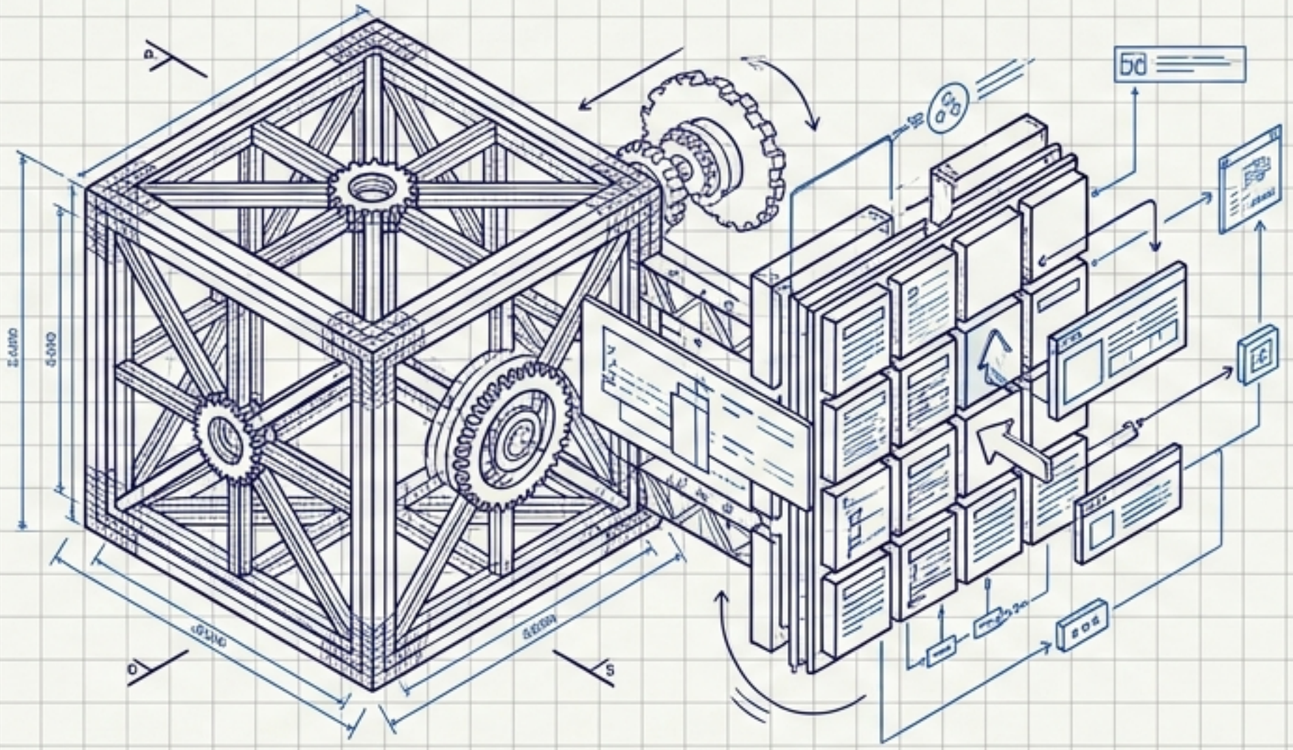
PROJ. REV. 2.1

DOC. NO. REQ-ARCH-001

Архитектура требований

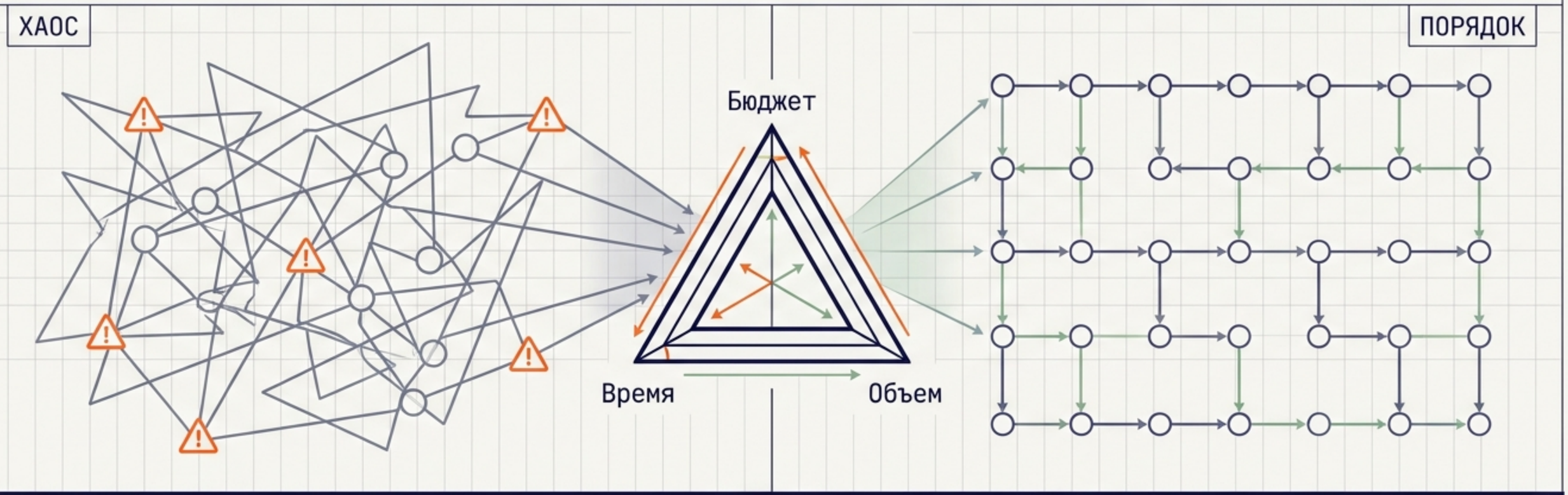


Практический инструментарий системного аналитика: от эмпатии User Stories до строгости матриц трассировки.

PROJ. REV. 2.1

DOC. NO. REQ-ARCH-001

Иллюзия «Важно абсолютно всё»



Любой проект зажат в тисках тройственного ограничения. Когда заказчик утверждает, что все требования критичны, он перекладывает ответственность за приоритизацию на хаос разработки.

- Невнятные требования ведут к расползанию границ (scope creep).
- Срыв сроков возникает из-за позднего выявления зависимостей.
- Бюджет сгорает на разработку функций без бизнес-ценности.

Решение — строгий инженерный конвейер перевода абстрактных желаний в проверяемые артефакты.

Формула Connextra: Обещание будущего диалога

User Story — это не спецификация.
Это триггер для обсуждения,
фиксирующий, что нужно сделать, кому
и, главное, зачем.

[Роль] + [Действие] = [Ценность]

