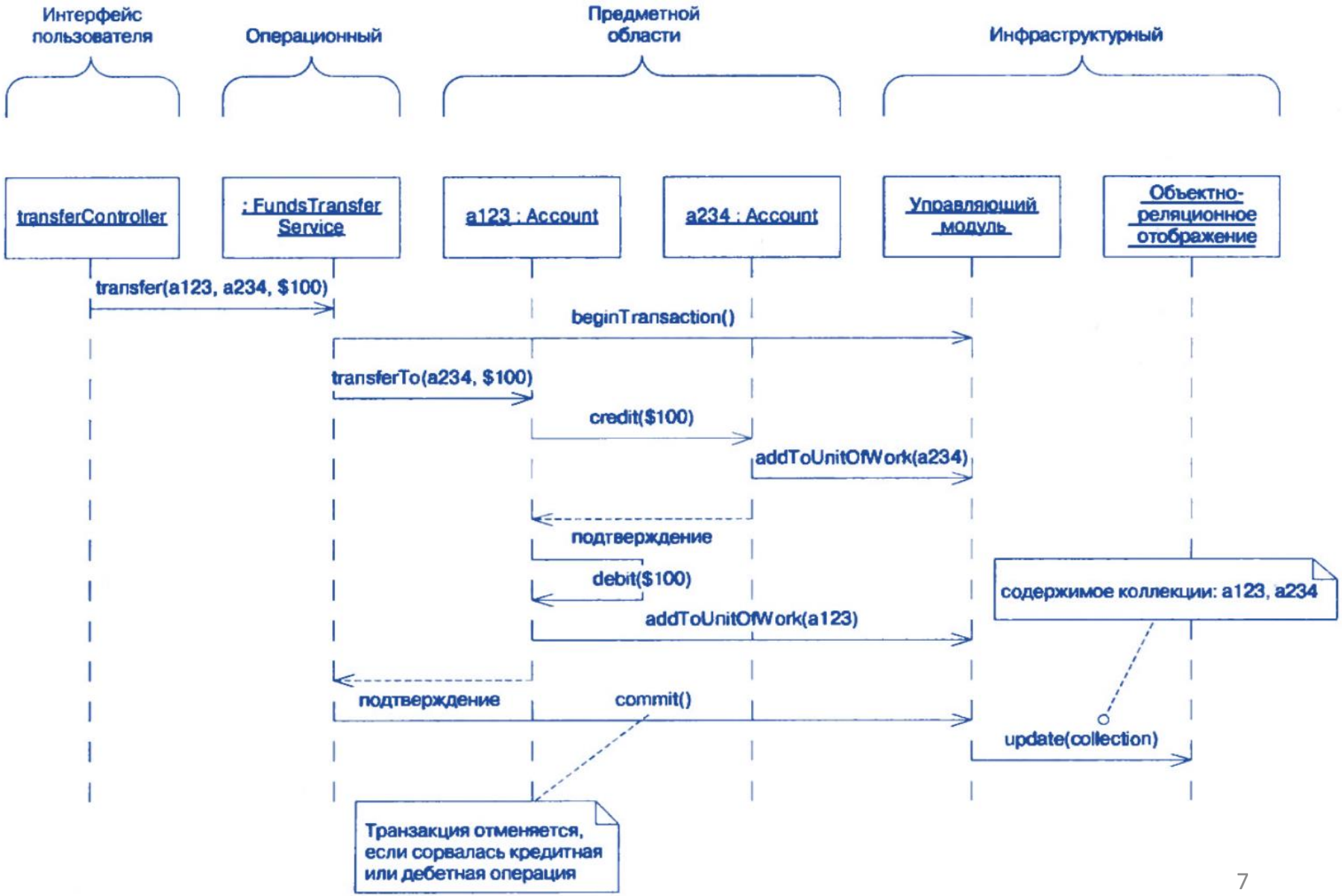
- Слой — самодостаточное целое
- Возможна альтернативная реализация
- Минимальная зависимость слоев (Изоляция)
- Несколько вышележащих слоев

«-»

- Каскадное изменение слоев
- Снижение производительности
- Размытые границы ответственности

