Архитектурные представления и решения

Программная инженерия
Лекция 2

Михаил Пожидаев

Институт прикладной математики и компьютерных наук ТГУ

2 ноября 2026 г.

Архитектура программной системы

Зачем её описывать и оценивать?

Что такое архитектура?

Архитектура определяет наиболее существенные элементы системы, связи между ними и принципы их развития. Она влияет не только на структуру кода, но и на стоимость изменений, безопасность, производительность, надёжность и удобство эксплуатации.

Кто заинтересован в описании архитектуры?

- Бизнесу важно понимать границы системы, риски, стоимость и соответствие целям.
- Разработчикам важны компоненты, интерфейсы, зависимости и правила реализации.
- Администраторам важны развёртывание, наблюдаемость, отказоустойчивость и восстановление.
- Специалистам по безопасности важны границы доверия, потоки данных, механизмы защиты и аудит.

Главный принцип

Архитектура должна быть описана достаточно подробно для принятия решений, но не превращаться в неподдерживаемую коллекцию диаграмм.

Архитектурные представления

Одна система — несколько точек зрения

Архитектурное представление показывает систему с позиции конкретной группы заинтересованных сторон и её интересов. Например, логическое представление отвечает на вопрос о структуре модулей и их взаимодействии. Представление развёртывания отвечает на вопрос, где работают сервисы, базы данных и инфраструктурные компоненты. Представление безопасности показывает зоны доверия, механизмы аутентификации, авторизации и защиты данных.

Представление для бизнеса

- Показывает пользователей, бизнес-цели, ключевые процессы и внешние системы.
- Уровень деталей должен быть понятен без знания реализации.

Представление для команды разработки

- Показывает контейнеры, компоненты, API, хранилища и технологические зависимости.
- Оно помогает реализовывать и изменять систему согласованно.

Один и тот же сервис может присутствовать на нескольких диаграммах, но каждая диаграмма должна отвечать на свой конкретный вопрос.

C4 Model

Четыре уровня описания программной архитектуры

C4 Model — подход Саймона Брауна к визуализации архитектуры программных систем. Название C4 образовано от четырёх уровней абстракции: Context, Containers, Components и Code.

System Context

Показывает систему как единое целое в окружении пользователей и внешних систем.

Containers

Показывает крупные исполняемые или развёртываемые части системы.

Components

Показывает внутреннюю структуру одного контейнера.

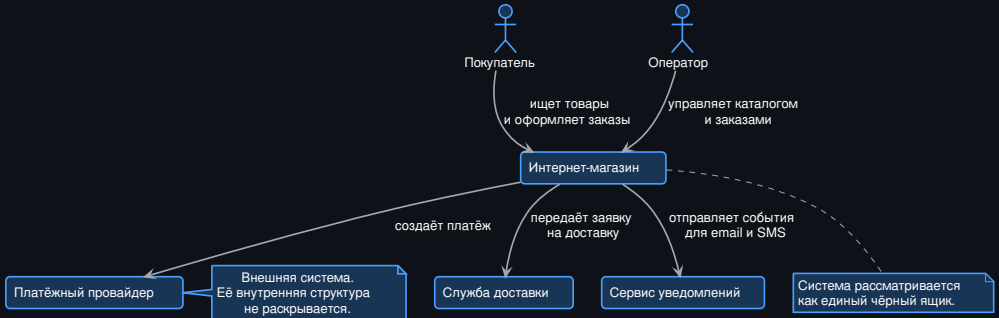
Code

Показывает классы, модули, пакеты или другие элементы исходного кода.

Термин container в C4 не равен Docker-контейнеру.

C4: System Context

Интернет-магазин в окружении пользователей и интеграций



C4: Container Diagram

Крупные части системы и технологии

Диаграмма контейнеров раскрывает внутренность системы на уровне приложений, сервисов и хранилищ. Она должна показывать ответственность каждого контейнера, используемые технологии и основные протоколы взаимодействия.

Примеры

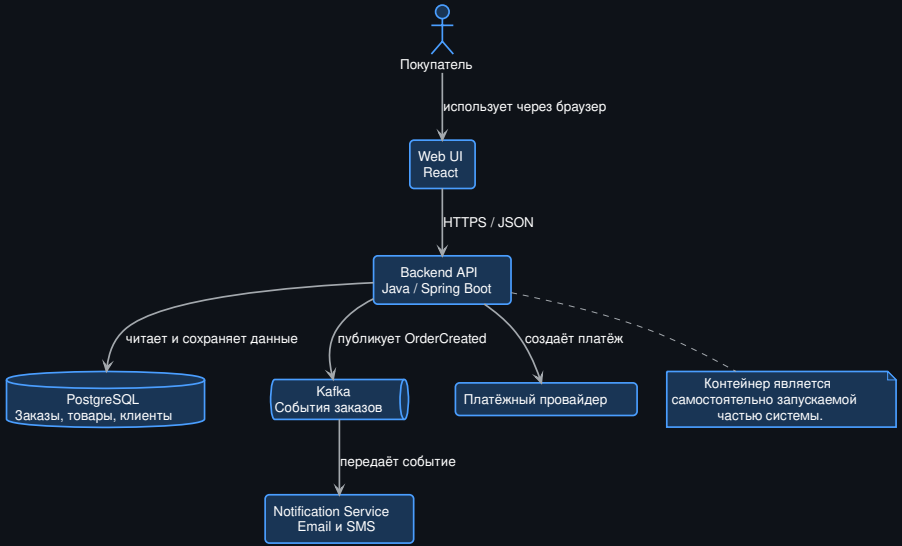
- **Web UI.** Одностраничное приложение для покупателей и операторов.
- **Backend API.** Обрабатывает заказы, каталог, пользователей и бизнес-правила.
- **PostgreSQL.** Хранит транзакционные данные о товарах, клиентах и заказах.
- **Message Broker.** Передаёт события о создании заказа и изменении статусов.
- **Notification Service.** Отправляет email, SMS или push-уведомления.