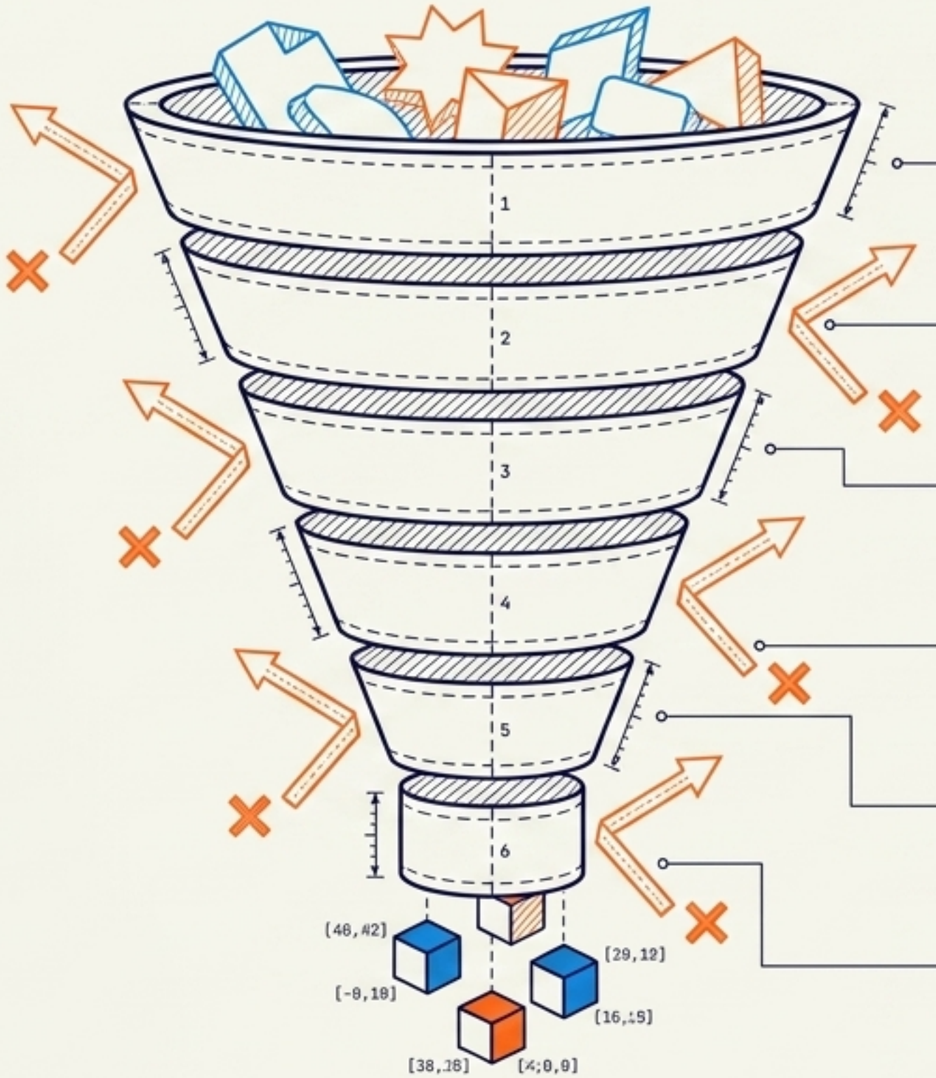
Как [Роль], я хочу [Действие], чтобы [Ценность]

Пример: Разные реализации от одного действия

Вывод: Отсутствие «чтобы» превращает историю в нерешаемую загадку.

Диагностический фильтр INVEST

Качественная User Story должна пройти 6 уровней фильтрации перед попаданием в спринт.



[I] Independent
Независимая. Не ждет завершения других историй.

[N] Negotiable
Обсуждаемая. Оставляет команде пространство для технических решений.

[V] Valuable
Ценная. Решает бизнес-задачу пользователя, а не просто обновляет БД.

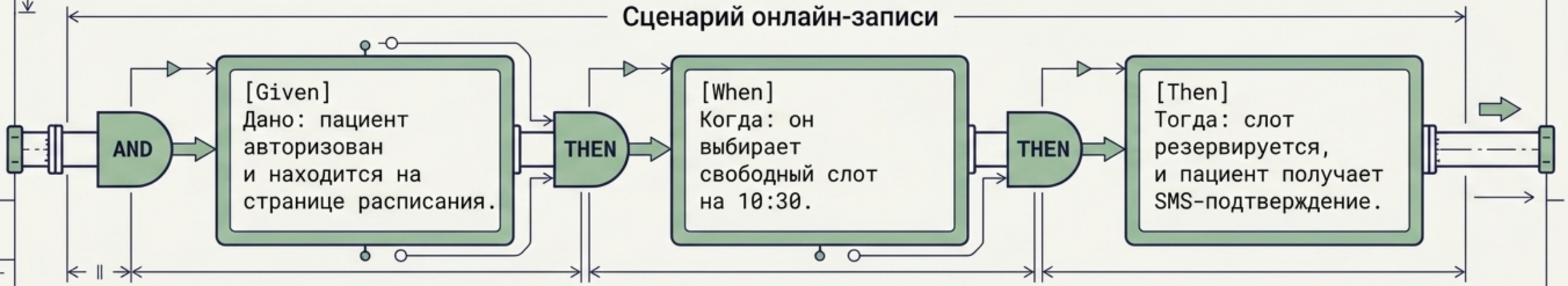
[E] Estimable
Оцениваемая. Команда понимает объем и может дать оценку в Story Points.