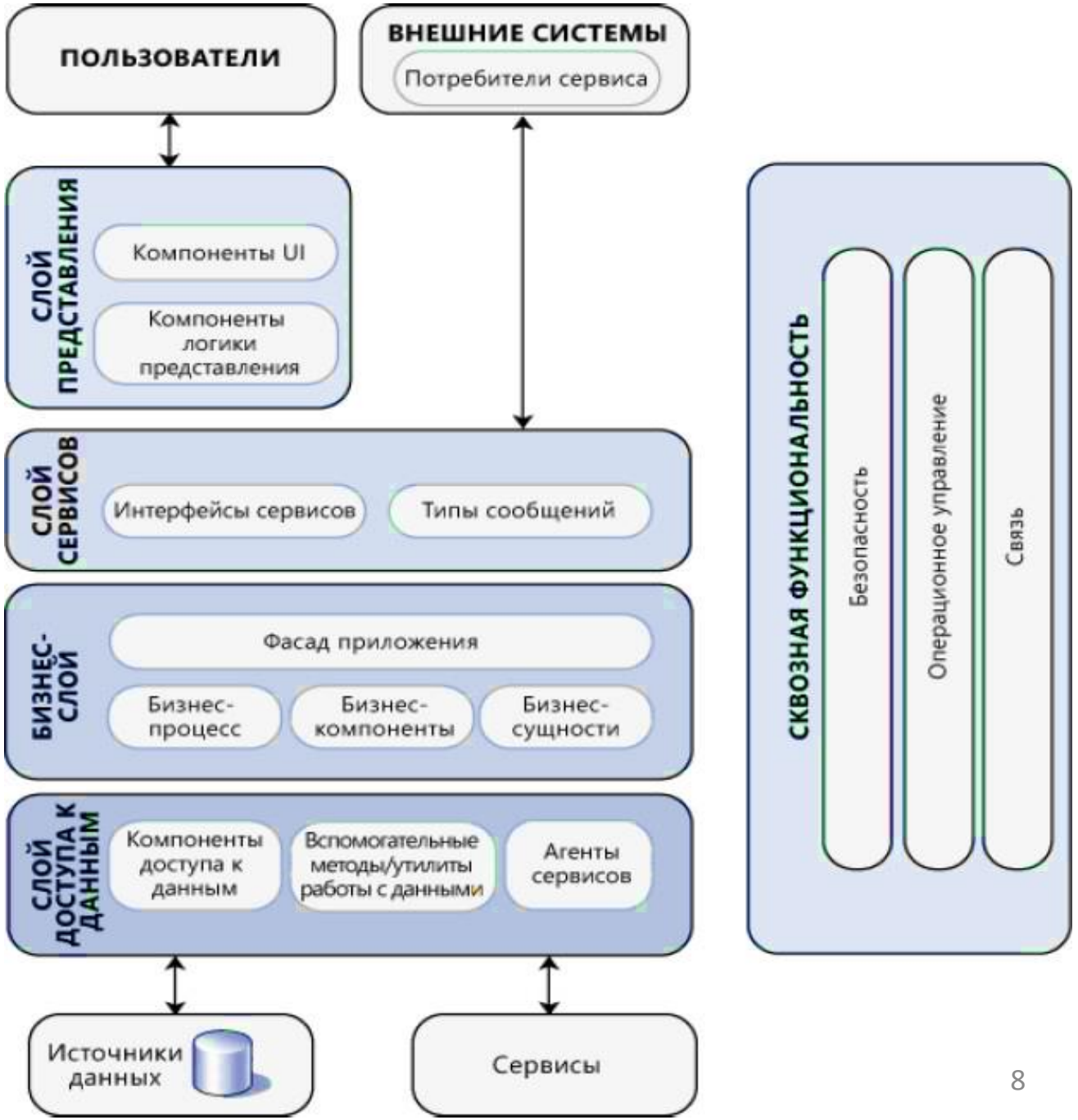


Пример
расслоения



Назначение архитектуры

организация компонентов с целью обеспечения определенной функциональности



Основные принципы проектирования

Разделение функций

- Разделите приложение на отдельные компоненты с, по возможности, минимальным перекрытием функциональности. Важным фактором является предельное уменьшение количества точек соприкосновения, что обеспечит высокую связность (high cohesion) и слабую связанность (low coupling). Неверное разграничение функциональности может привести к высокой связанности и сложностям взаимодействия, даже несмотря на слабое перекрытие функциональности отдельных компонентов.

Принцип единственности ответственности

- Каждый отдельно взятый компонент или модуль должен отвечать только за одно конкретное свойство/функцию или совокупность связанных функций.

Принцип минимального знания (Law of Demeter, LoD)

- Известен как Закон Деметера. Компоненту или объекту не должны быть известны внутренние детали других компонентов или объектов.

Не повторяйтесь (Don’t repeat yourself, DRY)

- Намерение должно быть обозначено только один раз. В применении к проектированию приложения это означает, что определенная функциональность должна быть реализована только в одном компоненте и не должна дублироваться ни в одном другом компоненте.

