Systems • Education

ШКОЛА СИСТЕМНОГО АНАЛИЗА

Скрытая работа аналитика по проектированию систем

Flow 2023

Юрий Куприянов

Канал «Системный сдвиг»:
t.me/systemswing

За 25 лет в ИТ приходилось заниматься всяким...

Запустил «Открытое образование» - 1,5 млн. пользователей с 0 бюджетом на маркетинг.

Спроектировал архитектуру системы сбора и анализа цифрового следа для Университета 2035

Разработал систему Казначейства Газпромбанка

Спроектировал UX конструктора курсов Coreapp,
сервиса портфолио учащегося МЭШ

Написал множество ТЗ, ПМИ и т.п.:
Для ВТБ, Сбера, НРД, ЦифровыеПрофессии.рф Открытого образования...

Что такое системный анализ?

Ана́лиз (др.-греч. ἀνάλυσις «разложение, разделение, расчленение, разборка») — метод исследования, характеризующийся ***выделением и изучением отдельных частей*** объектов исследования.

Системный аналитик должен **разделить** большую систему на понятные **небольшие части**, чтобы разработчики могли создавать её последовательно, шаг за шагом, чтобы знали — с чего начать и в какой последовательности действовать.

Аналитик отвечает за то, что в итоге соберется целостная система и никакие **части и аспекты не будут забыты.**

Главный парадокс системного анализа

Как разделить на части то, чего ещё не существует?

Требования вместо системы

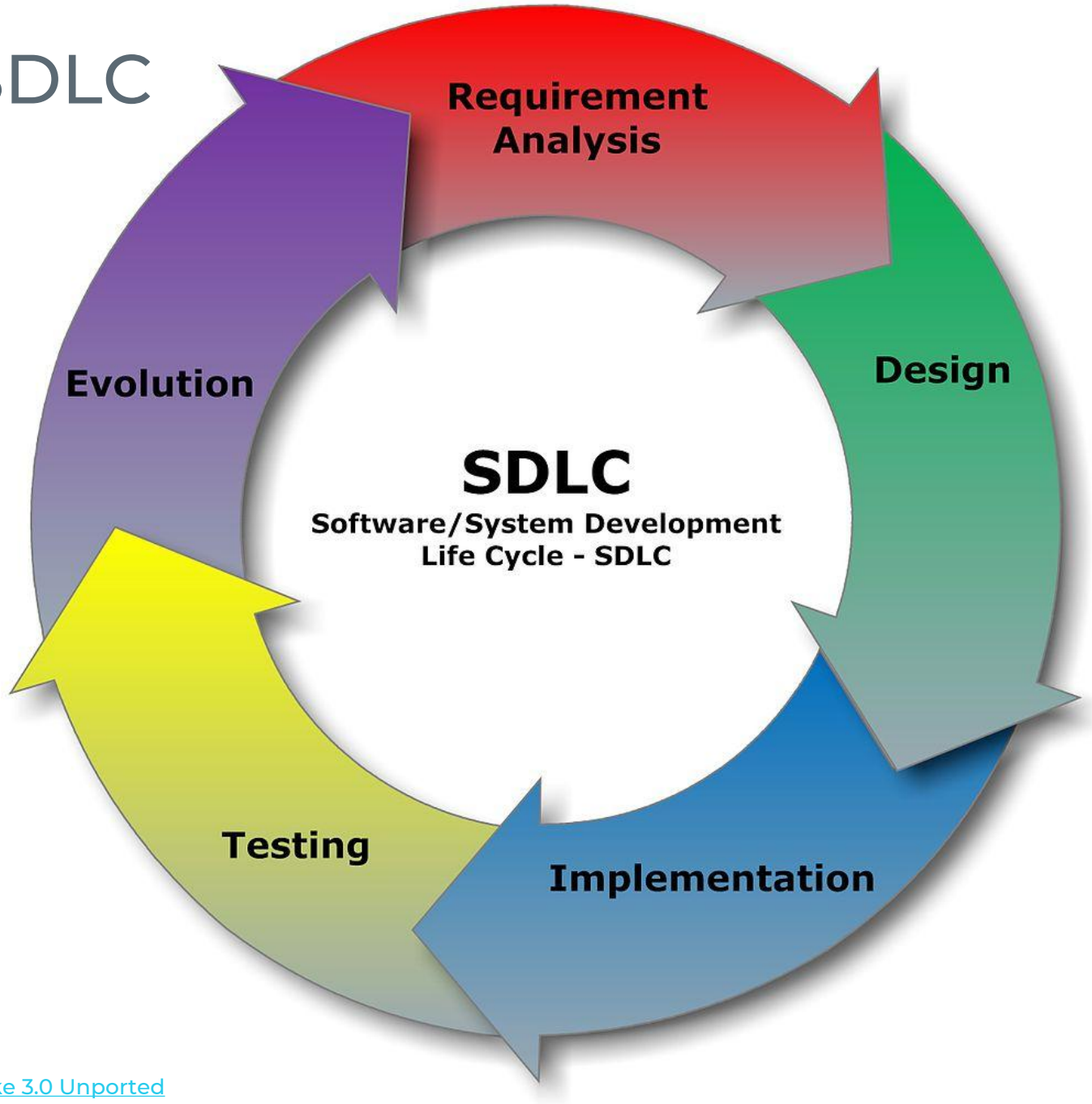
Как разделить на части то, чего ещё не существует?

Системы ещё не существует, но есть требования к системе!
Будем говорить про требования!

Сначала соберём **бизнес-требования** и спланируем весь проект
Потом проведём анализ и разработаем **требования (ФТ и НФТ)**
Потом **спроектируем** систему
Потом **запрограммируем**
Протестируем...
Установим...