[S] Small
Маленькая. Гарантированно помещается в один спринт (иначе это Epic).

[T] Testable
Тестируемая. Имеет четкие численные критерии приемки.

Синтаксис Gherkin: Логические вентили приемки

Acceptance Criteria — это Definition of Done. Gherkin переводит их в сценарный формат, исключающий двусмысленность.



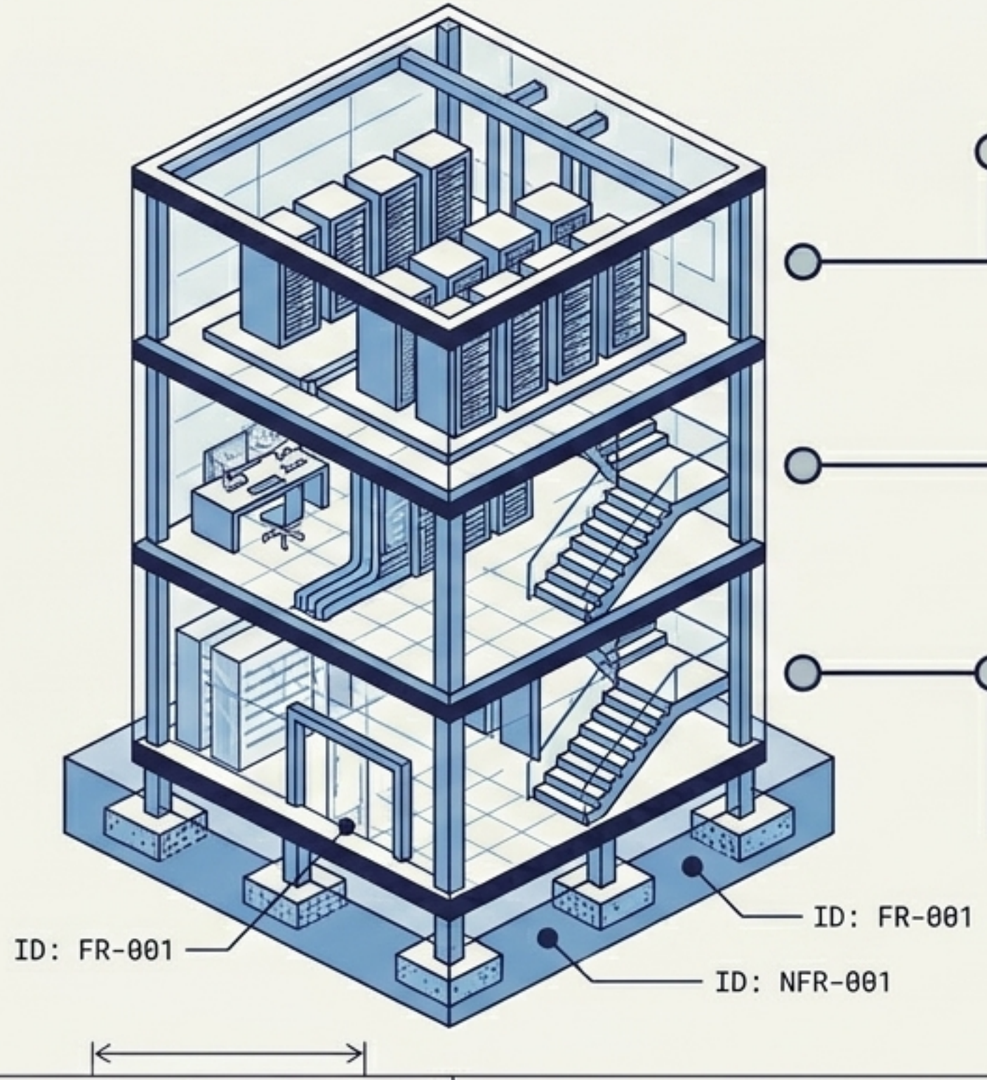
Анатомия покрытия одной истории

Трансформация: От Agile-диалога к формальному контракту

User Story	Functional Requirement (SRS)
Суть Обещание для обсуждения	Жесткая спецификация
Фокус Ценность для пользователя («Зачем?»)	Поведение системы («Что и как?»)
Жизненный цикл Живет до конца спринта	Базовый документ, живет годами
Создатель Команда (PO + аналитик + разработчик)	Системный аналитик / Архитектор
Уровень деталей Оставляет пространство для маневра	Исключает любую двусмысленность

Архитектура спецификации (IEEE 830)

SRS — это фундамент, на котором проектируется, разрабатывается и тестируется система.