Минимизируйте проектирование наперед

- Проектируйте только то, что необходимо. В некоторых случаях, когда стоимость разработки или издержки в случае неудачного дизайна очень высоки, может потребоваться полное предварительное проектирование и тестирование. В других случаях, особенно при гибкой разработке, можно избежать масштабного проектирования наперед (big design upfront, BDUF). Если требования к приложению четко не определены, или существует вероятность изменения дизайна со временем, старайтесь не тратить много сил на проектирование раньше времени. Этот принцип называют (You ain’t gonna need it, YAGNI).

Основные вопросы проектирования

Процесс проектирования архитектуры

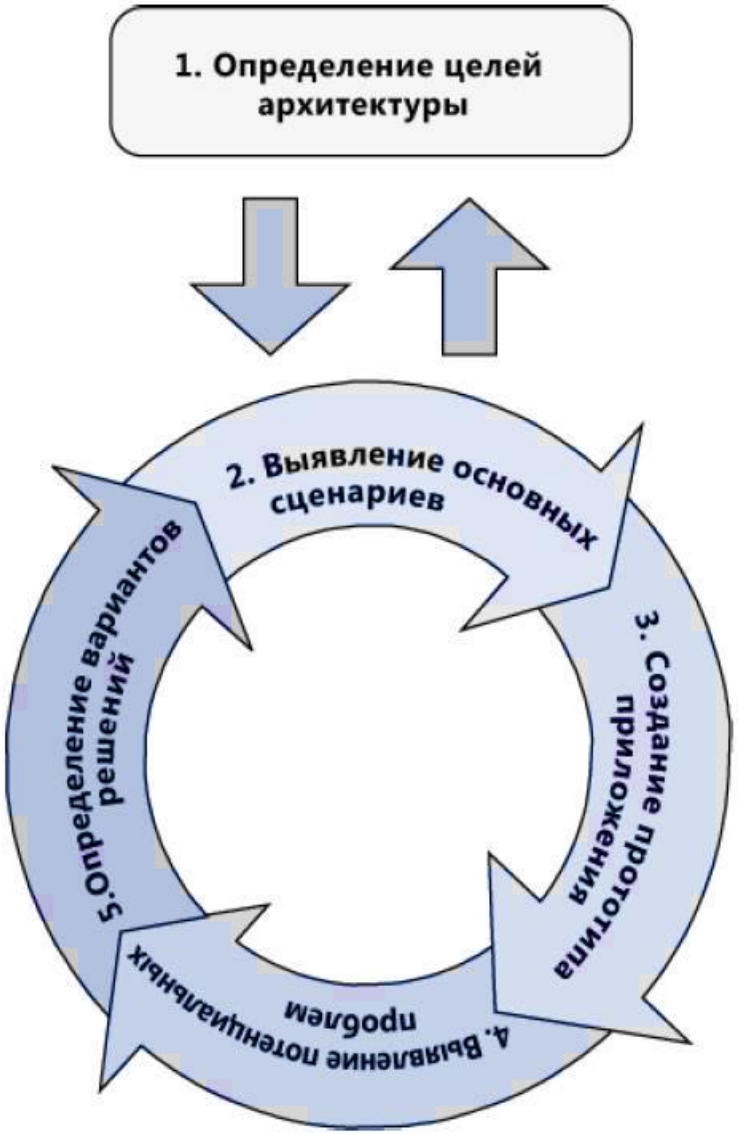
1.Определение целей архитектуры. Наличие четких целей поможет сосредоточиться на архитектуре и правильном выборе проблем для решения. Точно обозначенные цели помогают определить границы каждой фазы: момент, когда завершена текущая фаза и все готово для перехода к следующей.

2.Основные сценарии. Используйте основные сценарии, чтобы сосредоточиться на том, что имеет первостепенное значение, и проверяйте возможные варианты архитектур на соответствие этим сценариям.

3.Общее представление приложения. Определите тип приложения, архитектуру развертывания, архитектурные стили и технологии, чтобы обеспечить соответствие вашего дизайна реальным условиям, в которых будет функционировать создаваемое приложение.

4. Потенциальные проблемы. Выявите основные проблемные области на основании параметров качества и потребности в сквозной функциональности. Это области, в которых чаще всего делаются ошибки при проектировании приложения.

5. Варианты решений. В каждой итерации должен быть создан «пилот» или прототип архитектуры, являющийся развитием и доработкой решения. Прежде чем переходить к следующей итерации, необходимо убедиться в соответствии этого прототипа основным сценариям, проблемам и ограничениям развертывания.



Определение целей архитектуры

Это задачи и ограничения, очерчивающие архитектуру и процесс проектирования, определяющие объем работ и помогающие понять, когда пора остановиться.

Начальное определение задач архитектуры

- От этих задач будет зависеть время, затрачиваемое на каждую фазу проектирования архитектуры. Необходимо решить, что вы делаете: создаете прототип, проводите тестирование возможных вариантов реализации или выполняете длительный процесс разработки архитектуры для нового приложения.

