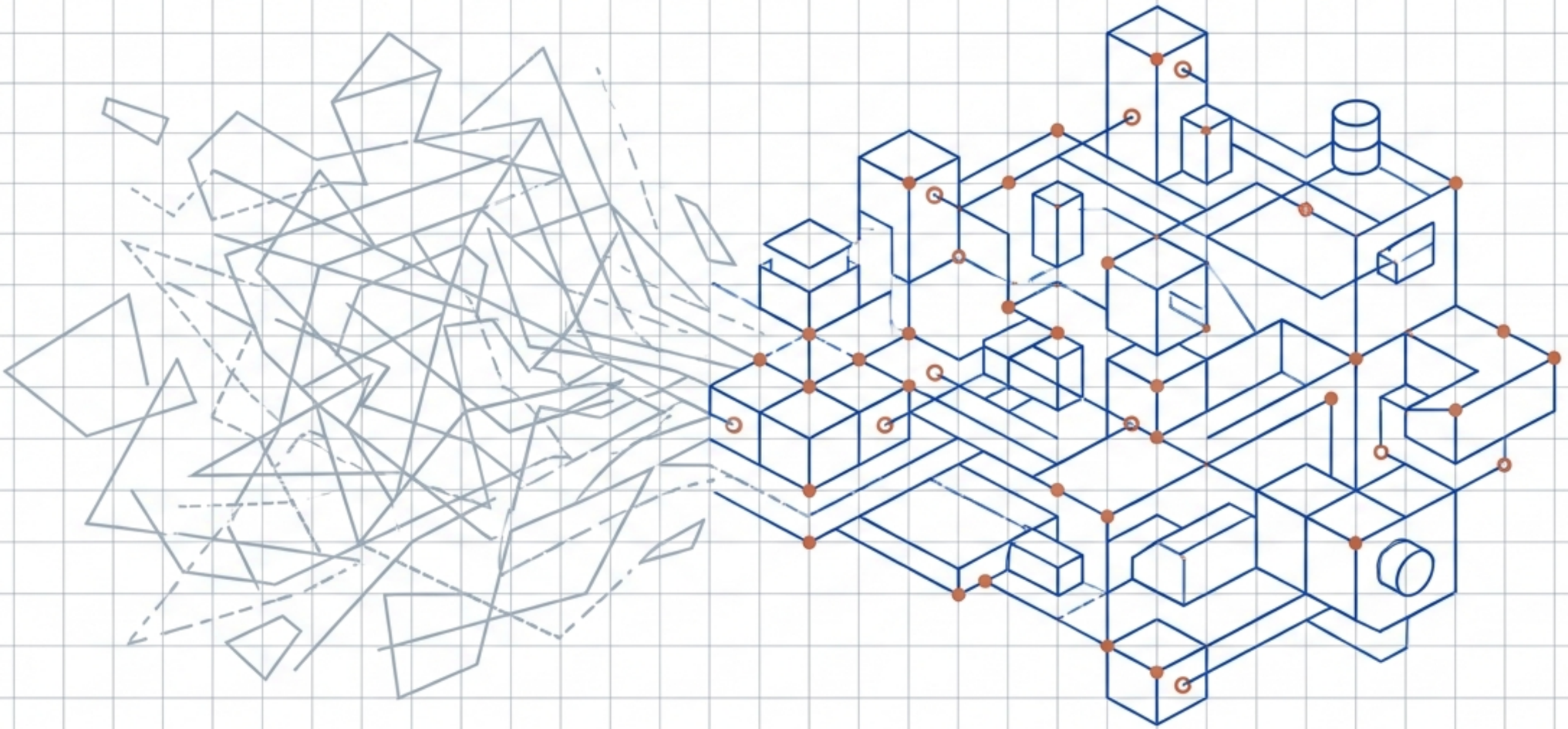
UML: Универсальный язык ИТ-архитектуры

От хаоса 90-х к современному подходу Diagram-as-Code

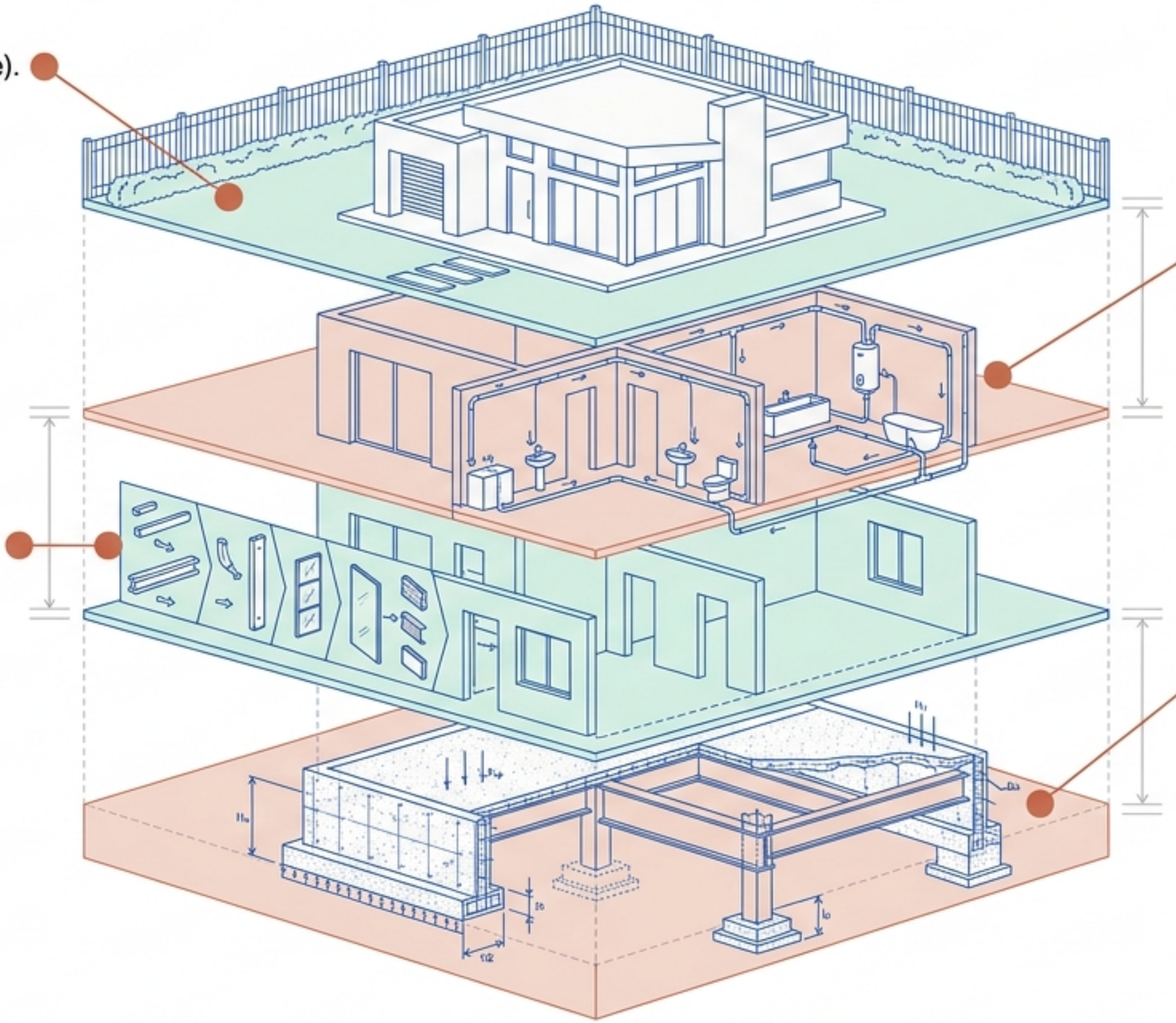
Руководство системного аналитика | OMG UML 2.5.1



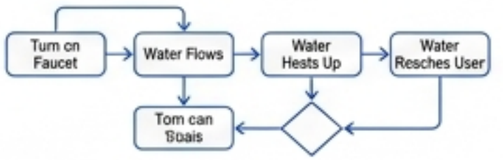
Архитектурный чертеж программного обеспечения

UML — это не язык программирования, это язык коммуникации. Без него разработчик строит дом по словесному описанию фотографии.

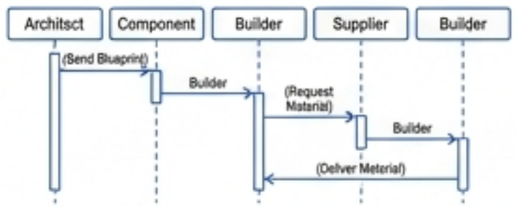
Уровень 1: План участка (Use Case). Показывает границы системы и кто имеет к ней доступ.



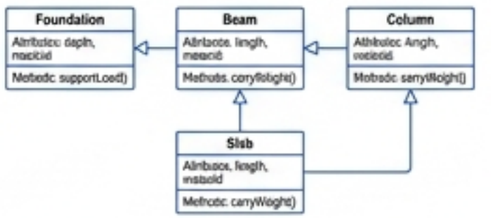
Уровень 2: Схема водопровода (Activity). Показывает, как протекают процессы внутри.



Уровень 3: Инструкция по сборке (Sequence). Шаг за шагом: кто, кому и когда передает детали.