Введение (Scope & Vision)

Границы системы. Четко определяет, что входит в проект, а что сознательно исключено (Out of Scope) для защиты от разрастания требований.

Общее описание (Context)

Взаимодействие с внешними системами, потоки данных, характеристики пользователей и жесткие ограничения (бюджет, стек, железо).

Специфические требования

Ядро документа.
Функциональные требования (FR) — что система делает.
Нефункциональные (NFR) — как она это делает (производительность, безопасность, доступность).

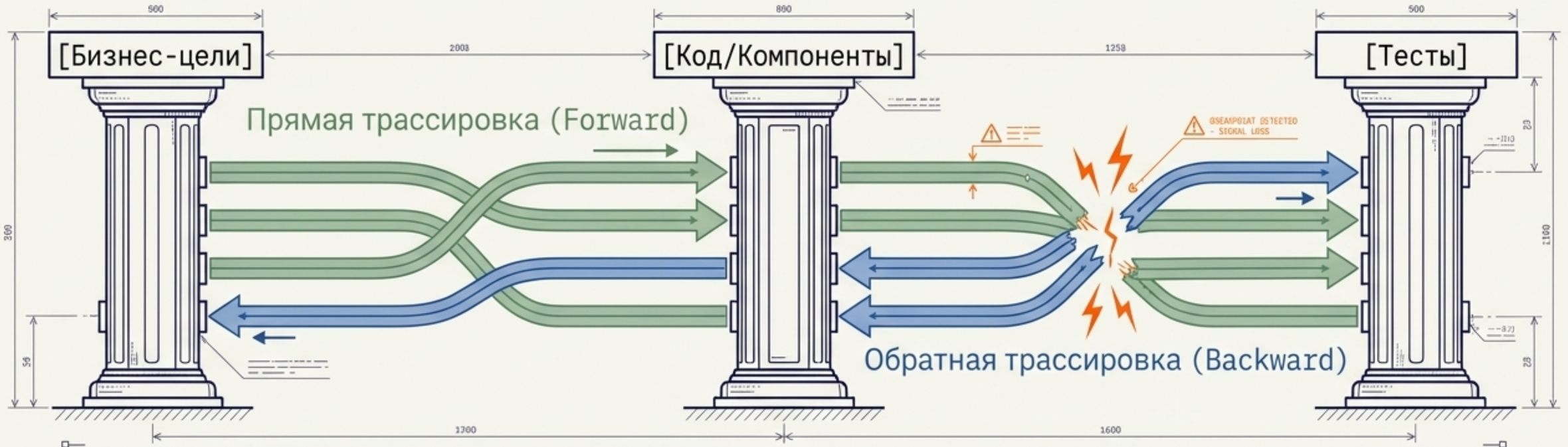
Лексикон двусмысленности: Рефакторинг бизнес-языка

Главный враг SRS — слова-паразиты. Если требование нельзя проверить автоматизированным тестом, это пожелание.

Красные флаги (Субъективно)	**Зеленые флаги** (Инженерные метрики)
[✗] Система должна работать ~~быстро~~	[✓] NFR: Система должна загружать отчет на 1000 строк за < 3 сек
[✗] Интерфейс должен быть ~~удобным~~	[✓] NFR: Сценарий оформления заказа выполняется максимум за 3 клика
[✗] Данные ~~надежно~~ защищены	[✓] NFR: RTO ≤ 1 час, RPO ≤ 15 минут, пароли хешируются bcrypt
[✗] ~~Заказ будет создан~~ Пассивный залог	[✓] FR: Система должна создать заказ после подтверждения оплаты

Трассировка: Кровеносная система проекта

Матрица RTM связывает бизнес-цели с конкретными строками кода и тест-кейсами.