Определение потребителей архитектуры

- Определите, будет ли разрабатываемая конструкция использоваться другими архитекторами, либо она предназначается для разработчиков и тестировщиков, ИТ-специалистов и руководителей. Учтите нужды и подготовленность целевой аудитории, чтобы сделать разрабатываемую конструкцию максимально удобной для них.

Определение ограничений

- Изучите все опции и ограничения применяемой технологии, ограничения использования и развертывания. Полностью разберитесь со всеми ограничениями в начале работы, чтобы не тратить время или не сталкиваться с сюрпризами в процессе разработки приложения.

Ключевые сценарии

Основной целью при продумывании архитектуры системы должно быть выявление нескольких ключевых сценариев, что поможет при принятии решения об архитектуре. Задача – найти баланс между целями пользователя, бизнеса и системы.

- В контексте архитектуры и дизайна *вариант использования* (*use case*) – это описание ряда взаимодействий между системой и одним или более действующими лицами (либо пользователем, либо другой системой).
- *Сценарий* – это более широкое и всеобъемлющее описание взаимодействия пользователя с системой, чем ветвь варианта использования.
- Важные с точки зрения архитектуры варианты использования оказывают влияние на многие аспекты дизайна. Эти варианты использования важны для приемки развернутого приложения и должны охватывать достаточно большую часть дизайна, чтобы быть полезными при оценке архитектуры.

Бизнес-критический (Business Critical).

- Вариант использования, имеющий высокий уровень использования либо особую важность для пользователей или других заинтересованных сторон, по сравнению с другими функциями, или предполагающий высокий риск.

Имеющий большое влияние (High Impact).

- Вариант использования охватывает и функциональность, и параметры качества, либо представляет сквозную функцию, имеющую глобальное влияние на слои и уровни приложения. Примерами могут служить особо уязвимые с точки зрения безопасности операции Create, Read, Update, Delete (CRUD).

Ключевые сценарии

Наиболее важные сценарии для успеха создаваемого приложения	представляет проблемную область
	ссылается на существенный для архитектуры вариант использования
	представляет взаимодействие параметров качества с функциональностью
	представляет компромисс между параметрами качества

- Например, сценарии аутентификации пользователей могут быть ключевыми сценариями, потому что являются пересечением параметра качества (безопасность) с важной функциональностью (регистрация пользователя в системе).
- В качестве другого примера можно привести сценарий, основанный на незнакомой или новой технологии.

Общее представление приложения

позволит сделать архитектуру более осязаемой, свяжет ее с реальными ограничениями и решениями

1. Определение типа приложения.

- Прежде всего, определите, приложение какого типа создается. Будет ли это мобильное приложение, насыщенный клиент, насыщенное Интернет-приложение, сервис, Веб-приложение или некоторое сочетание этих типов?

2. Определение ограничений развертывания.

- При проектировании архитектуры приложения необходимо учесть корпоративные политики и процедуры, а также среду, в которой планируется развертывание приложения.

Если целевая среда фиксированная или негибкая, конструкция приложения должна отражать существующие в этой среде ограничения. Также в конструкции приложения должны быть учтены нефункциональные требования (Quality-of- Service, QoS), такие как безопасность и надежность. Иногда необходимо поступиться чем-либо в дизайне из-за ограничений в поддерживаемых протоколах или топологии сети. Выявление требований и ограничений, присутствующих между архитектурой приложения и архитектурой среды на ранних этапах проектирования позволяет выбрать соответствующую топологию развертывания и разрешить конфликты между приложением и целевой средой.

Общее представление приложения

Приложения часто используют сочетание стилей

3. Определение значащих архитектурных стилей проектирования.

- Определите, какие архитектурные стили будут использоваться при проектировании. Архитектурный стиль – это набор принципов. Он может рассматриваться как обобщенный шаблон, обеспечивающий абстрактную базу для семейства систем.

4. Выбор подходящих технологий.

- На основании типа приложения и других ограничений выбираем подходящие технологии и определяем, какие технологии будут использоваться в будущей системе. Основными факторами являются тип разрабатываемого приложения, предполагаемая топология развертывания приложения и предпочтительные архитектурные стили.

Каждый стиль определяет набор правил, задающих типы компонентов, которые могут использоваться для компоновки системы, типы отношений, применяемых в компоновке, ограничения по способам компоновки и допущения о семантике компоновки. Архитектурный стиль улучшает секционирование и способствует возможности повторного использования дизайна благодаря предоставлению решений часто встречающихся проблем. Типовыми архитектурными стилями являются сервисно-ориентированная архитектура, клиент/сервер, многослойная, шина сообщений и проектирование на основе предметной области.

