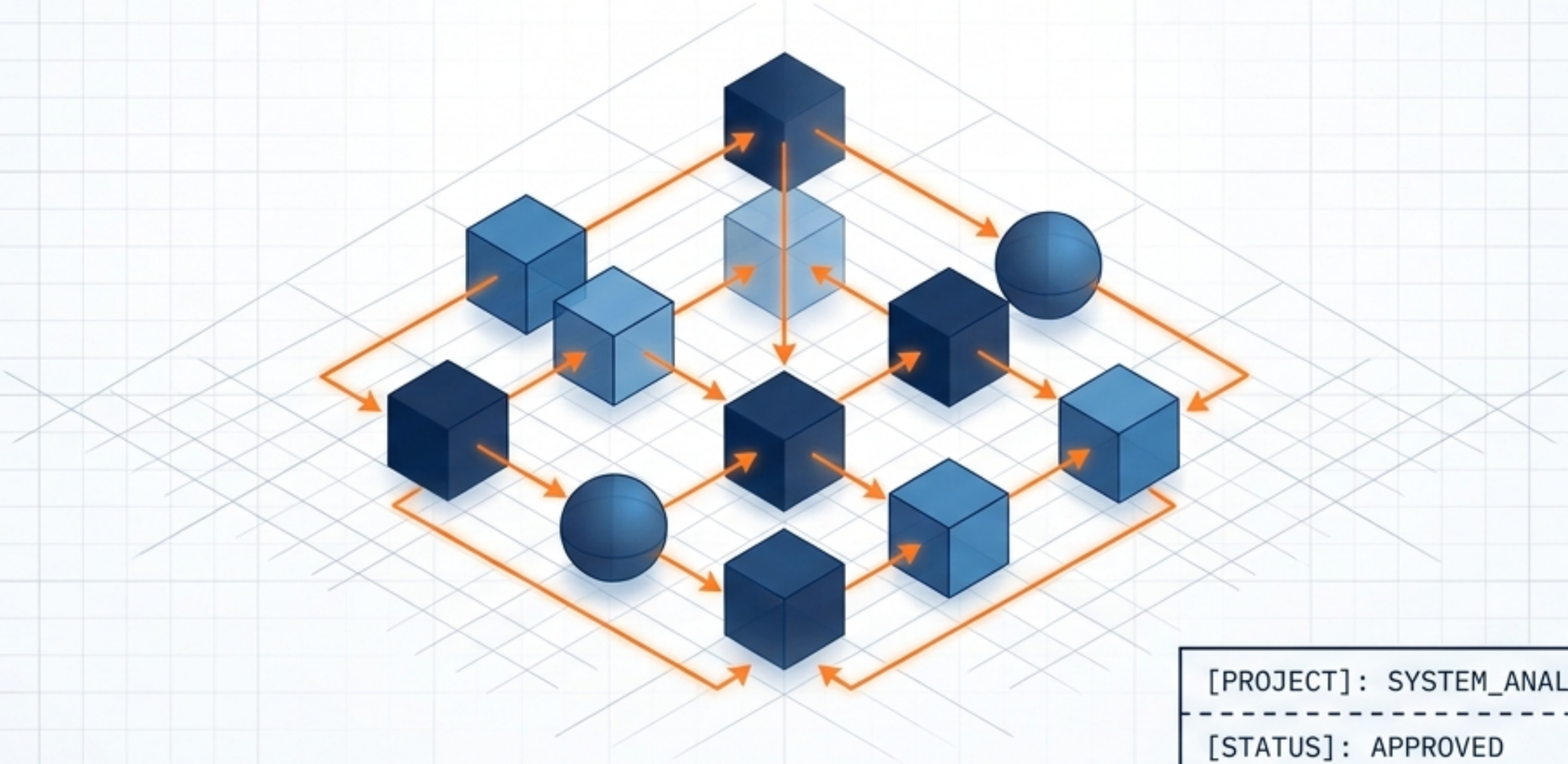
Введение в системный анализ: От хаоса к архитектуре

Инженерный подход к проектированию ИТ-систем, работе со стейкхолдерами и классификации требований.



[PROJECT]: SYSTEM_ANALYSIS_SURVIVAL_GUIDE

[STATUS]: APPROVED

Анатомия системы: Свойство эмерджентности

Система — не просто набор компонентов, а целостный объект, где элементы взаимозависимы.

Frontend:
Интерфейс, каталог.

Backend API:
Логика, корзина, заказы.

База Данных (PostgreSQL/Redis):
Хранение данных.