Forward	Backward
Исток ➔ Реализация. Доказывает 100% покрытие. Ответ на вопрос: «Мы сделали всё, что просил заказчик?»	Тест ➔ Бизнес-цель. Обосновывает ценность. Выявляет gold-plating (избыточный код, который не решает ни одной бизнес-задачи).

Если нить оборвана: Код написан зря (нет бизнес-цели), или функционал уязвим (нет тест-кейса).

Анатомия матрицы трассировки

RTM — это не бумажный отчет, а **живой дашборд** контроля над разработкой.
Каждая строка содержит критические метаданные.

Анализ влияния (Blast Radius): Пятничный Change Request

Вводная:

Пятница, вечер. Заказчик просит изменить логику создания заказа (FR-002), разрешив покупку гостям без авторизации.

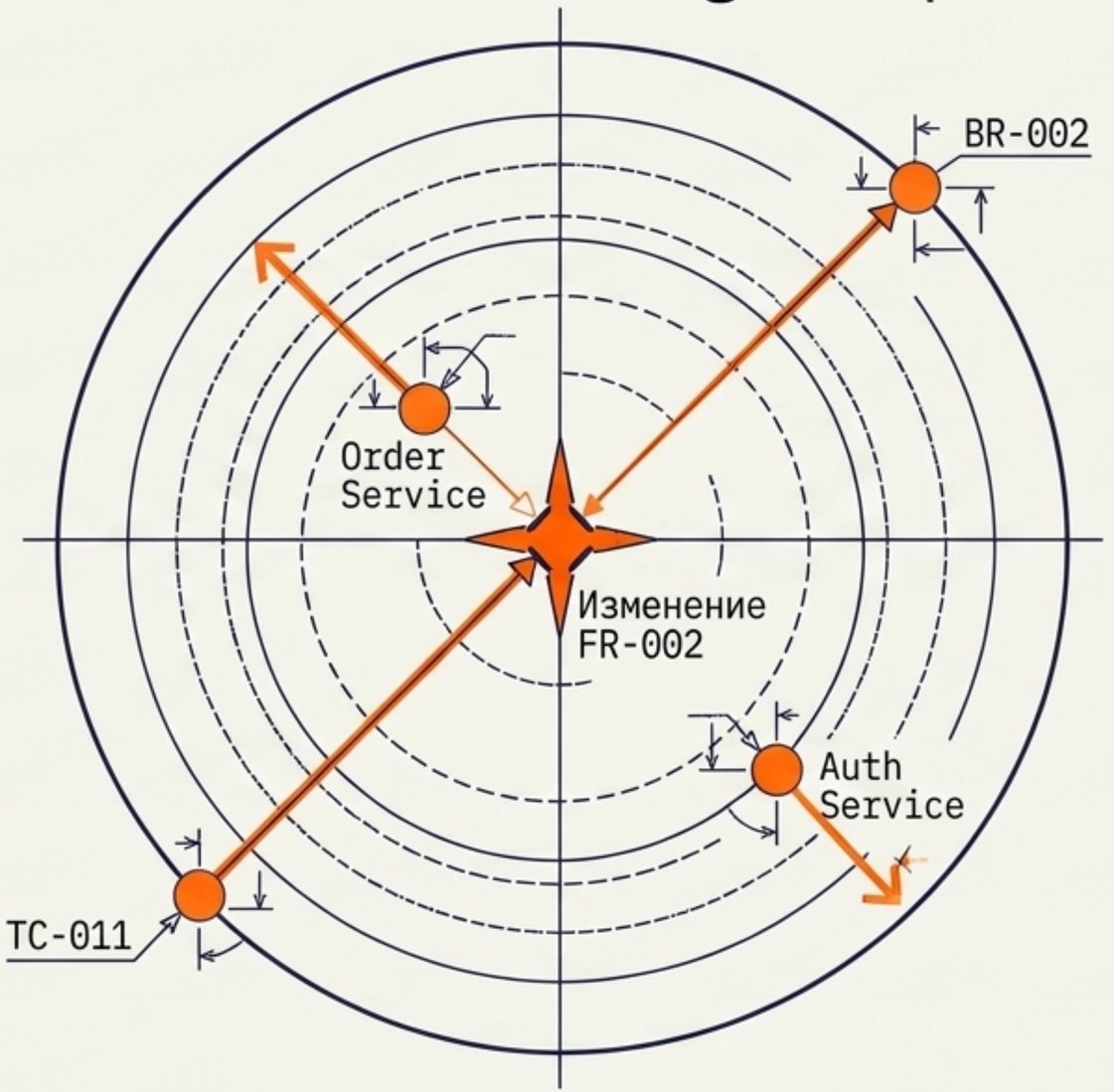
Анализ:

Без RTM команда анализирует код днями. С RTM аналитик за 15 минут определяет радиус поражения:

- **Компоненты:** Order, Notification, Auth Service.
- **Упавшие тесты:** Нужно переписать TC-010, TC-011, TC-020.
- **Бизнес-цели:** Конфликтует с BR-002 (прозрачность статуса).

Вывод:

RTM превращает панику в точный инженерный расчет стоимости изменений.



Стратегическая приоритизация: Матрица MoSCoW

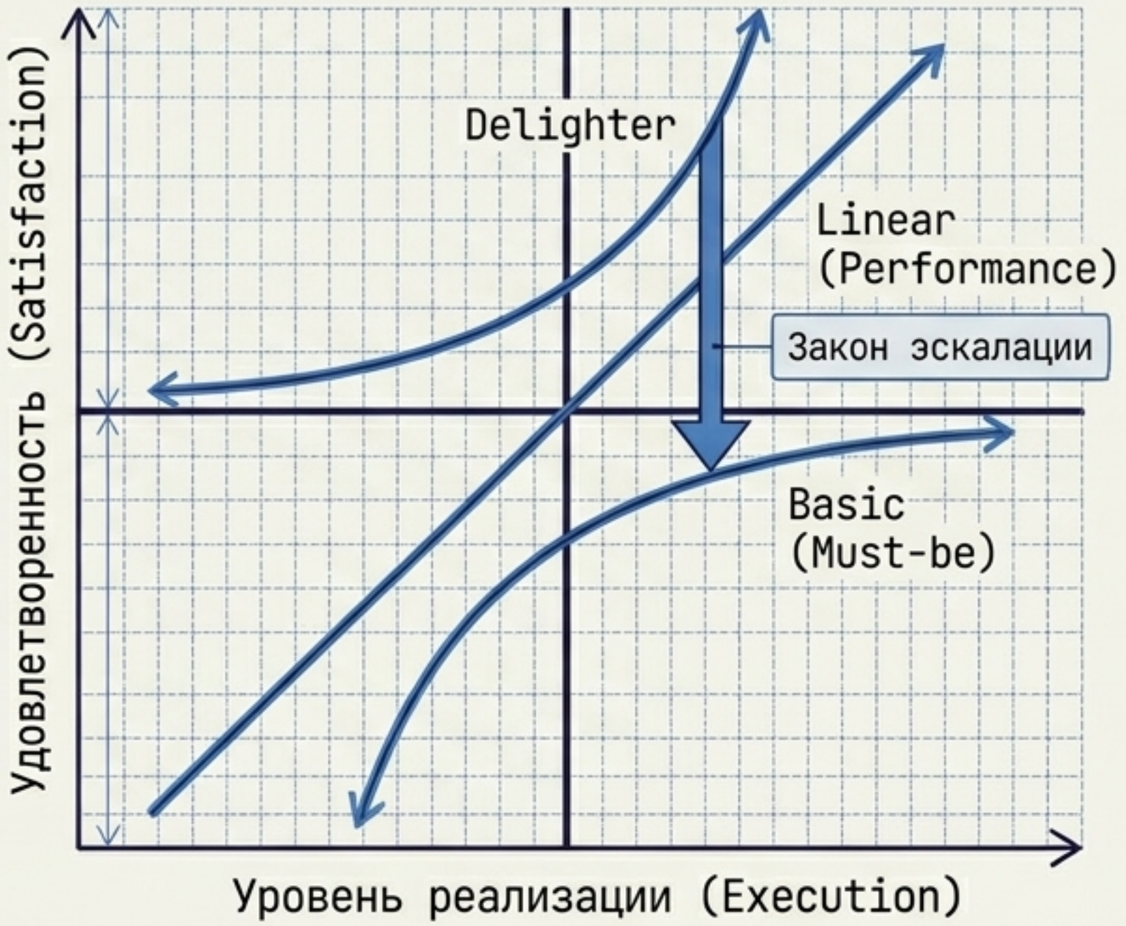
В условиях тройственного **ограничения** приоритизация — это ответ на вопрос: «От чего мы отказываемся, если не успеваем?»

