CUSTIS® ИТ-РЕШЕНИЯ ДЛЯ РАЗВИТИЯ

ANALYST DAYS #13

Проектирование в разных культурах и парадигмах разработки

Максим Цепков

Главный архитектор решений CUSTIS

Навигатор в мире Agile, бирюзовых организаций и спиральной динамики

AnalystDays
Москва, 19 ноября 2021

О чем этот доклад

Ситуация:

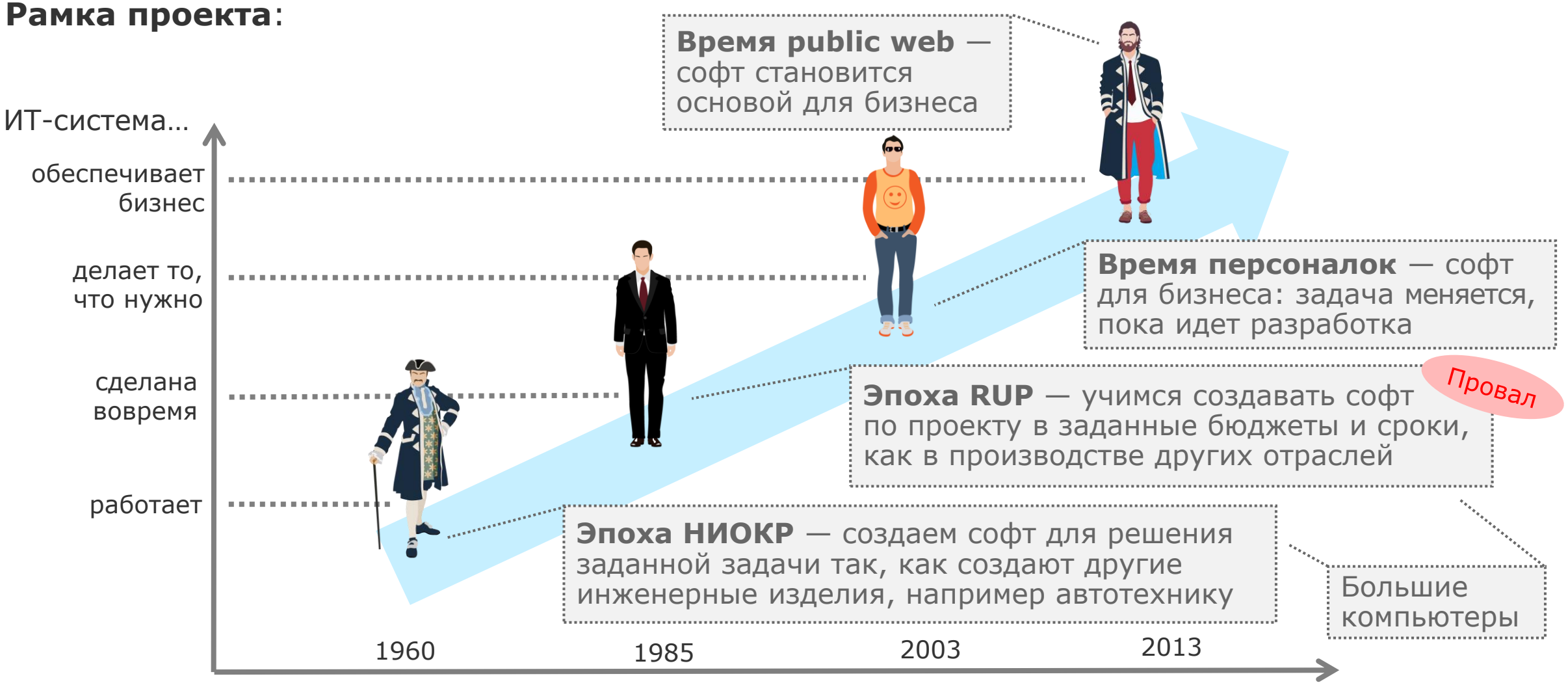
- каждая эпоха формировала свои методы проектирования и ведения разработки
- они несут контекст эпохи, настроены на те задачи, которые были нужны, но контекст часто отсутствует в учебниках, **ведь других задач тогда не было**
- поэтому люди часто не видят ограничения и уверены, что методы можно применять для любых задач, рассматривают как альтернативы