Это называется SDLC

Так обычно представляют цикл разработки программного обеспечения



Что «выявляет» и анализирует системный аналитик?

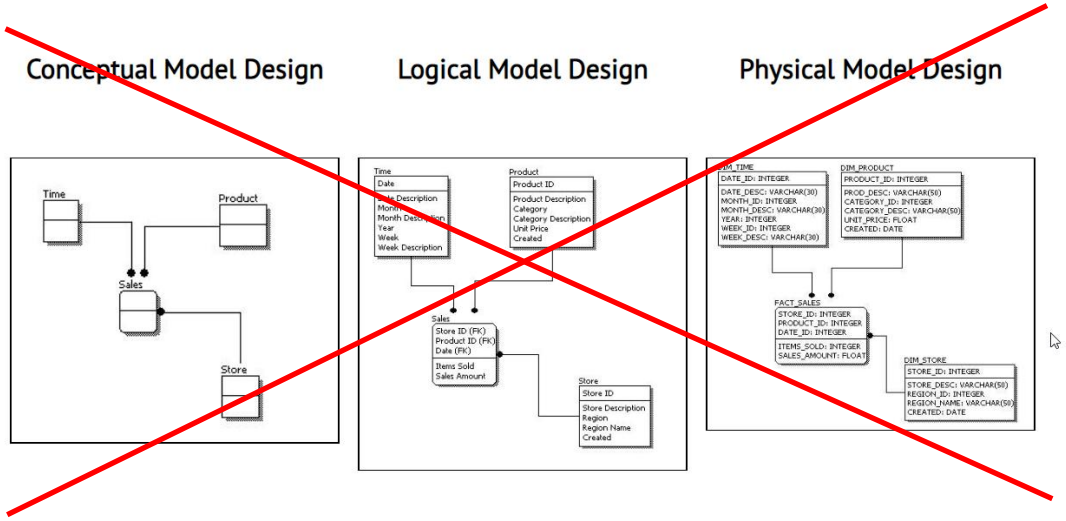
Что «объективно существует» в мире?

Где граница требований и проектных решений?

Очень узкая грань — чуть в сторону, и уже начали проектировать!

Ушли от концептуальной модели объектов к логической — значит, приняли решение по модели хранения. Реляционная? Объектная? Почему именно они? Почему не графы? Почему не документы? Триплеты RDF? Пары key-value?

На основе чего принято проектное решение?



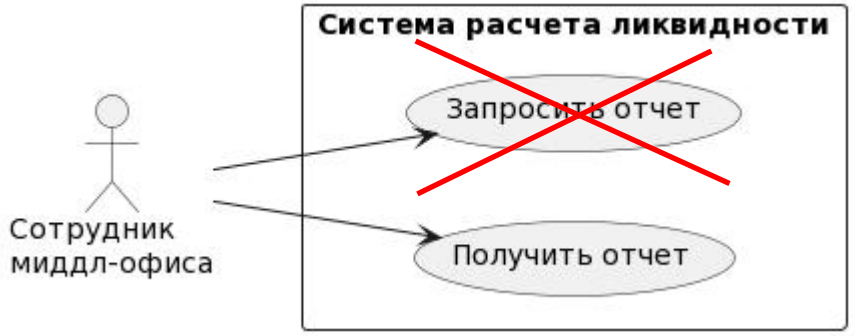
Где граница требований и проектных решений?

Очень узкая грань — чуть в сторону, и уже начали проектировать!

Начали расписывать use case по шагам сценария? — проектируете взаимодействие пользователя с системой, то есть UX.

Основной поток:

1. Пользователь выбирает функцию Создать задачу
2. Система выводит форму [Задача] для ввода информации
3. Пользователь вносит данные в форму Задача
4. Пользователь выбирает функцию Сохранить задачу
5. Система убеждается в полноте данных, введенных пользователем
6. Система сохраняет введенные пользователем данные
7. Система сообщает о успешном завершении операции



Где граница требований и проектных решений?

Очень узкая грань — чуть в сторону, и уже начали проектировать!

Требования у вас **полны**, **непротиворечивы**, **прослеживаемы** и не зависят от реализации? А как вы решали противоречия? Откинули часть требований? Предложили компромисс? Или **придумали** решение, учитывающее их все?

Зафиксировали требования **к устройству системы** — сделали проектное решение! Упомянули **тип системы**? Упомянули какой-то **элемент интерфейса**? Указали **технологию или конкретный формат**?

А почему именно этот? Как вы делали этот выбор?

Существуют ли требования?

На самом деле требования заказчика **всегда**:

- **Противоречивы**
- **Неполны**
- **Необязательны**
- **Меняются**

* В любом процессе есть конфликт. Если вы не видите противоречий в требованиях — вы выявили не всех стейкхолдеров!

** Стейкхолдер всегда говорит о какой-то отдельной детали, на которой он сфокусирован. Вся обвязка, обеспечение и частные случаи остаются за рамками.

*** Обязательны только требования регуляторов и нормативки. Обычно одну и ту же задачу можно решить разными способами.

The illusion of requirements in software development.