Модель Кано: Эволюция пользовательского восторга

Наличие функции и удовлетворенность пользователя — это не линейная зависимость.

- **Базовые (Must-be):** Не вызывают восторга, но их отсутствие вызывает ярость.
- **Линейные (Performance):** Чем больше/быстрее, тем лучше.
- **Восторгающие (Delighters):** WOW-эффект. Пользователь не ожидал, но в восторге.

Закон эскалации Кано: Восторгающие свойства неизбежно деградируют в базовые гигиенические факторы со временем (например, оплата онлайн в 1995 vs 2025).

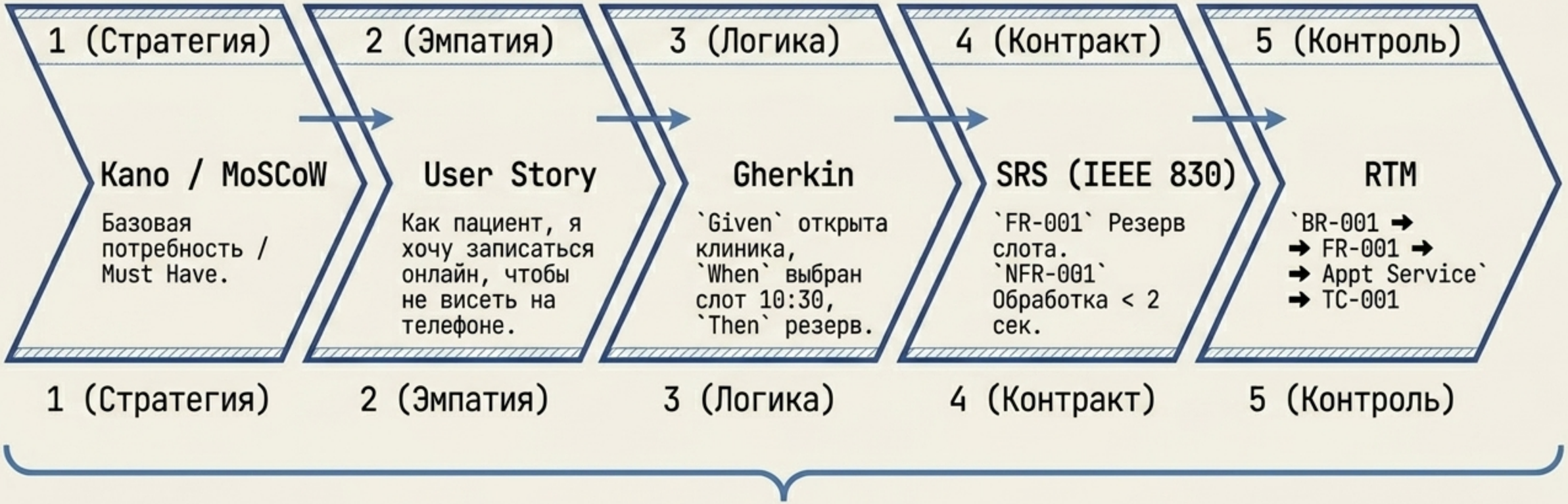


Синтез: Бизнес-реалии пересекаются с эмоциями

Интеграция двух фреймворков дает аналитику железобетонную аргументацию при защите скоупа MVP.

Экосистема требований: Единый непрерывный цикл

Кейс «Онлайн-запись к врачу». Требования рождаются из эмпатии, но реализуются благодаря структуре.



Все фреймворки — это не изолированные теории, а единый инженерный конвейер.