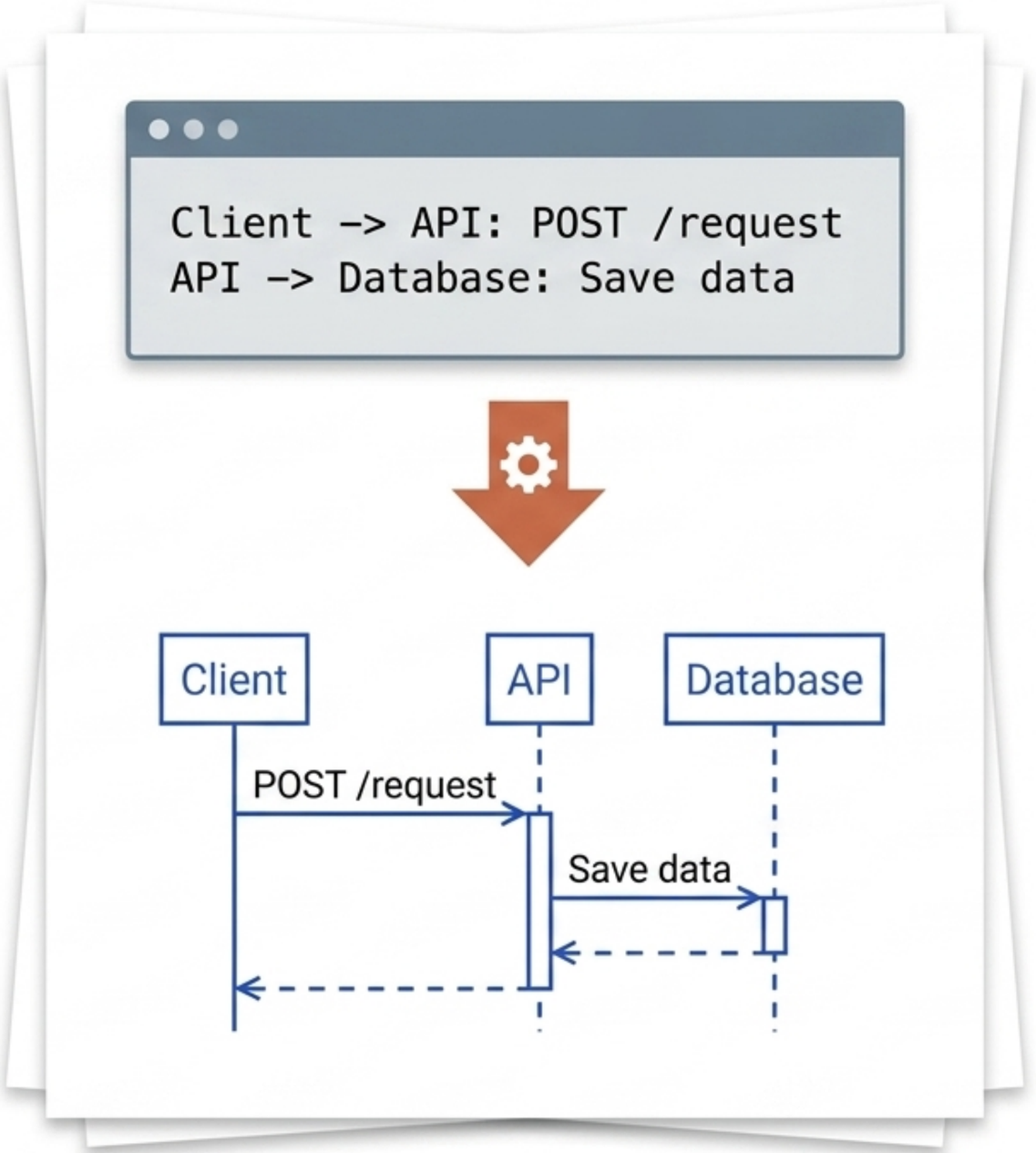
Уровень 4: Чертеж фундамента (Class). Структура, на которой держится дом.



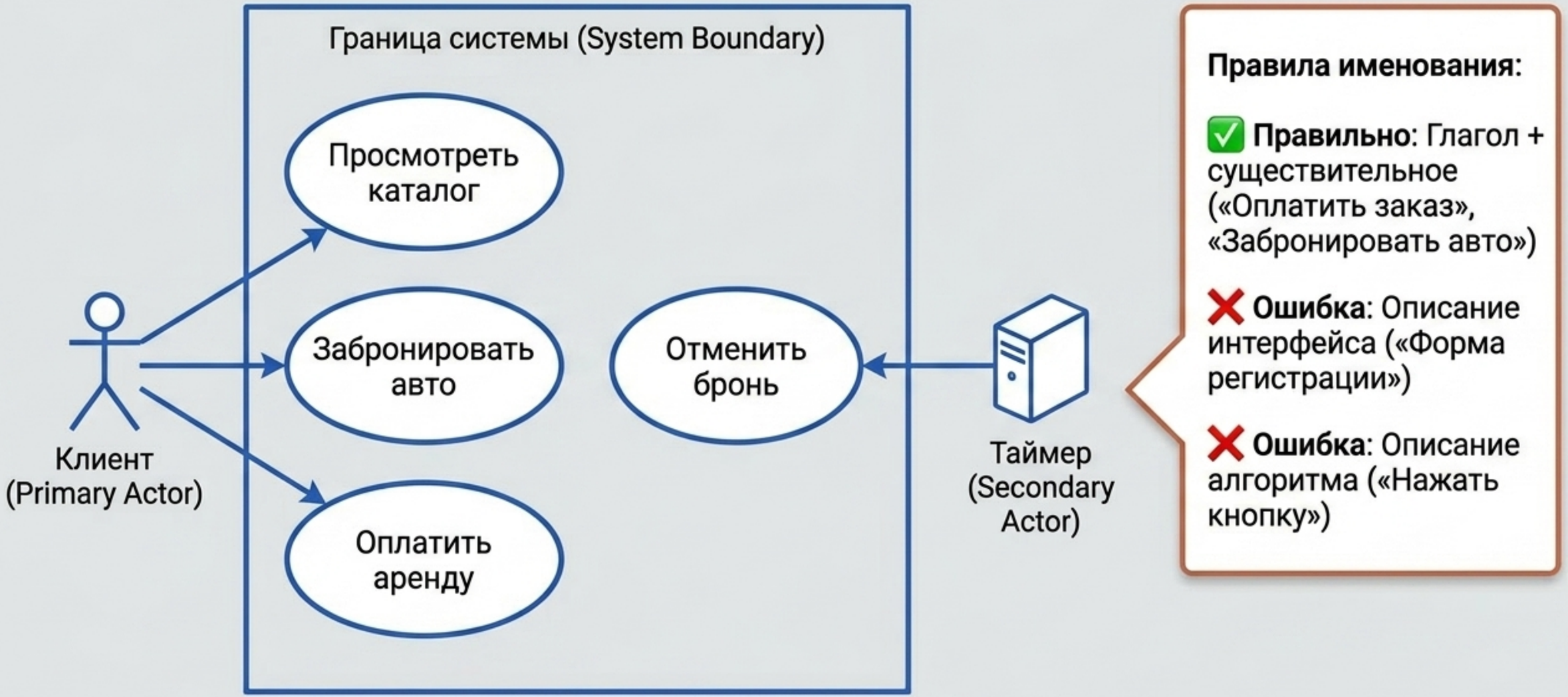
Инструментарий архитектора: Diagram as Code

UML — это не мертвые картинки. В Agile диаграммы живут в репозитории и обновляются вместе с кодом.