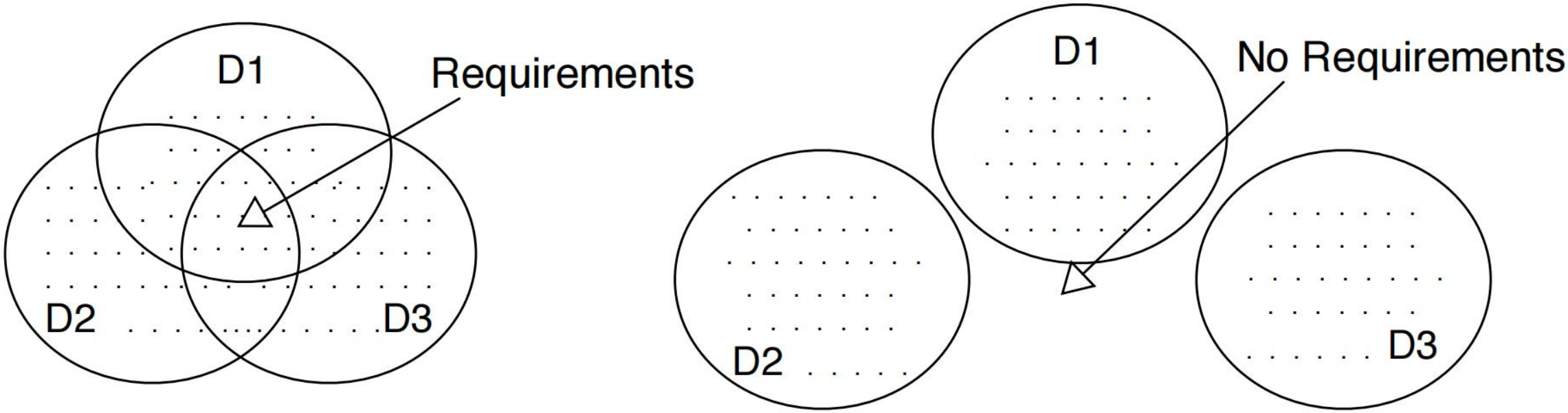


Figure 1. Relationship between features (dots) and requirements.

Ralph, P. The illusion of requirements in software development. Requirements Eng 18, 293–296 (2013).

Что говорит наука

Ralph, P. **The illusion of requirements in software development.** Requirements Eng 18, 293–296 (2013).

P. Ralph and R. Mohanani, "**Is Requirements Engineering Inherently Counterproductive?**" 2015 IEEE/ACM 5th International Workshop on the Twin Peaks of Requirements and Architecture, Florence, Italy, 2015, pp. 20-23

R. Mohanani, P. Ralph, B. Turhan and V. Mandić, "**How Templated Requirements Specifications Inhibit Creativity in Software Engineering**" in IEEE Transactions on Software Engineering, vol. 48, no. 10, pp. 4074-4086, 1 Oct. 2022

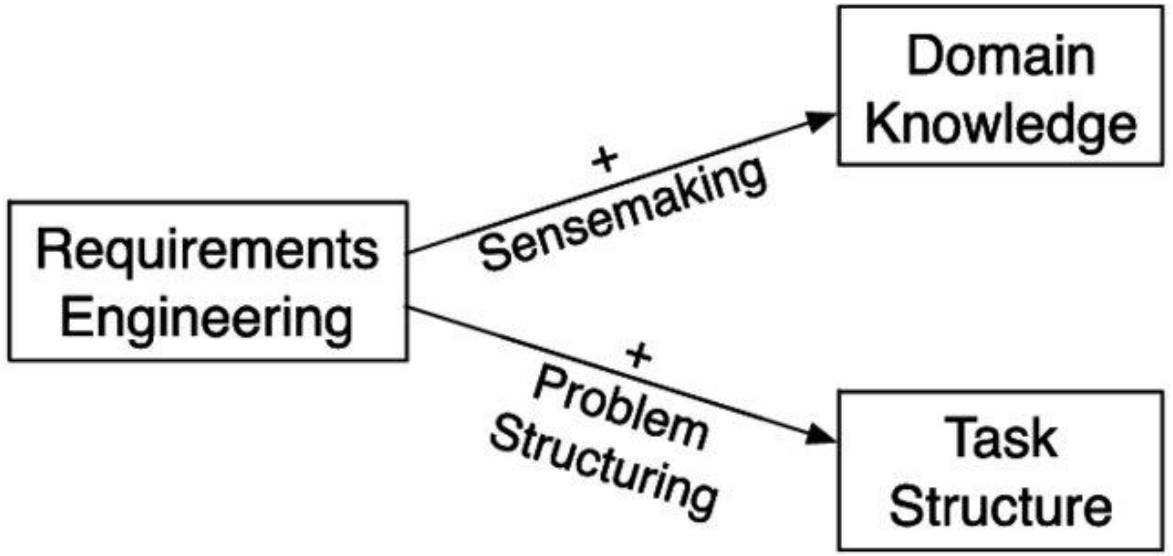
How Templated Requirements Specifications Inhibit Creativity in Software Engineering

> ... the more naturalistic side of RE, as well as research in human–computer interaction, user–centred design, and the interdisciplinary design literature tends to assume that:

- **software projects do not have discoverable and documentable requirements** (cf. [5]);
- **stakeholders do not even have stable, retrievable preferences** (cf. [6]); and
- products have numerous stakeholders who **do not agree on the problem(s) to solve or how the product should solve them** (cf. [7]).

Forcing vague, unstable, conflicting preferences into unam_x0002_biguous, consistent requirements specifications encourages designers to converge **prematurely on oversimplified problems and inappropriate solutions** [8].

Is Requirements Engineering Inherently Counterproductive?