Я хочу показать не просто разнообразие вариантов, а их происхождение и связь:

- чтобы мы могли ориентироваться в разных методах, выбирать их для проектов
- чтобы могли аргументированно обсуждать методы с другими

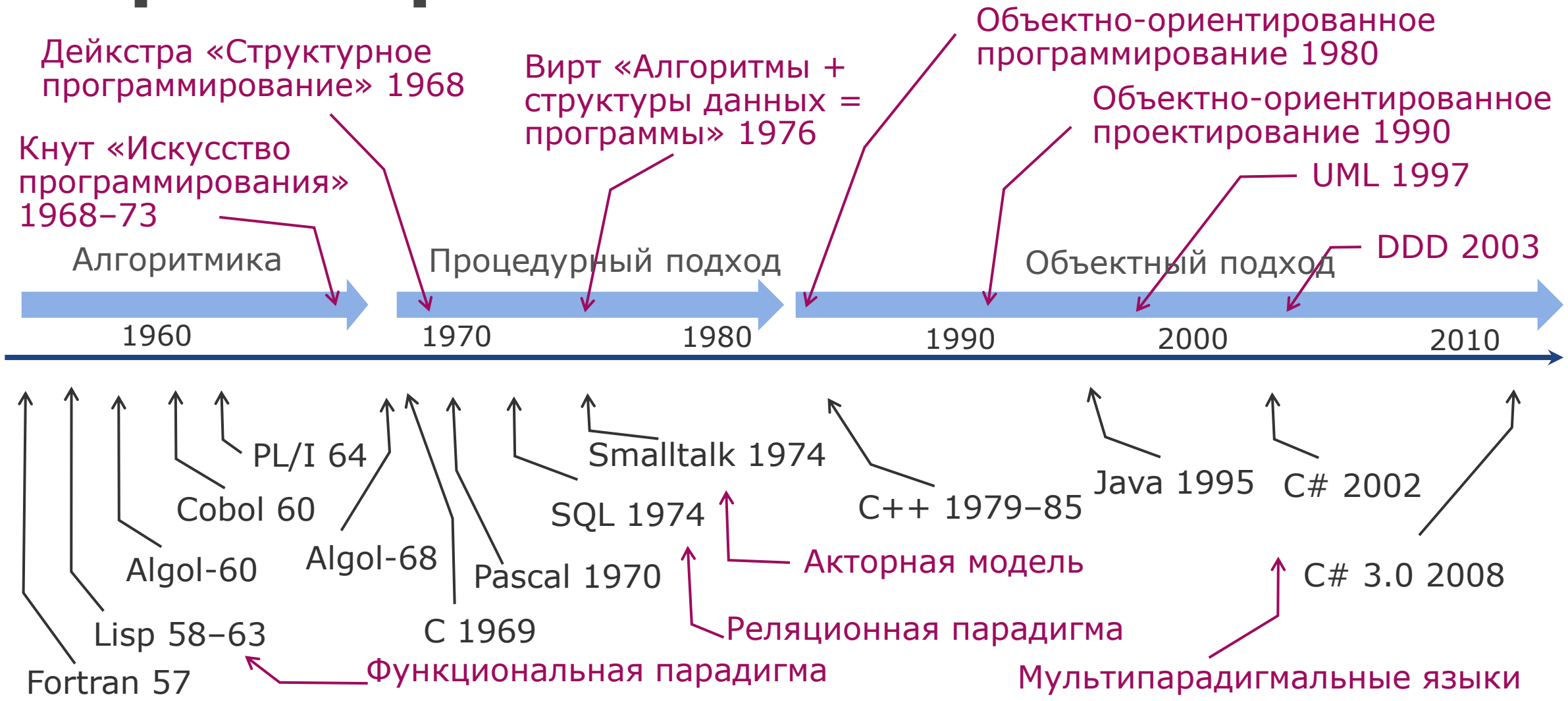
История культур ИТ-проектов

Рамка проекта:



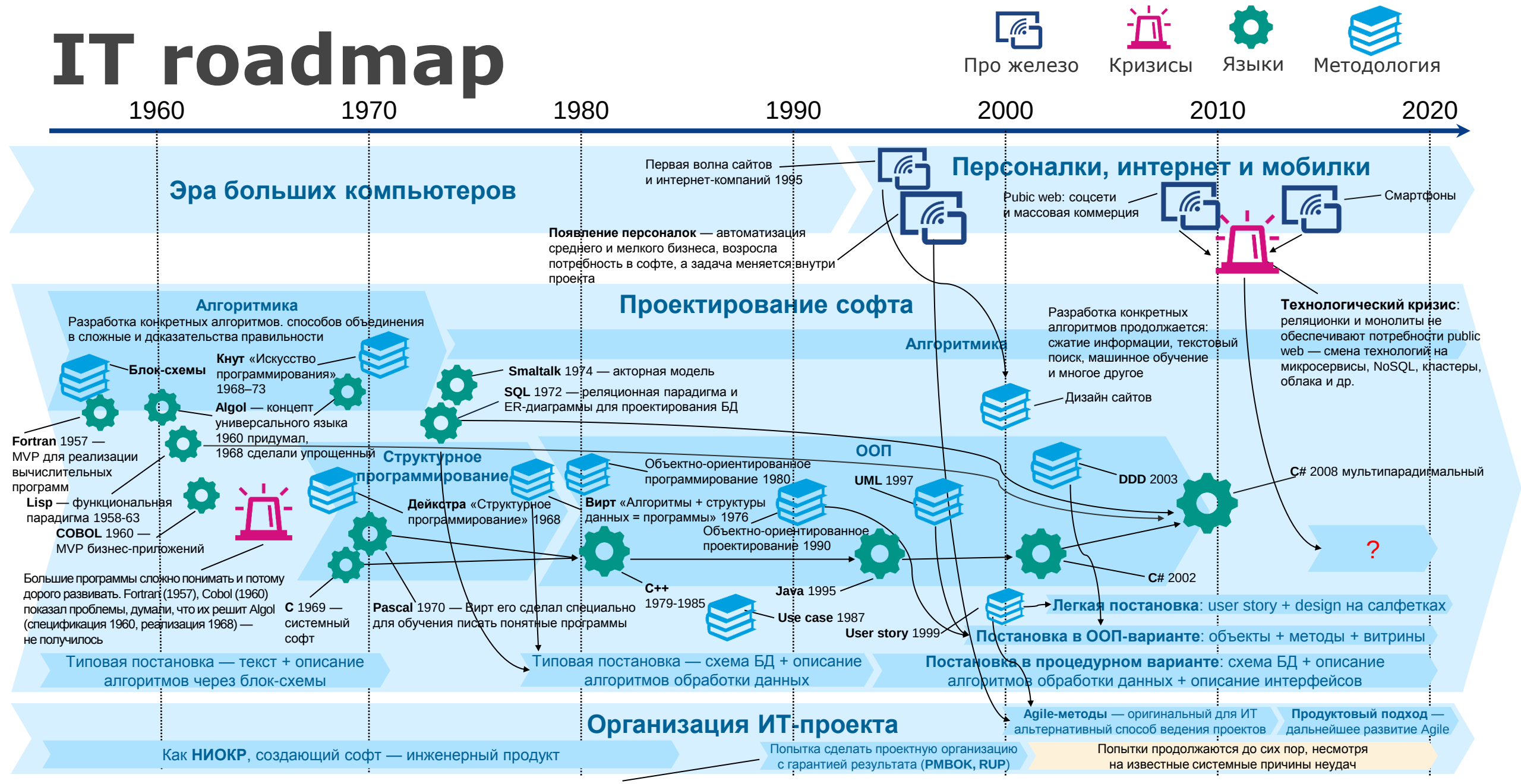
История программирования и проектирования