Подходящие технологии

Мобильные приложения.

- Для разработки приложения для мобильных устройств могут использоваться технологии слоя представления, такие как .NET Compact Framework, ASP.NET для мобильных устройств и Silverlight для мобильных устройств.

Насыщенные клиентские приложения.

- Для разработки приложений с насыщенными UI, развертываемыми и выполняемыми на клиенте, могут использоваться сочетания технологий слоя представления Windows Presentation Foundation (WPF), Windows Forms и XAML Browser Application (XBAP).

Насыщенные клиентские Интернет-приложения (RIA).

- Для развертывания насыщенных UI в рамках Веб-браузера могут использоваться подключаемый модуль Silverlight™ или Silverlight в сочетании с AJAX.

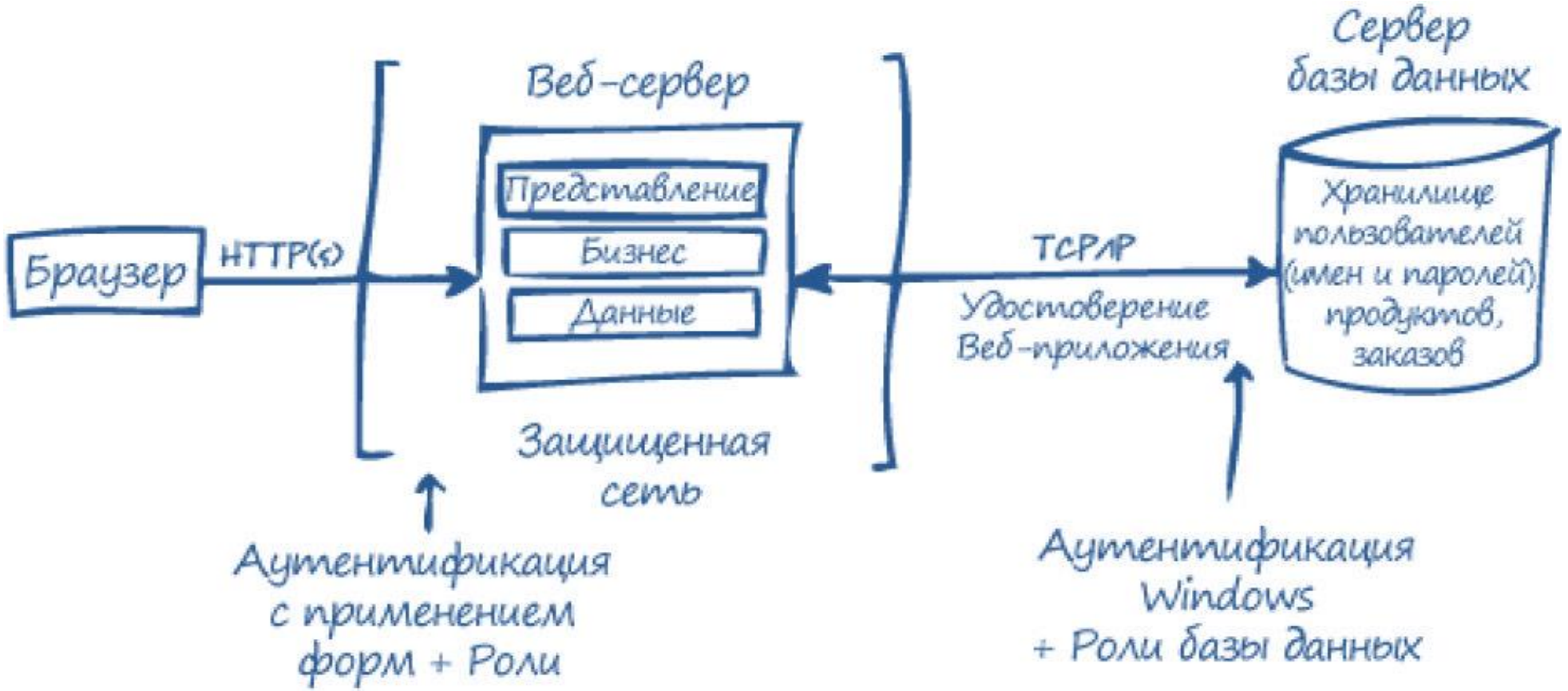
Веб-приложения.

- Для создания Веб-приложений могут применяться ASP.NET WebForms2, AJAX, Silverlight, ASP.NET MVC и ASP.NET Dynamic Data3.

Сервисные приложения.