Внешняя Среда (Надсистема):
Платёжные шлюзы, СДЭК.

Эмерджентность: Ни один компонент по отдельности не умеет «продавать товары». База данных не знает о покупках, фронтенд без бэкенда — просто картинки. Свойство «купить онлайн» возникает только при взаимодействии всех частей.

Два звена одной цепи: БА против СА

	Бизнес-Аналитик (БА)	Системный Аналитик (СА)
Главный вопрос.	«Зачем и ЧТО делаем?»	«КАК именно сделаем?»
Фокус	Проблема бизнеса, процессы (AS-IS / TO-BE).	Решение, архитектура, данные, интеграции.
Инструменты	Miro, Visio, Excel, CJM.	Swagger/OpenAPI, PlantUML, DBeaver, Postman.
Артефакты	Business Requirements Document (BRD), User Story Map.	System Requirements Specification (SRS), API-контракты, SQL-скрипты.

На практике границы размыты. В стартапах (до 15 чел) это один человек. В корпорациях БА работает **«снаружи»** с бизнесом, а СА — **«внутри»** с командой разработки.

Аналитик как центральный узел маршрутизации

Эволюция артефактов: От идеи до кода

Фаза 1: Инициация.

Vision / Concept Document

Границы проекта (Scope), бизнес-обоснование.

Фаза 2: Требования.

SRS (ТЗ) / User Stories

Единый источник правды, детальные спецификации функционала.

Фаза 3: Проектирование.

BPMN / UML / ER-диаграммы

Визуализация процессов и структуры БД.

Фаза 4: Интеграция.

API-контракты (OpenAPI)

Форматы данных, эндпоинты, коды ответов (JSON, REST).

Фаза 5: Контроль.

Матрица трассировки (RTM)

Связь: Требование -> Модуль -> Тест-кейс. Защита от потери логики.

Скрытые стейкхолдеры: Опасность подводной части

Очевидные:
Спонсоры, Product Owner, Пользователи, Разработчики.

Юристы:
Штрафы за 152-ФЗ (персональные данные).

Служба безопасности:
Блокировка релиза из-за нарушения политик шифрования.

Бухгалтерия:
Требование специфических актов сверки, без которых система бесполезна.

DevOps / Инфраструктура:
Отсутствие серверов для развертывания готового кода.

Если вы упустили стейкхолдера на старте, он напомнит о себе на этапе приёмки — когда переделывать код уже слишком дорого.

Выбор архитектуры процесса: Waterfall vs Agile

Waterfall (Каскадная модель)

- Фиксированные фазы. Назад пути нет.
- **Требования:** Полный SRS (100+ стр.) до старта кода.
- **Изменения:** Дорого, через формальный Change Request.
- **Среда:** Госзаказ, медицина, оборонка (высокая цена ошибки).

Agile (Гибкая методология)

- Итеративные циклы (Спринты по 1-4 недели).
- **Требования:** Just-in-time, Бэклог, User Stories.
- **Изменения:** Дешево, приветствуются на ранних этапах.
- **Среда:** Высокая неопределенность, стартапы, быстрый Time-to-Market.

Роль CA в Waterfall: Архитектор требований

Роль CA в Scrum: Непрерывная синхронизация

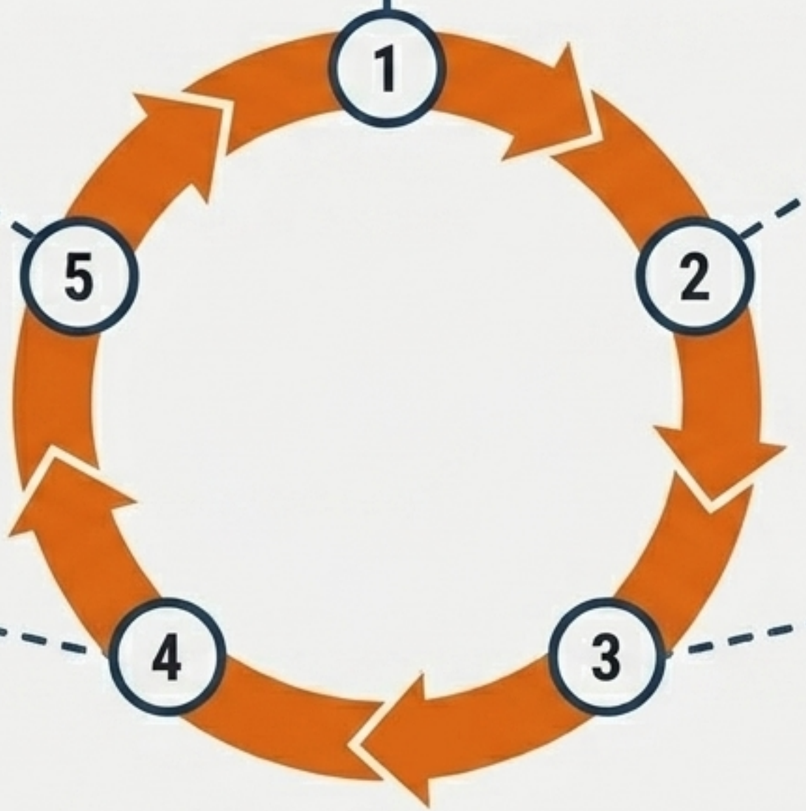
1. Backlog Refinement (Grooming)
Подготовка User Stories до состояния «Ready» (Definition of Ready). Формулировка Acceptance Criteria, оценка рисков.