CUSTIS



Мои статьи «История IT. Когда компьютеры были большими...» и «История IT. ООП»

IT roadmap


Про железо
Кризисы
Языки
Методология
1960
1970
1980
1990
2000
2010
2020
Эра больших компьютеров
Первая волна сайтов и интернет-компаний 1995
Персоналки, интернет и мобилки
Pubic web: соцсети и массовая коммерция
Смартфоны
Появление персоналок — автоматизация среднего и мелкого бизнеса, возросла потребность в софте, а задача меняется внутри проекта
Технологический кризис: реляционки и монолиты не обеспечивают потребности public web — смена технологий на микросервисы, NoSQL, кластеры, облака и др.
Проектирование софта
Алгоритмика
Разработка конкретных алгоритмов. способов объединения в сложные и доказательства правильности
Блок-схемы
Кнут «Искусство программирования» 1968–73
Algol — концепт универсального языка 1960 придумал, 1968 сделали упрощенный
Fortran 1957 — MVP для реализации вычислительных программ
Lisp — функциональная парадигма 1958-63
COBOL 1960 — MVP бизнес-приложений
Большие программы сложно понимать и потому дорого развивать. Fortran (1957), Cobol (1960) показал проблемы, думали, что их решит Algol (спецификация 1960, реализация 1968) — не получилось
Smaltalk 1974 — акторная модель
SQL 1972 — реляционная парадигма и ER-диаграммы для проектирования БД
Алгоритмика
Разработка конкретных алгоритмов продолжается: сжатие информации, текстовый поиск, машинное обучение и многое другое
Дизайн сайтов
Структурное программирование
Дейкстра «Структурное программирование» 1968
C 1969 — системный софт
Pascal 1970 — Вирт его сделал специально для обучения писать понятные программы
ООП
Объектно-ориентированное программирование 1980
Вирт «Алгоритмы + структуры данных = программы» 1976
Объектно-ориентированное проектирование 1990
C++ 1979-1985
Java 1995
UML 1997
Use case 1987
User story 1999
DDD 2003
C# 2002
C# 2008 мультипарадигмальный
?
Легкая постановка: user story + design на салфетках
Постановка в ООП-варианте: объекты + методы + витрины
Типовая постановка — текст + описание алгоритмов через блок-схемы
Типовая постановка — схема БД + описание алгоритмов обработки данных
Постановка в процедурном варианте: схема БД + описание алгоритмов обработки данных + описание интерфейсов
Организация ИТ-проекта
Как НИОКР, создающий софт — инженерный продукт
Попытка сделать проектную организацию с гарантией результата (PMBOK, RUP)
Agile-методы — оригинальный для ИТ альтернативный способ ведения проектов
Продуктовый подход — дальнейшее развитие Agile
Попытки продолжаются до сих пор, несмотря на известные системные причины неудач
Повторялся путь других отраслей — специализация, создание методик, нормирование проектных работ

Эпоха НИОКР: когда компьютеры были большими

Прикладные задачи:

- физические расчеты, управление ракетами и т. п.
- автоматизация для корпораций со стабильными процессами

Системный софт: операционные системы, языки, компиляторы, библиотеки программ, чтобы упростить разработку и выполнение софта

Использование математических моделей как основы для теории

Алгоритмика: Кнут «Искусство программирования» (1968–1973)