1	**Draw.io**	Быстрые наброски (Ad-hoc). Удобно, визуально, идеально для Confluence.
2	**Miro**	Воркшопы. Вся команда рисует одновременно на одной бесконечной доске.
3	**PlantUML**	Diagram as Code. Строгий стандарт, версионирование в Git, автоматизация.



Уровень 1: План участка (Use Case Diagram)



Правила именования:

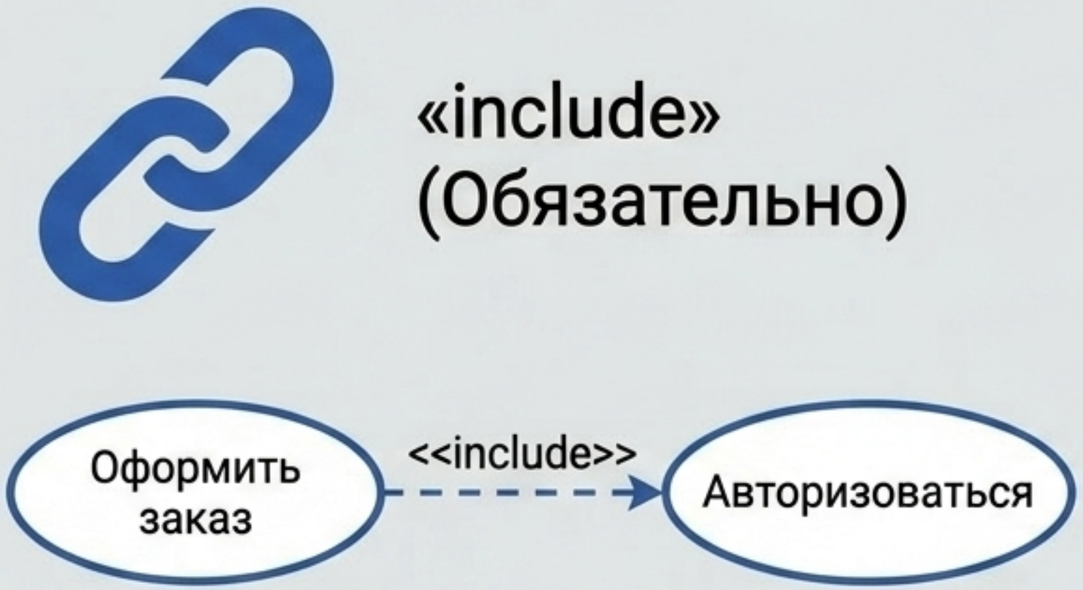
✅ **Правильно**: Глагол + существительное («Оплатить заказ», «Забронировать авто»)

❌ **Ошибка**: Описание интерфейса («Форма регистрации»)

❌ **Ошибка**: Описание алгоритма («Нажать кнопку»)

Шпаргалка архитектора: Включение vs Расширение

«include» (Обязательно)



Правило: Выполняется в 100% случаев. Базовый процесс невозможен без включенного.

Метафора: Вызов базовой функции в коде.

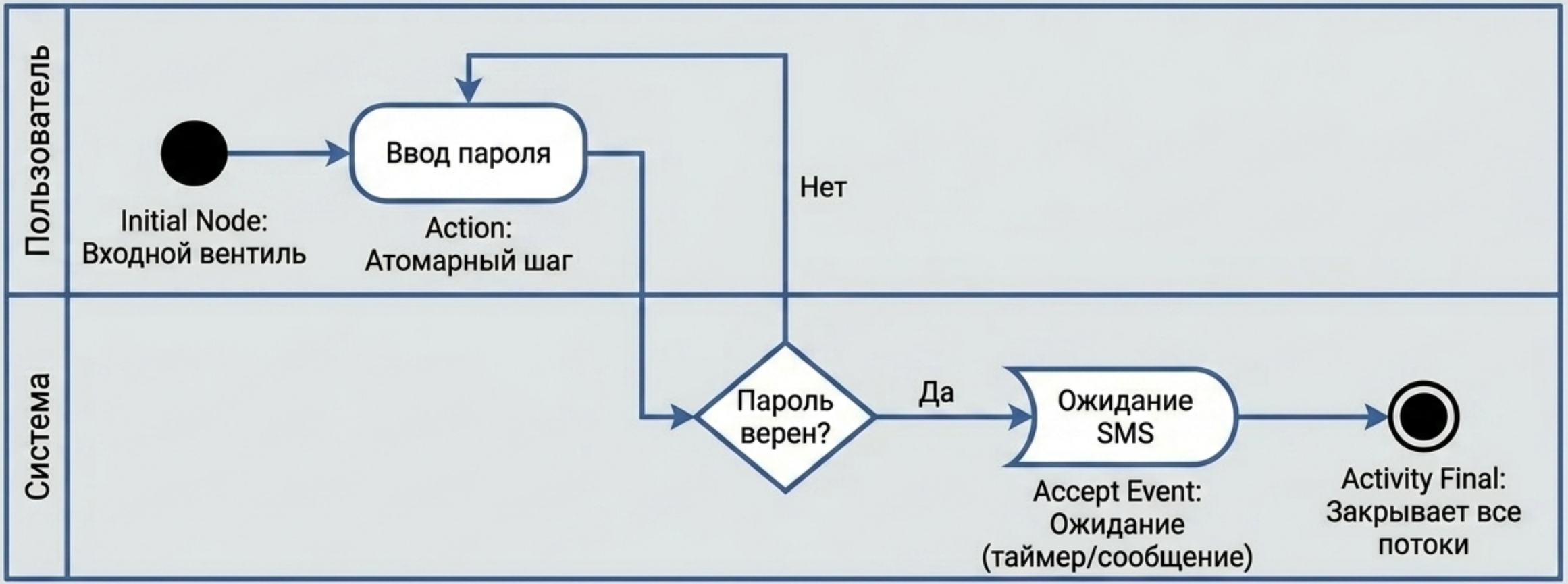
«extend» (Опционально)