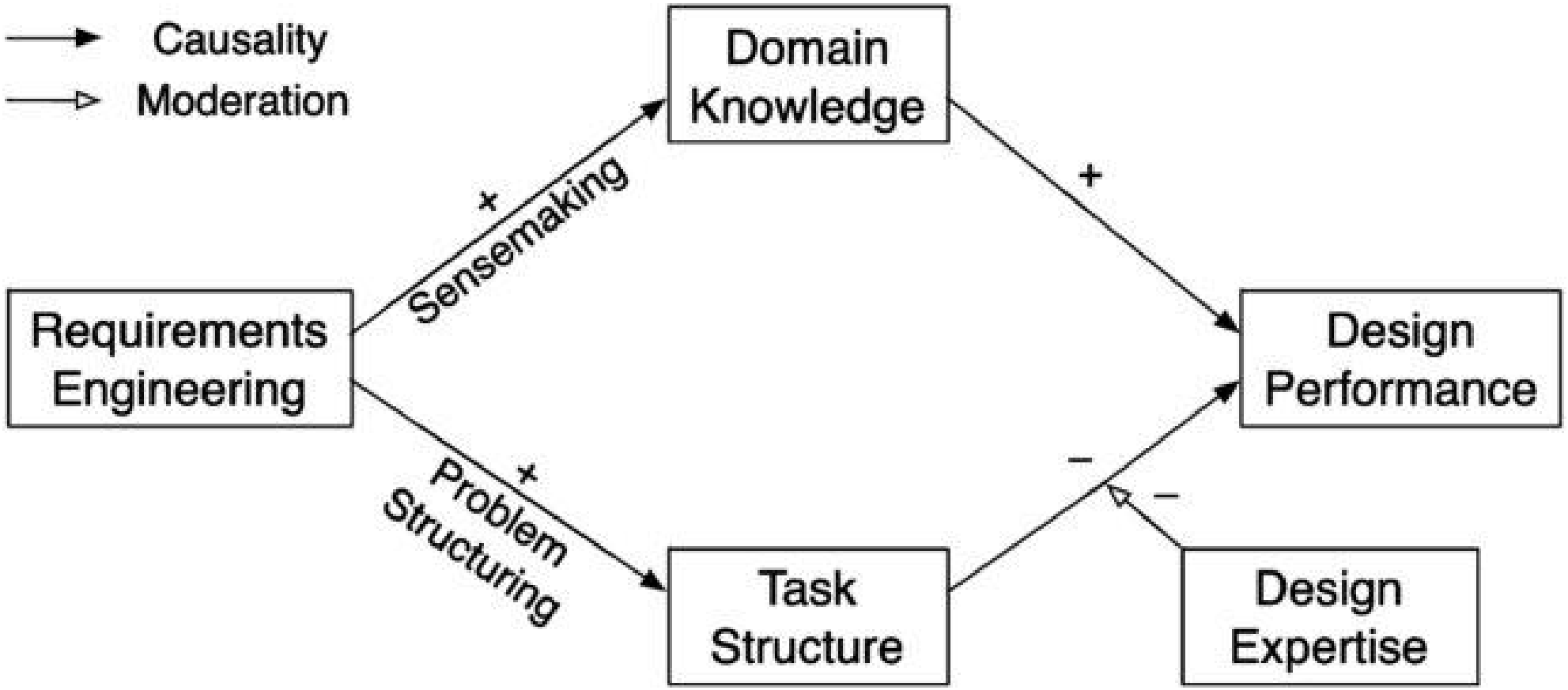


> Problem structuring involves **inventing** **goals, problems, constraints and other structure elements** [12]. Problem structure does not exist in an objective reality waiting for discovery; it is created by human actors [13].

Is Requirements Engineering Inherently Counterproductive?

> Expert designers explore problematic contexts by generating **solution concepts** [13], [51]. They intentionally maintain ambiguity and continue problem framing throughout the development process [52]. **Problem framing and designing solutions are one and the same process – coevolution** [49], [51].
>
> Lifecycle views of software development therefore **incorrectly categorize problem structuring with sensemaking instead of with design**. Consequently, structuring tasks are incorrectly assigned to managers and analysts instead of architects and developers.

Is Requirements Engineering Inherently Counterproductive?



Рекомендации Пола Ральфа

1. Избегайте чрезмерно структурированных, переупрощенных и излишне рационализированных требований.

2. Признавайте, анализируйте и принимайте двумсысленность и неопределенность как неизбежность и даже преимущество.

3. Рассматривайте структурирование проблем и проектированование решения как единый процесс.

4. Структурировать проблему и проектировать решение должны одни и те же люди.

Что конкретно мы проектируем?

- **Взаимодействие** пользователя с системой
- **Структуру** хранения данных
- Взаимодействие с **внешними системами**
- **Внутреннее устройство** системы (архитектуру)

По-хорошему, это четыре разных специальности:

- UX-проектировщик (это не дизайнер!)
- Проектировщик БД / хранилищ данных (это разработчик?)
- Проектировщик интеграций и API (это снова разработчик?)
- Архитектор ПО / архитектор решений (или опять разработчик?)

Взаимодействие пользователя с системой

Каждый раз, когда мы описываем сценарий – use case или BDD или описываем интерфейс – мы проектируем взаимодействие. Что будет делать человек?

- Какая **задача** стоит у пользователя?
- Какие **шаги** он предпринимает? (Даже без системы)
- Какие **решения** он должен принять?
- Какая **информация** ему нужна для этих решений?
  - В первую очередь? Дополнительная?
  - Что он должен увидеть? Что ввести? Какие команды отдать?
- Какими **объектами** он манипулирует?

Взаимодействие пользователя с ~~системой~~ другими людьми объектами реального мира

Каждый раз, когда мы описываем сценарий – use case или BDD или описываем интерфейс – мы проектируем взаимодействие. Что будет делать человек?

- Какая **задача** стоит у пользователя?
- Какие **шаги** он предпринимает? (Даже без системы)
- Какие **решения** он должен принять?
- Какая **информация** ему нужна для этих решений?
  - В первую очередь? Дополнительная?
  - Что он должен увидеть? Что ввести? Какие команды отдать?
- Какими **объектами** он манипулирует?