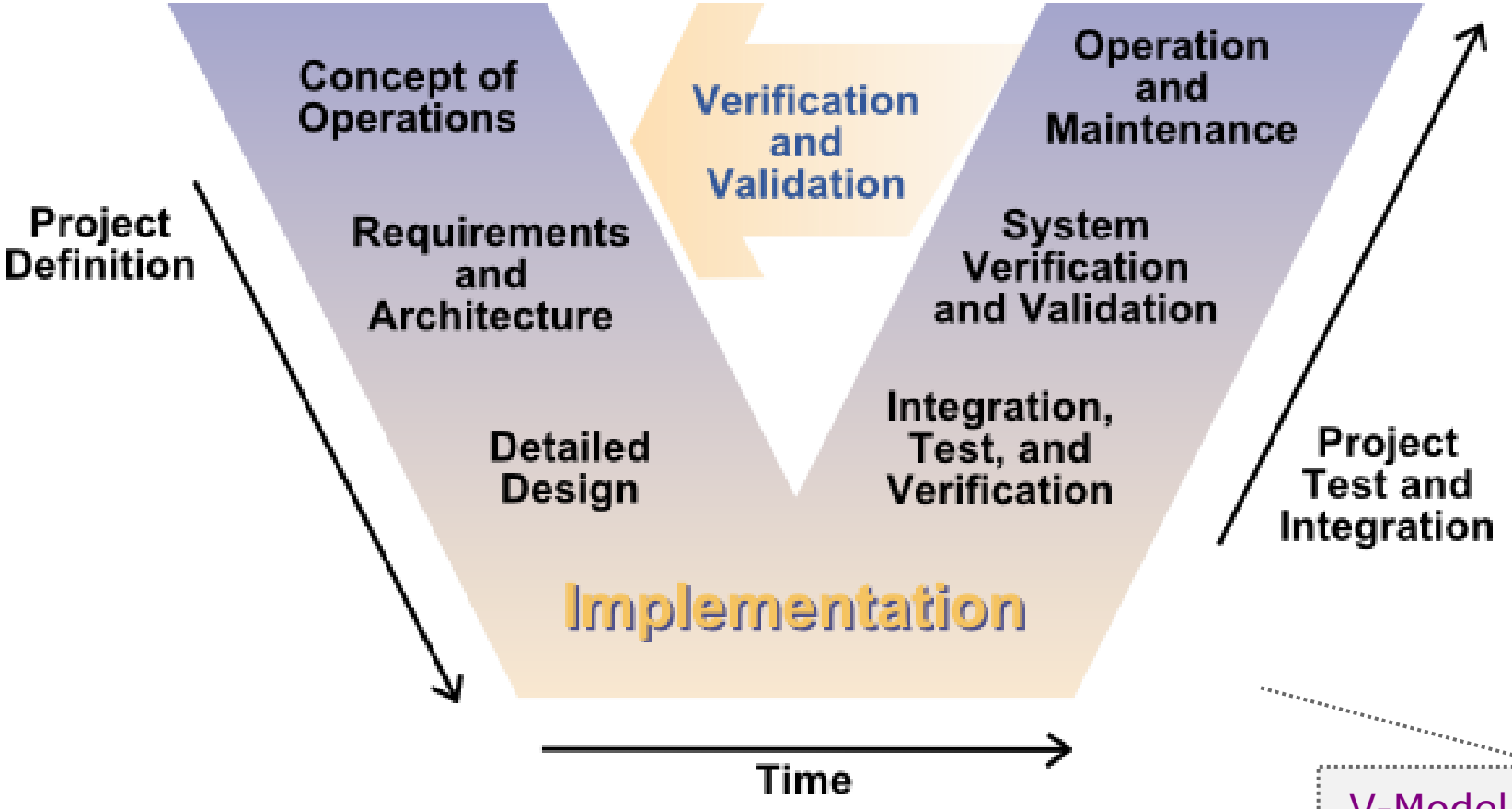
Структурное программирование, чтобы программа была понятной

SQL и **ER-диаграммы** – проектирование и работа с большими данными

Вирт «Алгоритмы + Структуры данных = программы» (1976)

Учимся создавать и развивать большие и сложные системы

Проект представляет V-модель



V-Model (Wikipedia)

Эпоха RUP

Эпоха RUP: укладываемся в бюджет и сроки

PMBOK-3 (2004)
RUP (2003)