Смысл!

Для каждой связи важно указать смысл, а не только направление стрелки. Вместо «API использует БД» лучше написать «Backend API сохраняет заказы в PostgreSQL». Вместо «сервис связан с брокером» лучше написать «Backend API публикует событие OrderCreated в Kafka».

C4: Container Diagram

Контейнеры интернет-магазина



C4: Components и Code

Когда нужна детализация контейнера

Диаграмма компонентов применяется, когда одного уровня контейнеров недостаточно для понимания внутренней структуры приложения. Например, Backend API можно разделить на контроллеры, прикладные сервисы, доменную модель, репозитории и адаптеры внешних систем.

Component Diagram

- Подходит для обсуждения обязанностей крупных частей приложения.
- Помогает увидеть зависимость web-слоя от прикладной логики и инфраструктурных адаптеров.
- Полезна при онбординге разработчиков и выделении модулей.

Code Diagram

- Показывает классы, интерфейсы, пакеты или зависимости модулей.
- Нужна только для сложных или критичных участков.
- Часто заменяется исходным кодом, тестами и автоматически сгенерированной документацией.

Не нужно создавать диаграмму кода для каждого класса.

Архитектурная диаграмма ценна, пока она помогает принять решение или быстро понять систему.

ADR

Architecture Decision Record

Architecture Decision Record — короткий документ, фиксирующий важное архитектурное решение и причины его принятия. ADR сохраняет контекст, который обычно теряется после обсуждений в чатах, на встречах и в задачах трекера.

Типовая структура ADR

- **Title.** Краткое название решения.
- **Status.** Предлагается, принято, отменено, заменено или устарело.
- **Context.** Проблема, ограничения, требования и исходные условия.
- **Decision.** Что именно принято делать.
- **Alternatives.** Какие варианты были рассмотрены.
- **Consequences.** Положительные последствия, недостатки, риски и обязательства.
- **Links.** Ссылки на задачи, диаграммы, протоколы и связанные ADR.

Пример ADR

Выбор способа интеграции уведомлений

- **Title:** Асинхронная отправка уведомлений через брокер сообщений.
- **Status:** Принято.
- **Context:** Оформление заказа не должно зависеть от доступности email- и SMS-провайдеров. При синхронном вызове внешнего сервиса сбой провайдера увеличивает время ответа и может сорвать создание заказа.
- **Decision:** Backend API публикует событие OrderCreated в Kafka. Notification Service читает событие и отправляет уведомление независимо от основного пользовательского запроса.
- **Alternatives:** Синхронный REST-вызов Notification Service.
- **Consequences:** Время оформления заказа уменьшается и повышается устойчивость к сбоям провайдеров.

Накопление истории

Если решение в будущем изменится, предыдущий ADR не переписывают. Для нового решения создают отдельный ADR со статусом, например, «Supersedes ADR-003».

ATAM

Architecture Tradeoff Analysis Method

ATAM — метод оценки архитектуры через анализ рисков и компромиссов между атрибутами качества. Метод разработан Software Engineering Institute, SEI. ATAM не ищет «лучшую архитектуру вообще». Он проверяет, насколько конкретная архитектура соответствует приоритетным качественным требованиям заинтересованных сторон.

Основные шаги ATAM

1. Сформулировать бизнес-драйверы и архитектурные подходы.
2. Выявить и приоритизировать quality attribute scenarios.
3. Проанализировать архитектурные решения относительно сценариев.
4. Найти риски, точки чувствительности и tradeoff points.
5. Зафиксировать результаты и план снижения рисков.

В ATAM участвуют не только архитекторы.

В оценке полезно участие владельцев продукта, разработчиков, эксплуатации, безопасности и представителей пользователей.

ATAM: риски и компромиссы

Что выявляет архитектурная оценка

Sensitivity Point

Параметр архитектуры, существенно влияющий на характеристику качества. Например, размер пула соединений с базой данных влияет на пропускную способность и время ответа. Неправильное значение может создать очереди или перегрузить базу данных.

Tradeoff Point

Решение, одновременно влияющее на несколько характеристик качества. Например, кэширование повышает производительность, но усложняет консистентность данных. Репликация повышает доступность, но увеличивает стоимость и сложность эксплуатации.