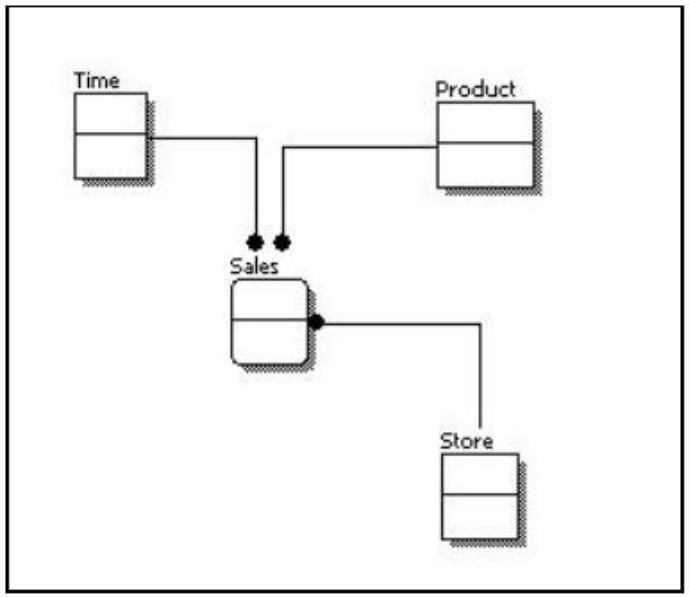
Взаимодействие с пользователем: ресурсы

Неочевидные источники:

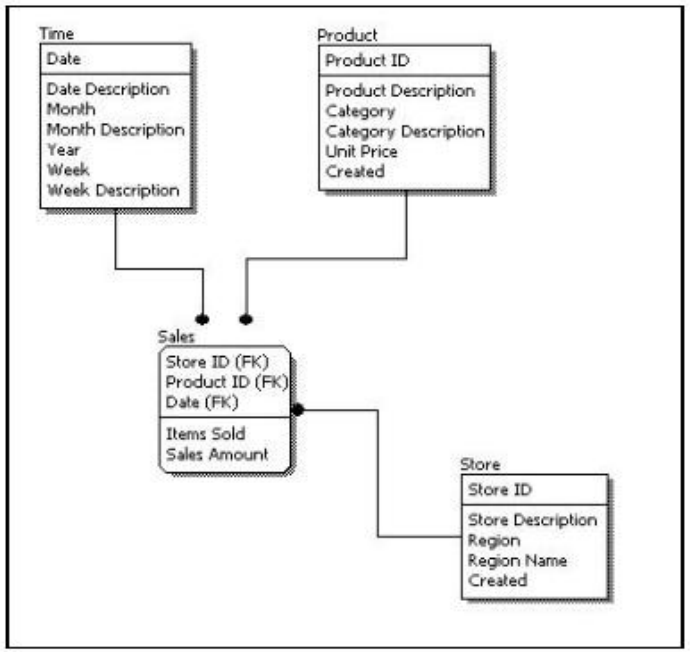
- Джоел Сполски «**Руководство по UI дизайну для программистов**» (https://www.joelonsoftware.com/2001/10/24/user-interface-design-for-programmers/)
- Что-то по информационной архитектуре: например, Дэн Браун «**8 принципов информационной архитектуры**»
- ГОСТ Р ИСО 9241-210— 2016, ГОСТ Р ИСО 14915-1—2016, ГОСТ Р ИСО 14915-3—2016
- Jeff Gothelf, Josh Seiden. **Lean UX**, 3rd Edition
- Курс «**Дизайн интерфейсов для недизайнеров**» Алексея Копылова (https://systems.education/4-non-designers)

Проектирование хранения данных

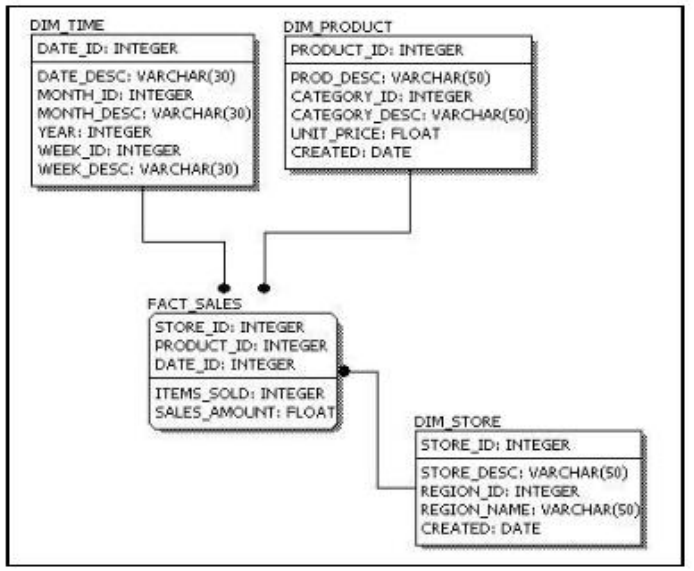
Conceptual Model Design



Logical Model Design



Physical Model Design



Проектирование хранения данных

Концептуальная модель – «что есть в мире».

Логическая модель – какой ***подход к хранению мы выбираем***. Тут нам нужно понять структуру данных и процессы работы с ними.

Физическая модель – как настроить конкретную схему данных в выбранном продукте.