2. Sprint Planning
Перевод бизнес-задач PO на технический язык. Выявление зависимостей («Эта API нужна до создания таблиц»).

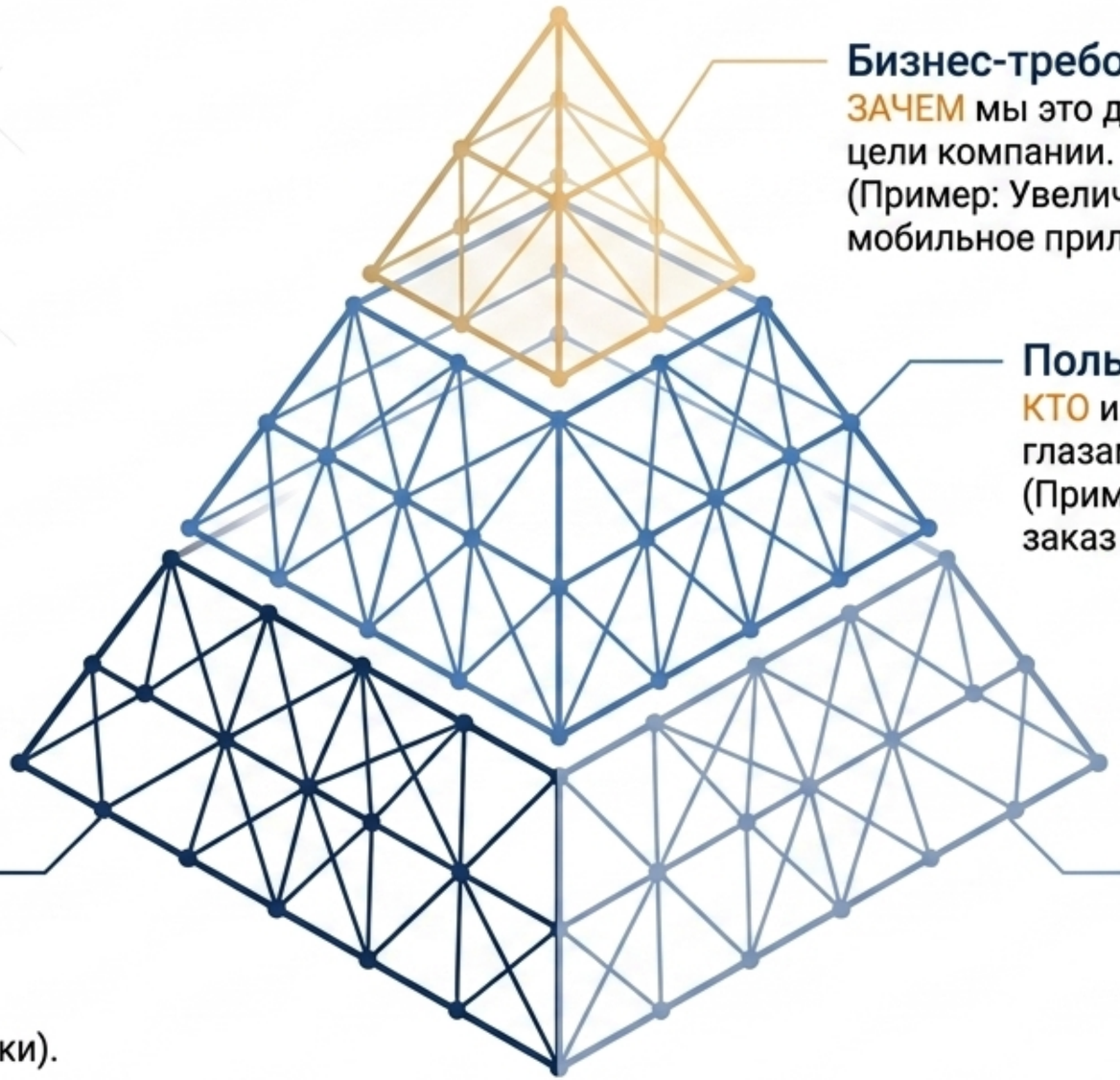
3. Daily Standup
Краткий статус, фиксация технических блокеров из-за неясных требований.