- Для создания сервисов, предоставляющих функциональность внешним потребителям систем и сервисов, могут использоваться Windows Communication Foundation (WCF) и ASP.NET Web services (ASMX)

Графическое представление архитектуры



Основные проблемы

Определите **основные потенциальные проблемы** архитектуры своего приложения, чтобы понять области, в которых наиболее вероятно возникновение ошибок.

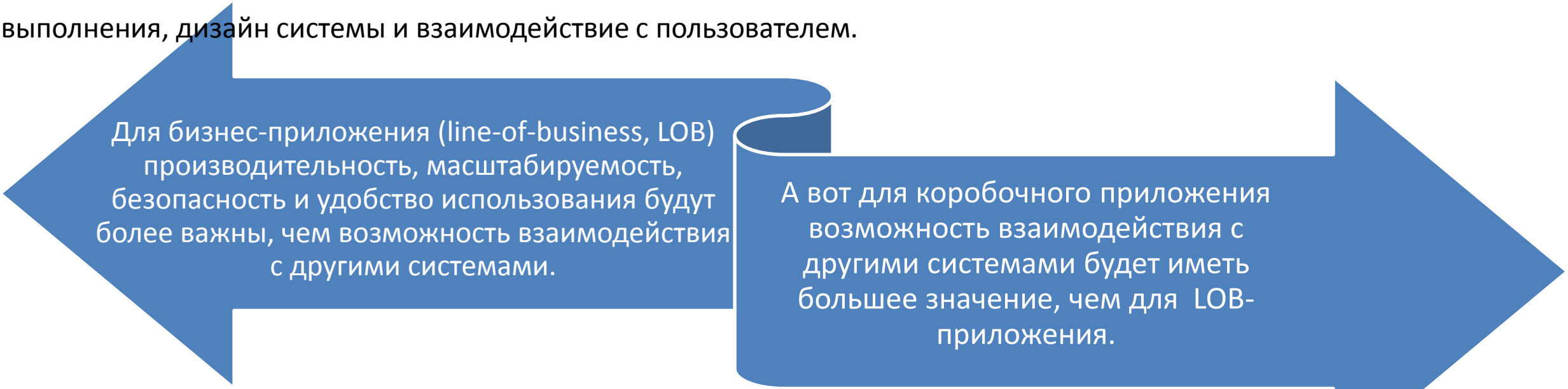
- К потенциальным проблемам относятся появление новых технологий и критически важные бизнес-требования.
- **Критически важные бизнес-требования** выявляются на стадии анализа требований заказчика.

- Несмотря на то, что это крайне обобщенные аспекты, как правило, при реализации они (и другие зоны риска) проецируются в *параметры качества* и *сквозную функциональность*.

Параметры качества

Та степень, с которой приложение обеспечивает требуемое сочетание параметров качества, таких как удобство и простота использования, производительность, надежность и безопасность, определяет успешность дизайна и общее качество продукта.

Параметры качества – это общие свойства архитектуры, которые оказывают влияние на поведение во время выполнения, дизайн системы и взаимодействие с пользователем.



- При проектировании приложения, отвечающего любому из параметров качеств, необходимо учесть влияние и других требований, должны быть проанализированы плюсы и минусы по отношению к другим параметрам качества. Важность или приоритетность каждого из параметров качества для разных систем разная.
- Параметры качества представляют функциональные области, которые потенциально могут оказывать влияние на все приложение, на все его слои и уровни. Некоторые параметры относятся ко всему дизайну системы, тогда как другие касаются только времени выполнения, времени проектирования или взаимодействия с пользователем.

Параметры качества

- Общие качества системы в целом, такие как возможность технической поддержки и тестируемость.

- Качества системы, проявляемые непосредственно во время выполнения, такие как доступность, возможность взаимодействия с другими системами, управляемость, производительность, надежность, масштабируемость и безопасность.

- Удобство и простота использования системы.

- Качества, отражающие дизайн системы, такие как концептуальная целостность, гибкость, удобство и простота обслуживания и возможность повторного использования.

Сквозная функциональность

это аспекты дизайна, которые могут применяться ко всем слоям, компонентам и уровням. Также это те области, в которых чаще всего делаются ошибки, имеющие большое влияние на дизайн.