Проектирование хранения данных: вопросы

- Какие бывают данные?
- К какой категории относятся данные? (структурированные, полуструктурированные, неструктурированные)
- Зачем данные? (Учет, операции, аналитика...)
- Как выполняются запросы? Индексы, ключи, деревья, шардирование, профайлинг запросов... Изучите хотя бы один язык запросов (SQL – must have).
- Какие есть типы данных и языки валидации? Регулярные выражения?
- Как распределяются данные по разным узлам? CAP-теорема.
- Транзакции, уровни изоляции, ACID, Saga.

Choosing the Right Database

AWS Summit Bahrain
15 SEP 2019
@awsgeek

WHAT IS YOUR DB STRATEGY?

PAST: PICK A PLATFORM
BUILD AROUND IT

PICK THE RIGHT TOOL FOR THE RIGHT JOB !

1 LEGACY APPS

LIFT & SHIFT

DMS

RDS

MOVE EXISTING APPS TO THE CLOUD

2 MODERN APPS

QUICKLY BUILD NEW APPS IN THE CLOUD

- MILLIONS OF USERS
- GLOBAL CUSTOMERS
- HIGHLY SCALABLE

DON'T TRADE
- Δ FUNCTIONALITY
- Δ PERFORMANCE
- Δ SCALABILITY

FOR CONVENIENCE

AMAZON AURORA

- ☑ 5x PERF
- ☑ UP TO 15 READ REPLICAS
- ☑ 6 WAY DATA REPLICATION

TIME-SERIES

- SEQ RECORDED OVER TIME
- TIME IS PRIMARY ACCESS PATTERN

DIFFICULT FOR REL'N DB

- SERVERLESS
- ANALYTICS

TIMESTREAM

LEDGER

- IMMUTABLE DATA
- CRYPTO VERIFIABLE

BLOCKCHAIN
journal

QLDB

DOCUMENT

- FLEXIBLE SCHEMA

DOCUMENTDB

SEARCH

- FULL TEXT
- FULLY MANAGED

ELASTICSEARCH SERVICE

RELATIONAL

- BREAK DATA INTO TABLES
- HIGHLY STRUCTURED

RDS

- MYSQL
- ORACLE
- POSTGRESQL
- SQL SERVER
- MARIADB

KEY-VALUE

- HORZ PARTITIONING
- CONSISTENT PERF AT SCALE

DYNAMODB

DOCUMENTS TOO

GRAPH

- HIGHLY CONNECTED
- REL'N ARE 1ST CLASS

NEPTUNE

buy → Product
query these edges → like → Sport

IN-MEMORY

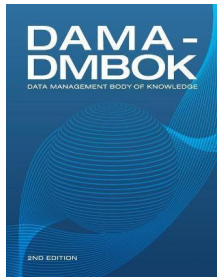
- FAST ACCESS
- SUB MS LATENCY

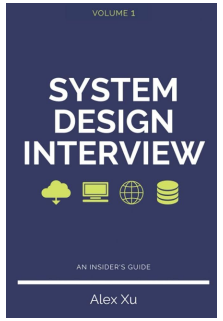
ELASTICACHE

- REDIS
- MEMCACHED

https://www.awsgeek.com/

Проектирование хранения данных: ресурсы



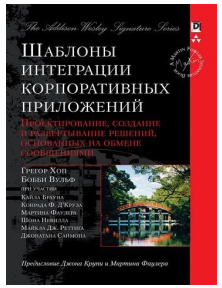


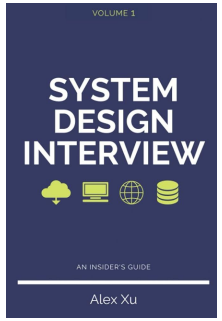
- DAMA DMBoK2: Data Management Body of Knowledge
- Alex Xu. System Design Interview. https://bytebytego.com/
- https://www.piter.com/collection/programmirovanie-osnovy-i-algoritmy/product/system-design-podgotovka-k-slozhnomu-intervyu
- Каталог ссылок на тему баз данных и анализа данных: https://systems.wiki/database

Взаимодействие с внешними системами, API

- Паттерны интеграций
- Разновидности технологий программных интерфейсов, принципы создания и спецификации API:
  - **REST, SOAP, GraphQL, gRPC**
    - **OpenAPI, AsyncAPI, RAML**
    - Шины, брокеры, очереди (message oriented middleware)
- Трансформация и валидация данных: XSLT, регулярные выражения...
- Сетевые протоколы и форматы: **http(s), http/3, xml, json, protobuf, avro, flatbuffers...**
- Безопасность: **OAuth, SSL, OWASP**, ...
- Особенности конкретных продуктов: **Kafka, Rabbit, KeyCloak и т.п.**...

Проектирование интеграций: ресурсы





- Хоп Грегор, Вульф Бобби. Шаблоны интеграции корпоративных приложений.

- Alex Xu. System Design Interview. https://bytebytego.com/
- https://www.piter.com/collection/programmirovanie-osnovy-i-algoritmy/product/system-design-podgotovka-k-slozhnomu-intervyu