Правило: Срабатывает только при определенном условии (Extension Point). Базовый процесс работает и без него.

Метафора: Блок if-условия, добавляющий фичу.

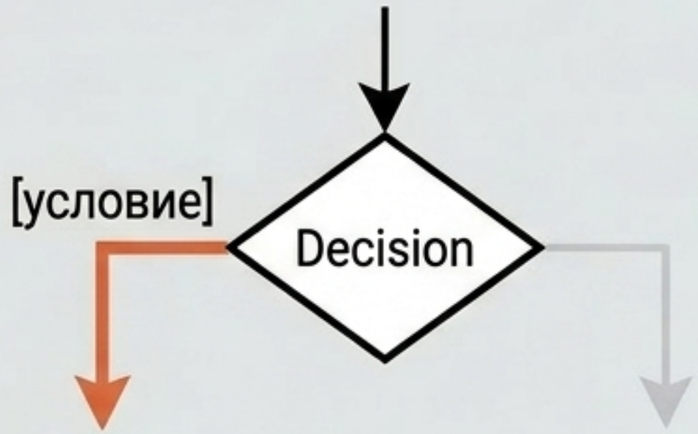
Уровень 2: Схема водопровода (Activity Diagram)

Идеально для сложных алгоритмов с циклами (например, логика 3FA или Retry). Архитектурный код процесса.



Шпаргалка архитектора: Ветвление vs Параллельность

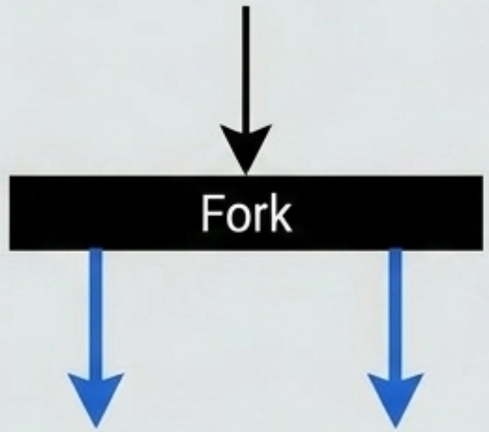
Узел Решения (Decision / Ромб)



Метафора: Перекресток (if-else).

Правило: Выбирается СТРОГО ОДИН путь. Обязательны условия [условие] и [else].

Узел Разветвления (Fork / Линия)