Потребность в софте возрастает, разработка — часть бизнес-проекта

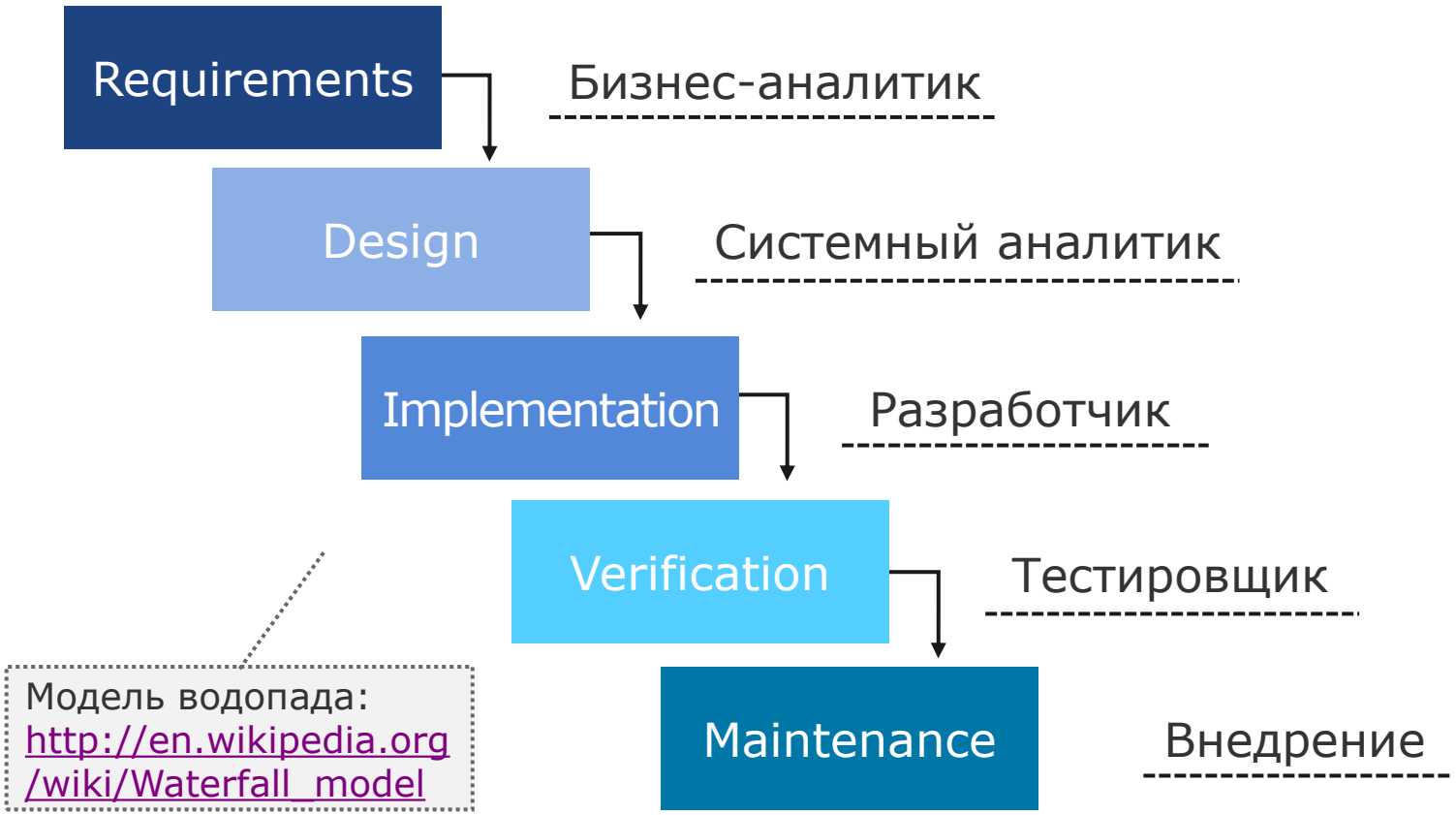
Применим к ИТ-разработке принципы промышленного производства

- Разделим задачу на этапы: проектирование, разработка, внедрение
- Наладим процессы, разделим роли и зоны ответственности
- Технические средства: базовый софт (СУБД, фреймворки) и **ООП**

Получалось не очень

Оценка качества: удалось ли выполнить проект в срок, уложиться в бюджет и достичь ожидаемых результатов