- Каталог ссылок на тему интеграций: https://systems.wiki/integration

Проектирование архитектуры

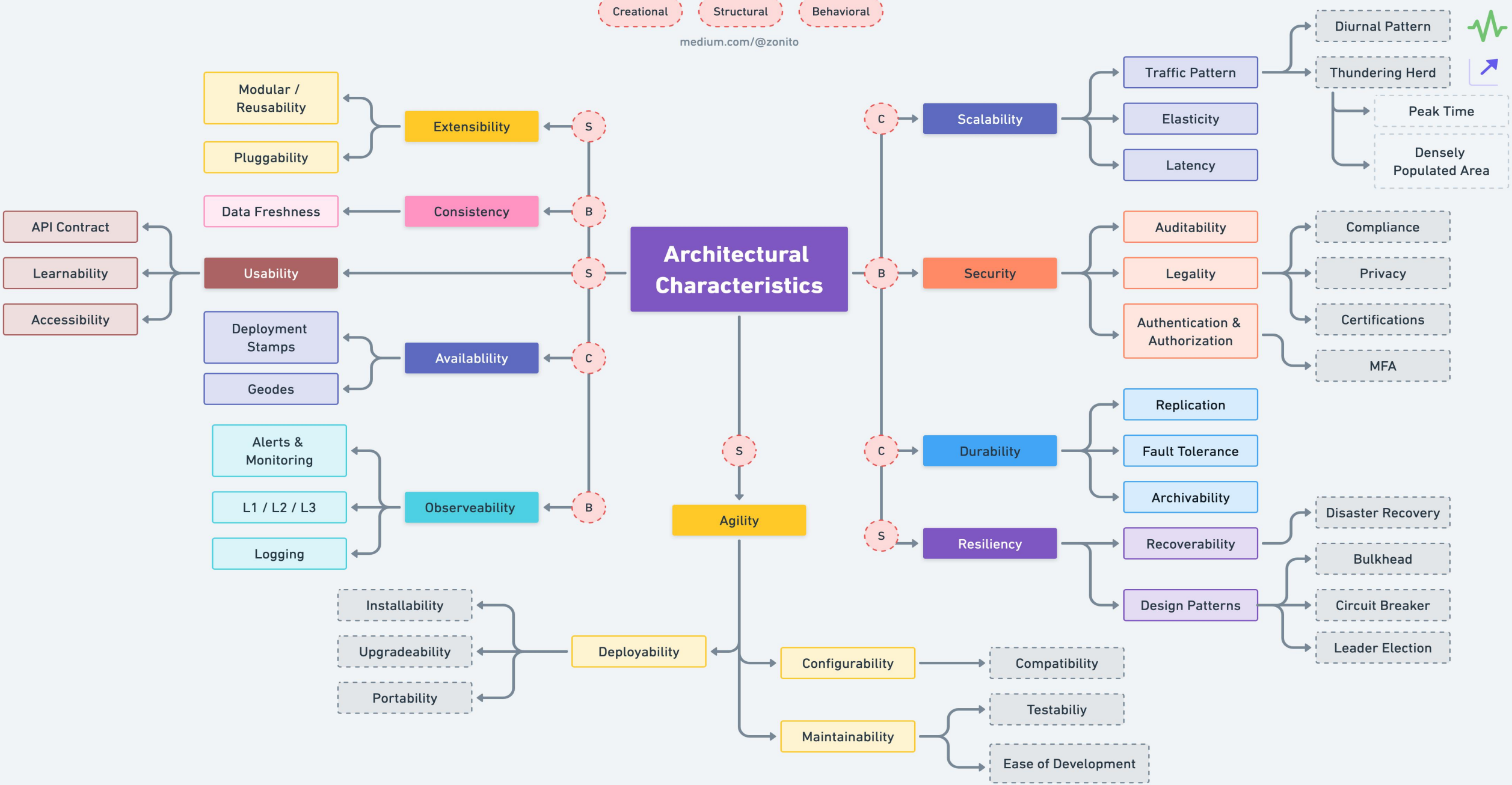
- Кодовая база и **распределение задач по командам**, Закон Конвея.
- Архитектурные стили: монолиты, пайплайны, микросервисы, EDA...
- **Сценарии развертывания**, DevOps, CI/CD.
- DDD, ограниченные контексты, Event Storming
- Паттерны: Saga, CQRS, SideCar, API Gateway, CircuitBreaker, Event Sourcing, CDC и т.д.
- **Нефункциональные требования!**

- Accessibility
- Adaptability
- Auditability and control
- Availability (see service level agreement)
- Backup
- Boot up time
- Capacity, current and forecast
- Certification
- Compliance
- Configuration management
- Conformance
- Cost, initial and Life-cycle cost
- Data integrity
- Data retention
- Dependency on other parties
- Deployment
- Development environment
- Disaster recovery
- Documentation
- Durability
- Efficiency (resource consumption for given load)
- Effectiveness (resulting performance in relation to effort)
- Elasticity
- Emotional factors (like fun or absorbing or has "Wow! Factor")
- Environmental protection
- Escrow
- Exploitability
- Extensibility (adding features, and carry-forward of customizations at next major version upgrade)
- Failure management
- Fault tolerance (e.g. Operational System Monitoring, Measuring, and Management)
- Flexibility (e.g. to deal with future changes in requirements)
- Footprint reduction - reduce the exe files size
- Integrability (e.g. ability to integrate components)
- Internationalization and localization
- Interoperability
- Legal and licensing issues or patent-infringement-avoidability
- Maintainability (e.g. mean time to repair – MTTR)
- Management
- Memory Optimization
- Modifiability
- Network topology
- Open source
- Operability
- Performance / response time (performance engineering)
- Platform compatibility
- Privacy (compliance to privacy laws)
- Portability
- Quality (e.g. faults discovered, faults delivered, fault removal efficacy)
- Readability
- Reliability (e.g. mean time between/to failures – MTBF/MTTF)
- Reporting
- Resilience
- Resource constraints (processor speed, memory, disk space, network bandwidth, etc.)
- Response time
- Reusability
- Robustness
- Safety or factor of safety
- Scalability (horizontal, vertical)
- Security (cyber and physical)
- Software, tools, standards etc. Compatibility
- Stability
- Supportability
- Testability
- Throughput
- Transparency
- Usability (human factors) by target user community
- Volume testing