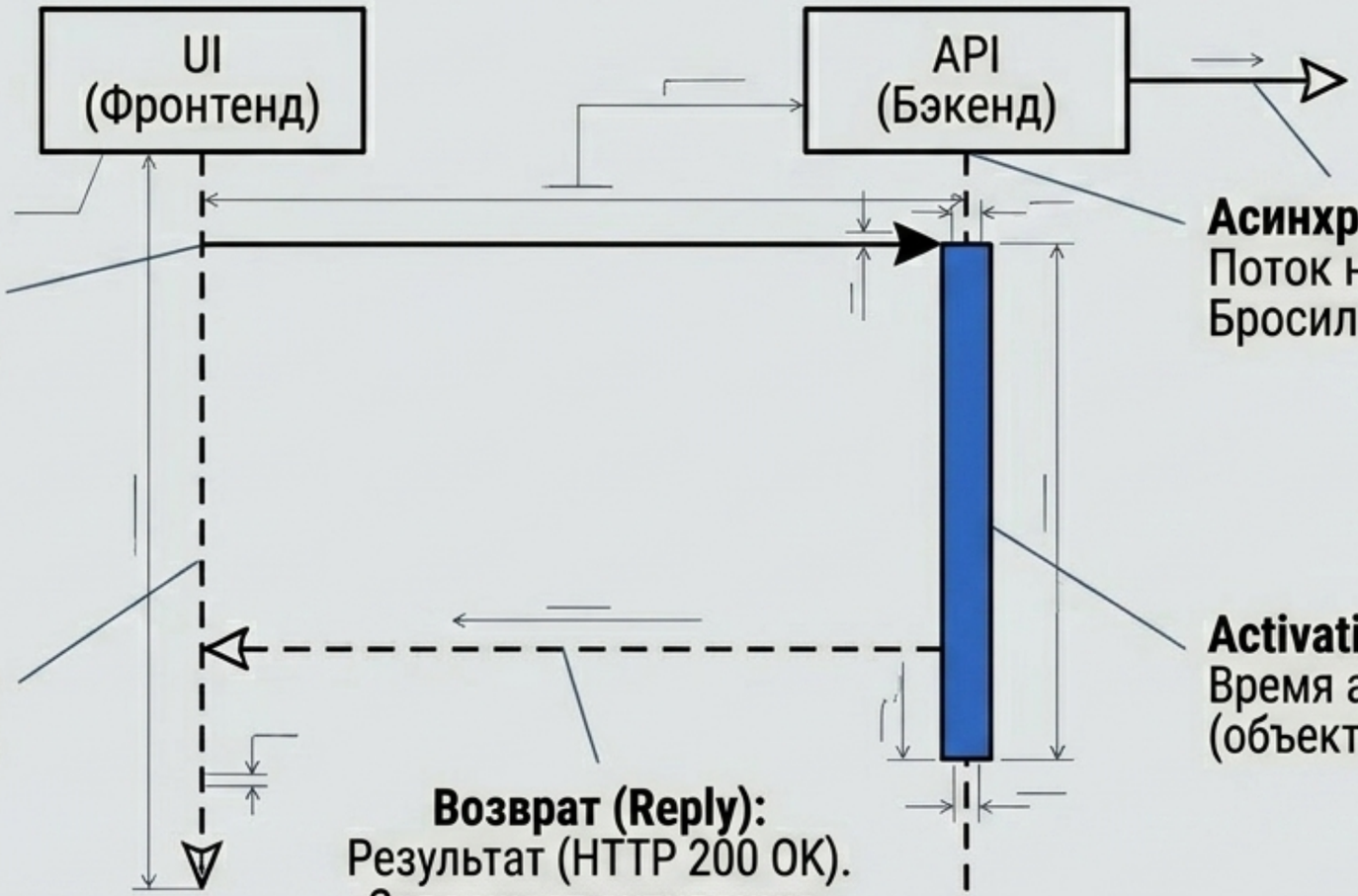
Метафора: Конвейер (async/threading).

Правило: Выполняются ВСЕ пути параллельно. Условий нет.

Внимание: Никогда не используйте Fork для выбора! Если потоки разошлись через Fork, они **должны сойтись через Join** (такую же черную линию).

Уровень 3: Инструкция по сборке (Sequence Diagram)

Главный контракт между Frontend и Backend.



Синхронное сообщение:
Поток блокируется.
Ждем ответ (HTTP POST).

Асинхронное сообщение:
Поток не блокируется.
Бросили в Kafka и забыли.

Lifeline: Время течет строго сверху вниз.

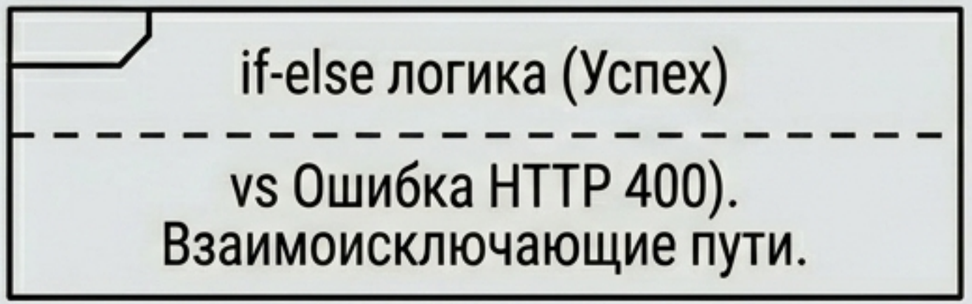
Activation Bar:
Время активности
(объект работает или ждет)

Возврат (Reply):
Результат (HTTP 200 OK).
Завершает активацию.

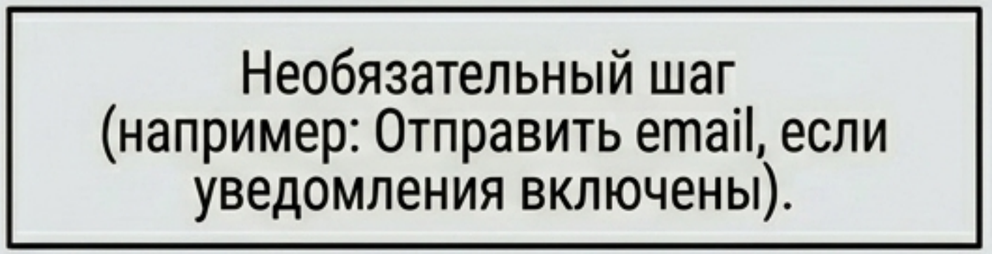
Шпаргалка архитектора: Фрагменты логики (Combined Fragments)

Каждый alt, opt или break на диаграмме — это готовый отдельный тест-кейс для QA.