- **Аутентификация и авторизация**. Как правильно выбрать стратегию аутентификации и авторизации, передачи идентификационных данных между слоями и уровнями и хранения удостоверений пользователей.
- **Кэширование**. Как правильно выбрать технику кэширования, определить данные, подлежащие кэшированию, где кэшировать данные и как выбрать подходящую политику истечения срока действия.
- **Связь**. Как правильно выбрать протоколы для связи между слоями и уровнями, обеспечения слабого связывания между слоями, осуществления асинхронного обмена данными и передачи конфиденциальных данных.
- **Управление конфигурацией**. Как выявить данные, которые должны быть настраиваемыми, где и как хранить данные конфигурации, как защищать конфиденциальные данные конфигурации и как обрабатывать их в серверной ферме или кластере.
- **Управление исключениями**. Как обрабатывать и протоколировать исключения и обеспечивать уведомления в случае необходимости.
- **Протоколирование и инструментирование**. Как выбрать данные, подлежащие протоколированию, как сделать протоколирование настраиваемым, и как определить необходимый уровень инструментирования.
- **Валидация**. Как определить, где и как проводить валидацию; как выбрать методики для проверки длины, диапазона, формата и типа; как предотвратить и отклонить ввод недопустимых значений; как очистить потенциально злонамеренный и опасный ввод; как определить и повторно использовать логику валидации на разных слоях и уровнях приложения.

Вопросы, требующие особого внимания при проектировании

- **Аудит и протоколирование**. Кто, что сделал и когда? Приложение функционирует в нормальном режиме? Аудит занимается вопросами регистрации событий, связанных с безопасностью. Протоколирование касается того, как приложение публикует данные о своей работе.
- **Аутентификация**. Кто вы? Аутентификация – это процесс, при котором одна сущность четко и однозначно идентифицирует другую сущность, обычно это делается с помощью таких учетных данных, как имя пользователя и пароль.
- **Авторизация**. Что вы можете делать? Авторизация определяет, как приложение управляет доступом к ресурсам и операциям.
- **Управление конфигурацией**. В каком контексте выполняется приложение? К каким базам данных подключается? Как выполняется администрирование приложения? Как защищены эти настройки? Управление конфигурацией определяет, как приложение реализует эти операции и задачи.
- **Шифрование**. Как реализована защита секретов (конфиденциальных данных)? Как осуществляется защита от несанкционированного доступа данных и библиотек (целостности)? Как передаются случайные значения, которые должны быть криптографически стойкими? Шифрование и криптография занимается вопросами реализации конфиденциальности и целостности.

Вопросы, требующие особого внимания при проектировании

- **Обработка исключений.** Что делает приложение при сбое вызова его метода? Насколько полные данные об ошибке оно предоставляет? Обеспечивает ли оно понятные для конечных пользователей сообщения об ошибках? Возвращает ли оно ценные сведения об исключении вызывающей стороне? Выполняется ли корректная обработка произошедшего сбоя? Предоставляет ли приложение администраторам необходимую информацию для проведения анализа основных причин сбоя? Обработка исключений касается того, как исключения обрабатываются в приложении.
- **Валидация входных данных.** Как определить, что поступаемые в приложение данные действительные и безопасные? Выполняется ли ограничение ввода через точки входа и кодировка вывода через точки выхода? Можно ли доверять таким источникам данных, как базы данных и общие файлы? Проверка ввода касается вопросов фильтрации, очистки или отклонения вводимых в приложение данных перед их дополнительной обработкой.
- **Конфиденциальные данные.** Как приложение работает с конфиденциальными данными? Обеспечивает ли оно защиту конфиденциальных данных пользователей и приложения? Здесь решаются вопросы обработки приложением любых данных, которые должны быть защищены либо при хранении в памяти, либо при передаче по сети, либо при хранении в постоянных хранилищах.
- **Управление сеансами.** Как приложение обрабатывает и защищает сеансы пользователей? Сеанс – это ряд взаимосвязанных взаимодействий пользователя и приложения.

Варианты решений

Возможный вариант архитектуры включает тип приложения, архитектуру развертывания, архитектурный стиль, выбранные технологии, параметры качества и сквозную функциональность.

Базовая архитектура

- описывает существующую систему, то как она выглядит сегодня. Для нового проекта исходная базовая архитектура – это первое высокоуровневое представление архитектуры, на основании которого будут создаваться возможные варианты архитектуры.

- ✓ На каждом этапе разработки дизайна надо понимать основные риски и предпринимать меры по их сокращению, проводить оптимизацию для эффективной и рациональной передачи проектных сведений и создаете архитектуру, обеспечивая гибкость и возможность реструктуризации. Возможно, архитектуру придется изменять несколько раз, использовать несколько итераций, возможных вариантов и множество пилотных архитектур.
- ✓ Итеративный и инкрементный подход позволяет избавиться от крупных рисков сначала, итеративно формировать архитектуру и через тестирование подтверждать, что каждая новая базовая архитектура является улучшением предыдущей.

Варианты решений

Пилотные архитектуры часто применяются в процессах гибкого или экстремального проектирования, но могут быть очень эффективным способом улучшения и доработки дизайна решения независимо от подхода к разработке.