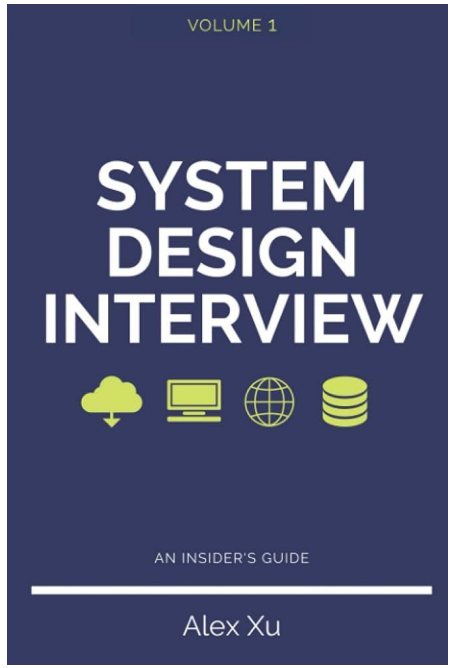
Architectural Characteristics

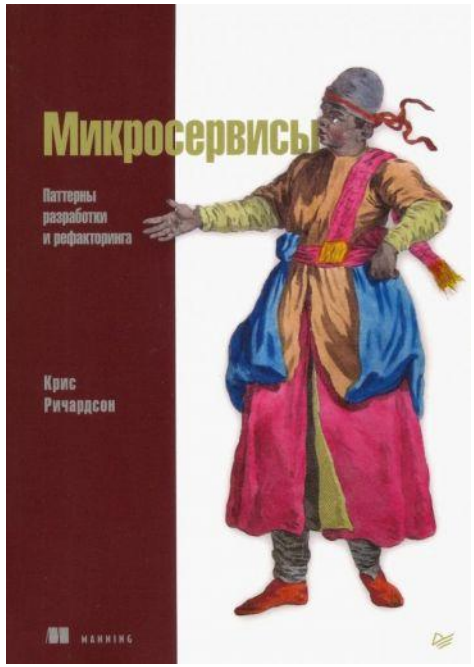
Creational | Structural | Behavioral

medium.com/@zonito



Проектирование архитектуры: ресурсы





- A pattern language for microservices https://microservices.io/patterns/index.html

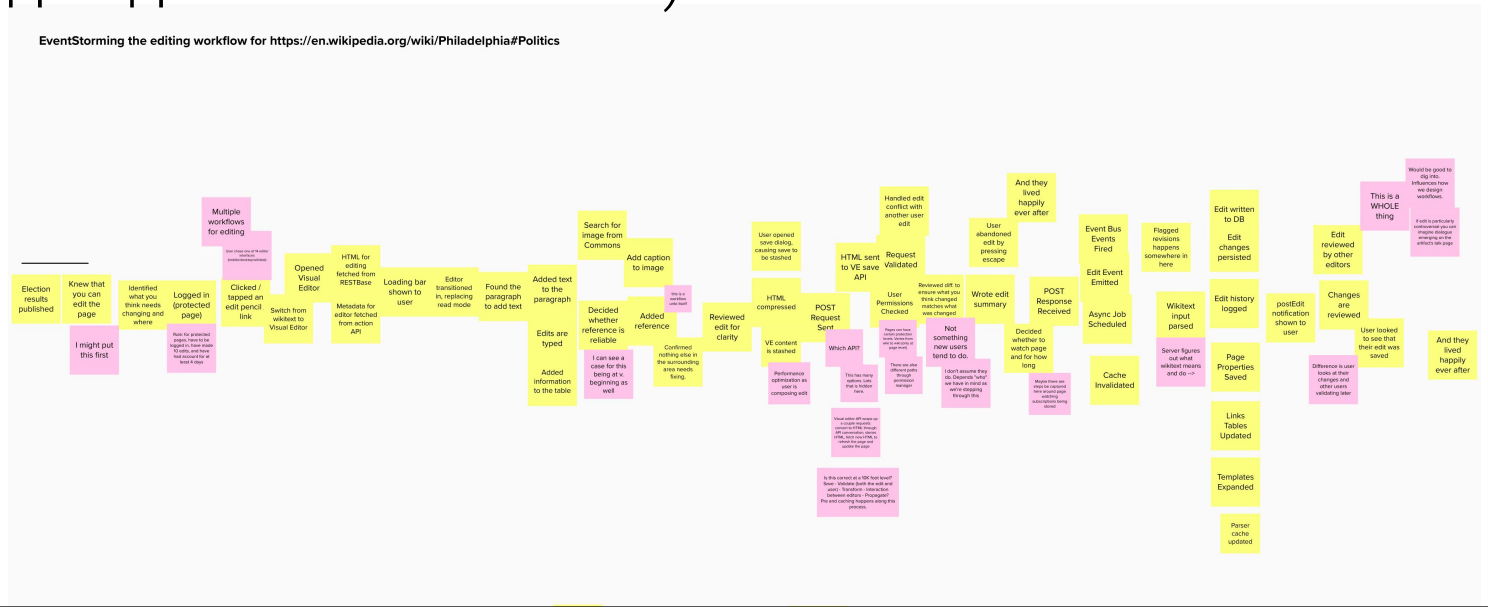
В чем роль аналитика: организатор работы

Структурировать проблему и проектировать решение должны одни и те же люди

Аналитик как организатор и фасилитатор совместной работы. Организуйте воркшоп, пригласите стейкхолдеров и разработчиков, объясните правила – и люди сделают всё сами :)

Форматы:

- Event Storming
- User Story Mapping
- Design Sprint



Спасибо за внимание!
Вопросы?

Юрий Куприянов

Systems.Education

ШКОЛА
СИСТЕМНОГО
АНАЛИЗА

t.me/systemswing

Канал по системному анализу и проектированию систем