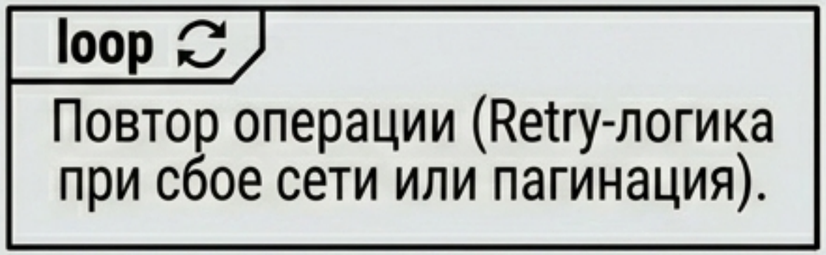
alt (Альтернатива)



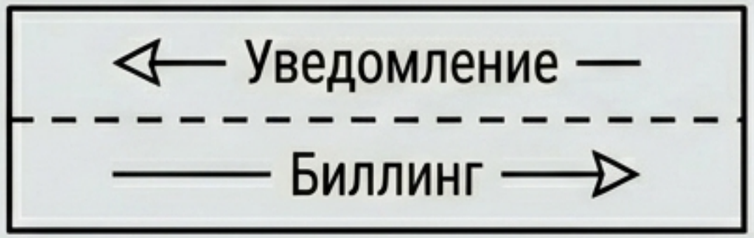
opt (Опция)



loop (Цикл)



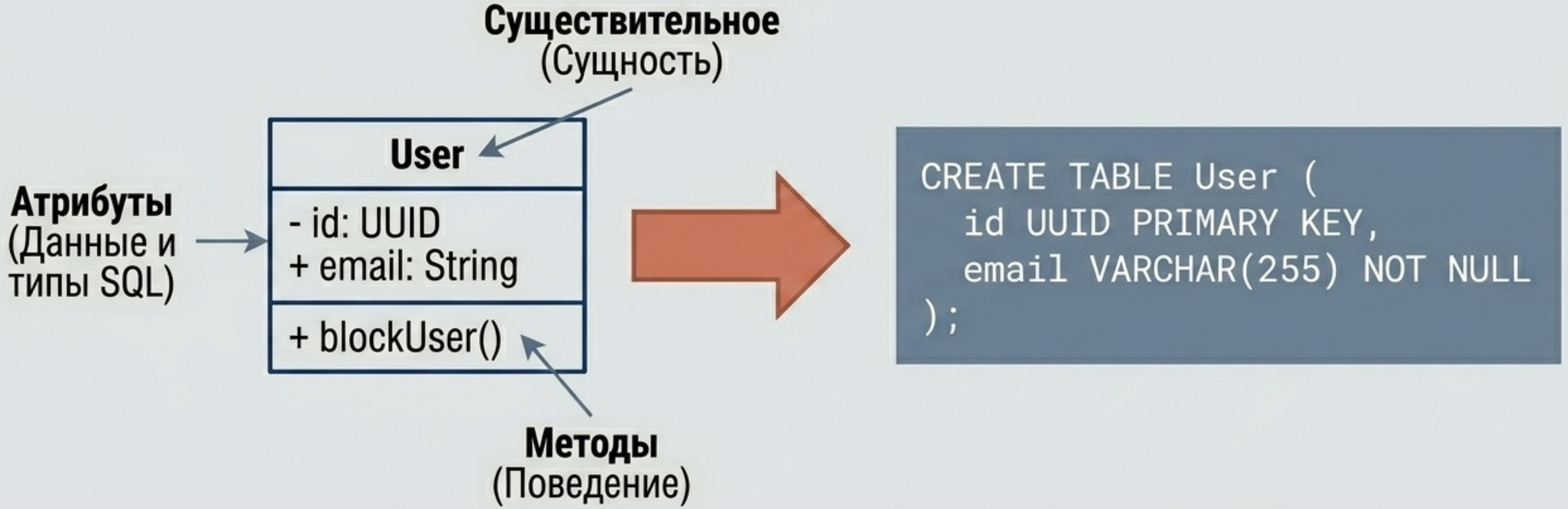
par (Параллельно)



Параллельные вызовы (например: Одновременное уведомление и биллинг).

Уровень 4: Чертеж фундамента (Class Diagram)

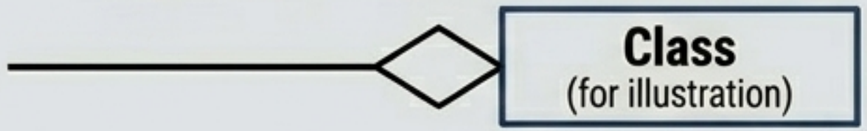
Для аналитика классы выявляют сущности. Ищите существительные в требованиях заказчика.



Уровни видимости: (+) public — попадет в API, (-) private — скрытая внутренняя логика.

Шпаргалка архитектора: Анатомия структурных связей

Агрегация (Пустой ромб)



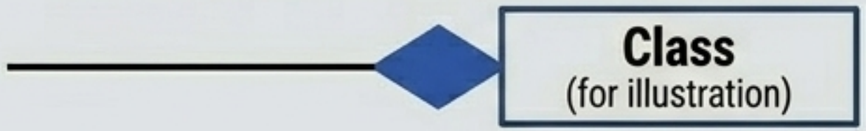
Метафора: Университет и Студенты.
Правило: Часть выживет без целого. Слабая связь.

Трансляция в SQL:

```
FOREIGN KEY ...
ON DELETE SET NULL
```

Если университет закроют, студенты останутся (перейдут в другой).

Композиция (Залитый ромб)



Метафора: Дом и Комнаты.
Правило: Часть умрет вместе с целым. Сильная связь.

Трансляция в SQL:

```
FOREIGN KEY ...
 ON DELETE CASCADE
```

Если дом снесут, комнат больше не существует.
Пример: Заказ и Позиция заказа.