Пилотная архитектура (*architectural spike*)

- это тестовая реализация небольшой части общего дизайна или архитектуры приложения. Назначение – анализ технических аспектов конкретной части решения для проверки технических допущений, выбора дизайна из ряда возможных вариантов и стратегий реализации или иногда оценка сроков реализации. Благодаря сфокусированности на основных частях общего проекта решения, пилотные архитектуры могут использоваться для решения важных технических проблем и для сокращения общих рисков и неопределенностей в дизайне.

Следующие вопросы помогут протестировать вариант «пилота» архитектуры:


Аутентификация пользователей
Обеспечение безопасной конфигурации
Предотвращение изменений параметров
Обработка исключений
Защита конфиденциальных данных
Браузер
Межсетевой экран
Веб-сервер
Веб-приложение
Сервер приложений
Приложения
Сервер базы данных
База данных
Защита конфиденциальных данных
Авторизация пользователей
Аудит, протоколирование и поддержка транзакции
Кодирование или хеширование конфиденциальных данных
Предотвращение перехватов сеансов и атак воспроизведения cookie
Валидация ввода
Аутентификация и авторизация входящих удостоверений

Анализ архитектуры

Анализ архитектуры приложения

- критически важная задача, поскольку позволяет сократить затраты на исправление ошибок, как можно раньше выявить и исправить возможные проблемы.

❖ Основная цель анализа архитектуры – подтверждение применимости базовой архитектуры и ее возможных вариантов, и также проверка соответствия предлагаемых технических решений функциональным требованиям и параметрам качества. Кроме того, анализ помогает обнаружить проблемы и выявить области, требующие доработки.

- ❑ Анализ архитектуры следует выполнять часто: по завершении основных этапов проекта и в ответ на существенные изменения в архитектуре.
- ❑ При создании архитектуры учитываются основные вопросы, задаваемых при таком анализе, это позволит как улучшить архитектуру, так и сократить время, затрачиваемое на каждый анализ.

Оценки на основании сценариев

основное внимание направлено на наиболее важные с точки зрения бизнеса и имеющие набольшее влияние на архитектуру сценарии

Представление дизайна архитектуры

Представление дизайна является очень важным для проведения анализа архитектуры, один из способов представления архитектуры – карта важных решений. Карта это не территория, а абстракция, которая помогает раскрыть и представить архитектуру.

4+1

- В данном подходе используется пять представлений готовой архитектуры. Четыре представления описывают архитектуру с разных точек зрения: логическое представление (например, объектная модель), представление процессов (например, аспекты параллелизма и синхронизации), физическое представление (схема программных уровней и функций в распределенной аппаратной среде) и представление для разработчиков. Пятое представление показывает сценарии и варианты использования ПО.

Гибкое моделирование

- Подход следует идее того, что содержимое важнее представления. Это обеспечивает простоту, понятность, достаточную точность и единообразие создаваемых моделей. Простота документа гарантирует активное участие заинтересованных сторон в моделировании артефактов.

IEEE 1471

- Сокращенное название стандарта, формально известного как ANSI/IEEE 1471-2000, который обогащает описание архитектуры, в частности, придавая конкретное значение контексту, представлениям и срезам.

UML

- Unified Modeling Language обеспечивает три представления модели системы. Представление функциональных требований (функциональные требования системы с точки зрения пользователя, включая варианты использования); статическое структурное представление (объекты, атрибуты, отношения и операции, включая диаграммы классов); и представление динамического поведения (взаимодействие объектов и изменения внутреннего состояния объектов, включая диаграммы последовательностей, деятельностей и состояний).

Проектирование многослойных приложений

Этапы проектирования многослойных приложений

- Приступая к проектированию приложения, прежде всего, сосредоточьтесь на самом высоком уровне абстракции и начинайте с группировки функциональности в слои. Далее следует определить открытый интерфейс для каждого слоя, который зависит от типа создаваемого приложения. Определив слои и интерфейсы, необходимо принять решение о том, как будет развертываться приложение.
- Наконец, выбираются протоколы связи для обеспечения взаимодействия между слоями и уровнями приложения. Несмотря на то, что разрабатываемая структура и интерфейсы могут изменяться со временем, особенно в случае применения гибкой разработки, следование этим этапам гарантированно обеспечит рассмотрение всех важных аспектов в начале процесса.

Обычно при проектировании используется следующая последовательность шагов:

Шаг 1 Выбор стратегии разделения на слои
Шаг 2 Выбор необходимых слоев
Шаг 3 Принятие решения о распределении слоев и компонентов
Шаг 4 Выяснение возможности сворачивания слоев
Шаг 5 Определение правил взаимодействия между слоями
Шаг 6 Определение сквозной функциональности
Шаг 7 Определение интерфейсов между слоями
Шаг 8 Выбор стратегии развертывания
Шаг 9 Выбор протоколов связи