Risk

Риск возникает, когда архитектурное решение может не удовлетворить важный сценарий качества. Например, один экземпляр базы данных является риском для требования доступности 99,95%.

Non-risk

Non-risk означает, что решение выглядит достаточным для рассматриваемого сценария при известных предположениях.

ATAM: пример анализа

Производительность, доступность и безопасность

- **Сценарий производительности:** в период распродажи 10 000 пользователей одновременно выполняют поиск товаров.
- **Мера реакции:** 95% запросов должны завершаться не более чем за 500 мс.
- **Архитектурный подход:** использовать поисковый индекс и кэширование популярных запросов.
- **Положительный эффект:** снижается нагрузка на основную базу данных и сокращается время ответа.
- **Компромисс:** данные в поиске могут обновляться с задержкой.
- **Сценарий доступности:** отказ одного экземпляра Backend API не должен прерывать оформление заказов.
- **Архитектурный подход:** несколько экземпляров API за балансировщиком нагрузки.
- **Риск:** общая база данных остаётся единой точкой отказа.
- **Сценарий безопасности:** доступ оператора к возврату средств должен требовать многофакторной аутентификации.
- **Компромисс:** повышается защита, но усложняется пользовательский сценарий и поддержка учётных записей.

Zachman Framework

Матрица полноты архитектурного описания

Zachman Framework — онтологическая схема классификации архитектурных артефактов предприятия. Она не предписывает процесс разработки и не заменяет проектную методологию. Она помогает проверить, какие аспекты предприятия и системы уже описаны, а какие были упущены.

Шесть вопросов Zachman Framework

1. **Что.** Данные, бизнес-сущности, документы и информационные объекты.
2. **Как.** Функции, процессы, сервисы и способы выполнения работы.
3. **Где.** Размещение, регионы, сети, каналы и точки взаимодействия.
4. **Кто.** Роли, организационные единицы, пользователи и ответственность.
5. **Когда.** События, расписания, жизненные циклы и временные ограничения.
6. **Почему.** Цели, правила, мотивация, стратегии и KPI.

Расширение круга интересов

Матрица Захмана особенно полезна в корпоративной архитектуре, где недостаточно описать только программные сервисы и базы данных.

Помимо шести вопросов, матрица Захмана содержит шесть перспектив.

1. **Планировщик.** Контекст предприятия и его границы.
2. **Владелец.** Бизнес-модель и требования к деятельности.
3. **Проектировщик.** Логическая модель системы и процессов.
4. **Разработчик.** Физическая реализация, технологии и схемы данных.
5. **Исполнитель.** Детальные спецификации, конфигурации и инструкции.
6. **Работающая система.** Фактическое состояние реализованного решения.

Пример ячеек матрицы

- На пересечении «Что» и перспективы владельца находятся бизнес-сущности, например заказ, клиент, товар и платёж.
- На пересечении «Как» и перспективы проектировщика находится логическая модель обработки заказа.
- На пересечении «Где» и перспективы разработчика может находиться схема размещения сервисов в кластере и сетевых сегментах.

ГОСТ Р 57100-2016 устанавливает основы описания архитектуры систем. Стандарт гармонизирован с ISO/IEC/IEEE 42010. Он не требует применять одну конкретную нотацию, язык моделирования или инструмент.

Ключевые понятия стандарта

- **Заинтересованная сторона.** Человек, группа или организация, имеющая интерес к системе.
- **Concern.** Значимый вопрос, требование, риск или проблема заинтересованной стороны.
- **Точка зрения.** Набор правил для создания представления архитектуры.
- **Архитектурное представление.** Описание системы с определённой точки зрения.
- **Модель.** Диаграмма, таблица, текст или иной артефакт внутри представления.
- **Обоснование архитектуры.** Причины выбора решений и связь с требованиями.

В рамках ГОСТ можно применять C4, UML, BPMN, ArchiMate, ER-диаграммы, сетевые схемы, ADR и текстовые спецификации.

Совместное применение подходов

Практический процесс работы с архитектурой

1. Определить заинтересованные стороны По ГОСТ Р 57100 выявить stakeholders и их concerns. Например, бизнесу важна скорость вывода функций, безопасности важна защита персональных данных, а эксплуатации важна восстанавливаемость.
2. Подготовить представления Использовать C4 для контекста, контейнеров и компонентов. Дополнить его диаграммами развёртывания, безопасности и потоков данных.
3. Зафиксировать решения Оформлять значимые решения в ADR. Указывать контекст, альтернативы и последствия, а не только итоговый выбор.
4. Провести оценку Применить ATAM к приоритетным сценариям качества. Зафиксировать риски, компромиссы, технический долг и план улучшений.

Zachman Framework можно использовать как контрольный список полноты.

Он помогает заметить, что команда описала технические сервисы, но не определила владельцев процессов, временные события или бизнес-цели.

Спасибо за внимание!

Домашняя страница:
https://marigostra.ru

Электронная почта:
msp@luwrain.org

Telegram:
@Marigostra