4. Sprint Review (Демо)
Подготовка сценариев демонстрации, фиксация нового фидбека в бэклог.

5. Retrospective
Анализ узких мест в требованиях. Предложения по улучшению коммуникации бизнеса и ИТ.



Архитектура требований: От бизнеса до кода



Бизнес-требования
ЗАЧЕМ мы это делаем? Высокоуровневые цели компании.
(Пример: Увеличить продажи на 30% через мобильное приложение).

Пользовательские требования
КТО и ЧТО хочет сделать? Описание глазами клиента.
(Пример: Как клиент, я хочу оформить заказ за 3 клика).

Функциональные (ФТ)
ЧТО система должна делать? Конкретное поведение.
(Пример: Система должна рассчитывать стоимость доставки).

Нефункциональные (НФТ)
КАК система должна это делать? Ограничения и качество.
(Пример: Время отклика API не более 500 мс).

Фреймворк FURPS+: Матрица ограничений качества

F Functionality

Точность, безопасность.

Пример: Точность расчёта до копейки.

U Usability

Удобство, UX.

Пример: Поддержка screen reader, время освоения < 15 мин.

R Reliability

Надёжность, отказоустойчивость.

Пример: Uptime 99.5%, RTO < 1 часа.

P Performance

Производительность.

Пример: 95-й перцентиль ответа < 500 мс при 1000 RPS.

S Supportability

Поддерживаемость.

Пример: Наличие логов с correlation ID, health check API.

+ (Плюс)

Дополнительные ограничения.

Пример: Legal (152-ФЗ), Physical (серверы в РФ), Design.

Детектор требований: ФТ или НФТ?

Правило: Если в требовании есть цифры (95%, < 200 мс, 99.9%) — это почти всегда НФТ.

Практика декомпозиции: Разбор одной фичи

Бизнес-цель: Обеспечить безопасный доступ сотрудников в систему.

Ветка 1: ФТ (Действия)

- **ФТ-001:** Система должна хешировать пароли (bcrypt) и сравнивать с БД.
- **ФТ-002:** При успехе система выдает JWT-токен на 24 часа.

Ветка 2: НФТ (Атрибуты)

- **НФТ-001 (Security):** Пароль – минимум 8 символов, 1 цифра, 1 заглавная.
- **НФТ-002 (Performance):** Время выдачи токена не превышает 2 секунд.
- **НФТ-003 (Reliability):** Блокировка аккаунта на 15 мин после 5 неверных попыток.

Антипаттерны: Как нельзя писать требования

❌ Плохо	✅ Отлично
«Система должна быть удобной и быстрой». (Субъективно)	**ФТ:** Добавлен поиск с автокомплитом. **НФТ** (Performance): Время поиска < 1 сек.
«Скидка на товар — 10% от 5000 руб». (Это бизнес-правило, а не функция системы).	**ФТ:** «Система применяет скидку 10% к корзине, если сумма >= 5000 руб».
«Система должна выдерживать много пользователей». (Невозможно протестировать).	**НФТ:** «Система поддерживает 10 000 concurrent users без деградации времени отклика > 20%».

Чек-лист аналитика: 5 правил архитектуры требований

- [ ] **[1] Ищите под водой:** Всегда проводите поиск скрытых стейкхолдеров (юристы, DevOps, бухгалтерия) до старта разработки.
- [ ] **[2] Соблюдайте трассируемость:** Каждое требование должно иметь источник (бизнес-цель) и покрытие тест-кейсом (RTM).
- [ ] **[3] Разделяйте ЧТО и КАК:** Никогда не смешивайте функциональные действия и нефункциональные ограничения в одном предложении.
- [ ] **[4] Требования должны быть измеримы:** Избавьтесь от слов «быстро», «удобно», «надежно». Используйте перцентили, секунды и стандарты.
- [ ] **[5] Адаптируйте процесс:** В Waterfall пишите исчерпывающие спецификации (SRS), в Scrum — фокусируйтесь на Definition of Ready для следующего спринта.