Синтез: Единая цифровая голограмма (Кейс: Каршеринг)

Как описать фичу «Начало аренды авто»?

1. Use Case
Заказчик спросит: Кто это делает и каковы границы системы?

2. Activity
Бизнес спросит: Каков пошаговый процесс проверки перед запуском?

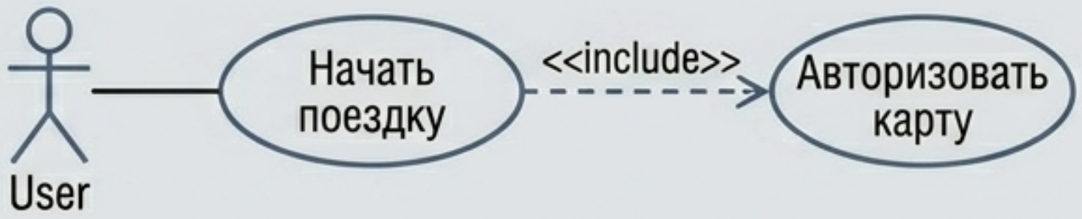
3. Sequence
Разработчик спросит: В каком порядке идут HTTP-вызовы к автомобилю?

4. Class
Архитектор БД спросит: Где и как сохранятся данные поездки?

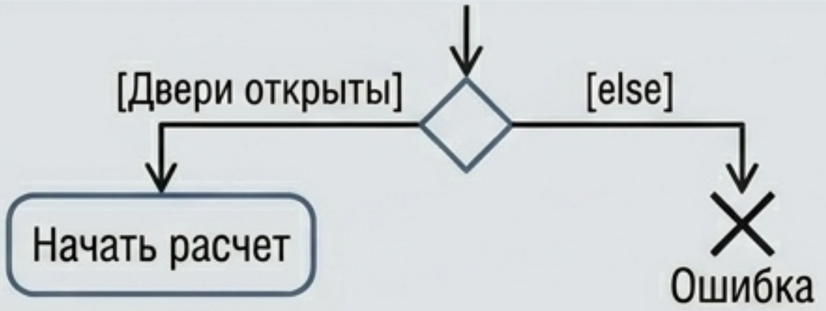
Четыре диаграммы — это просто разные проекции одного и того же куска кода.

Четыре линзы одного куска кода: Начало аренды

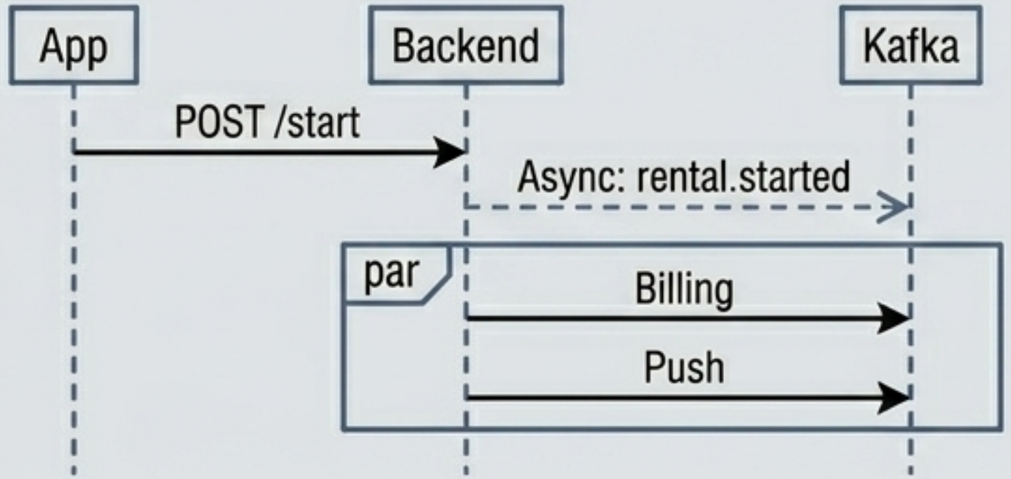
1. **Цель** (Use Case)



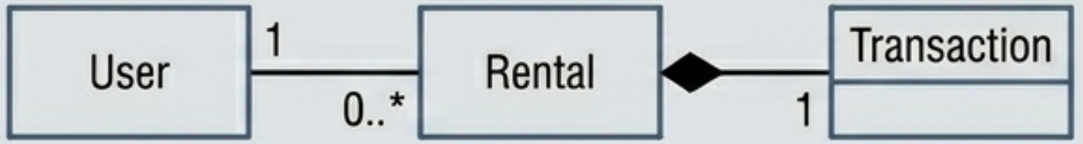
2. **Логика** (Activity)



3. **Контракт** (Sequence)



4. **Память** (Class)



UML Light: Инструмент мышления, а не бюрократии

Точно в срок (Just-in-Time)

Не рисуйте все 14 диаграмм Waterfall-стилем. Рисуйте только 5 основных (Use Case, Activity, Sequence, Class, State) перед самым началом спринта, чтобы устранить слепые зоны.

Diagram as Code

Храните UML в репозиториях (Git) рядом с кодом через PlantUML. Схемы должны эволюционировать вместе с продуктом, а не пылиться в файлах Word.

Цель — Коммуникация

Диаграмма достигла цели, если разработчик, тестировщик и бизнес одинаково поняли задачу. Эстетика вторична, техническая точность и ясность первичны.