Водопад – специализация по фазам проекта



Каждой фазе — своя роль

Роли выполняются разными людьми или командами

Передача работы — **через артефакты** на отдельных языках

Классические приложения

- Клиент-сервер, на сервере — СУБД, она обеспечивает конкурентную работу, транзакционность и консистентность данных в рамках обработки запроса пользователя
- Трехзвенная архитектура не принесла принципиальных изменений, она позволила писать бизнес-логику на сервере на объектном языке
- Масштабирование — за счет железа и системного софта
- Классическое проектирование: **процедурный** и **объектный подход**
- Разработаны средства визуального проектирования: ER-модель и диаграммы классов, потоков данных, модулей, последовательности и другие

Состав постановки

В теории два уровня:

- требования — описание системы как «черного ящика»
- архитектура и дизайн — описание устройства системы

На практике несколько иначе:

- модель данных — сущности и связи между ними, формально это внутреннее устройство, но, не используя названия сущностей и связей, очень сложно писать требования
- для бизнес-приложений сложно описать алгоритмы обработки данных как «черный ящик», алгоритм воспроизводит действия людей
- интерфейсы: use case в замысле — описание «черного ящика», но попробуйте описывать use case без представления об экранах системы — а ведь это уже ее устройство

Впрочем, интерфейсы на больших компьютерах — командная строка или текстовые экраны, интерактивные приложения — эпоха персоналок

Процедурный и объектный подход: пример

CUSTIS

Задача — интернет-магазин: заказы, оплата, склад, отгрузка, доставка

Процедурный подход

- Таблицы товаров, заказов, платежей, остатков на складе, курьеров, доставок
- Алгоритмы: фиксация оплаты, назначение даты доставки, планирование курьеров
- Интерфейсы: какие экраны, какие данные показываем и действия выполняем

Объектный подход

- Объекты: товар, заказ, платеж, курьер. Доставка — отдельный объект в заказе?
- Алгоритмы: в методах инкапсуляция внутренней логики объектов
- Интерфейсы: витрины каких объектов представляем, какие методы доступны

Если есть доставка самовывозом и курьером, то в процедурном подходе особенности во всех алгоритмах, а в объектном — делаем подтипы

Процедурный и объектный подход

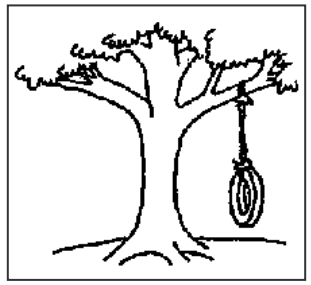
CUSTIS

- Процедурный подход: проектируем структуру БД, интерфейсы и алгоритмы обработки. Иногда — процедуры API backend и API RPC
- Объектный подход: типы и статусы объектов, методы бизнес-логики, интерфейсы в стиле **naked object** — витрины объектов и методы, REST API
- Разница в том, где бизнес-логика — в методах объектов или отдельно
- В реализации может быть анемичная модель транспортных объектов и соответствующие тем же объектам контролеры для бизнес-логики
- **DDD** распространил объектный подход на модель предметной области: вместо словаря мы делаем онтологию понятий и связей, декомпозируя область на фрагменты и применяя методы инкапсуляции, наследования и другие, — концепция **bounded context**

Проблемы RUP

Каждая передача искажает смысл

Что хотел заказчик



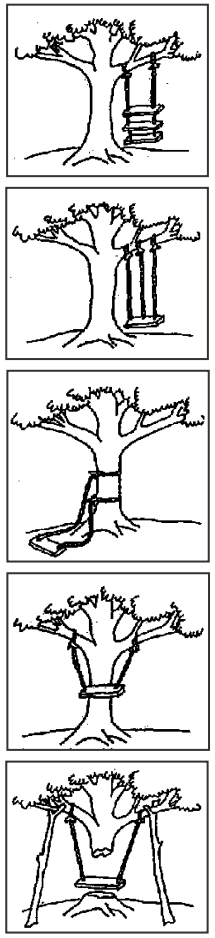
Requirements → Бизнес-аналитик

Design → Системный аналитик

Implementation → Разработчик

Verification → Тестировщик

Maintenance → Внедрение

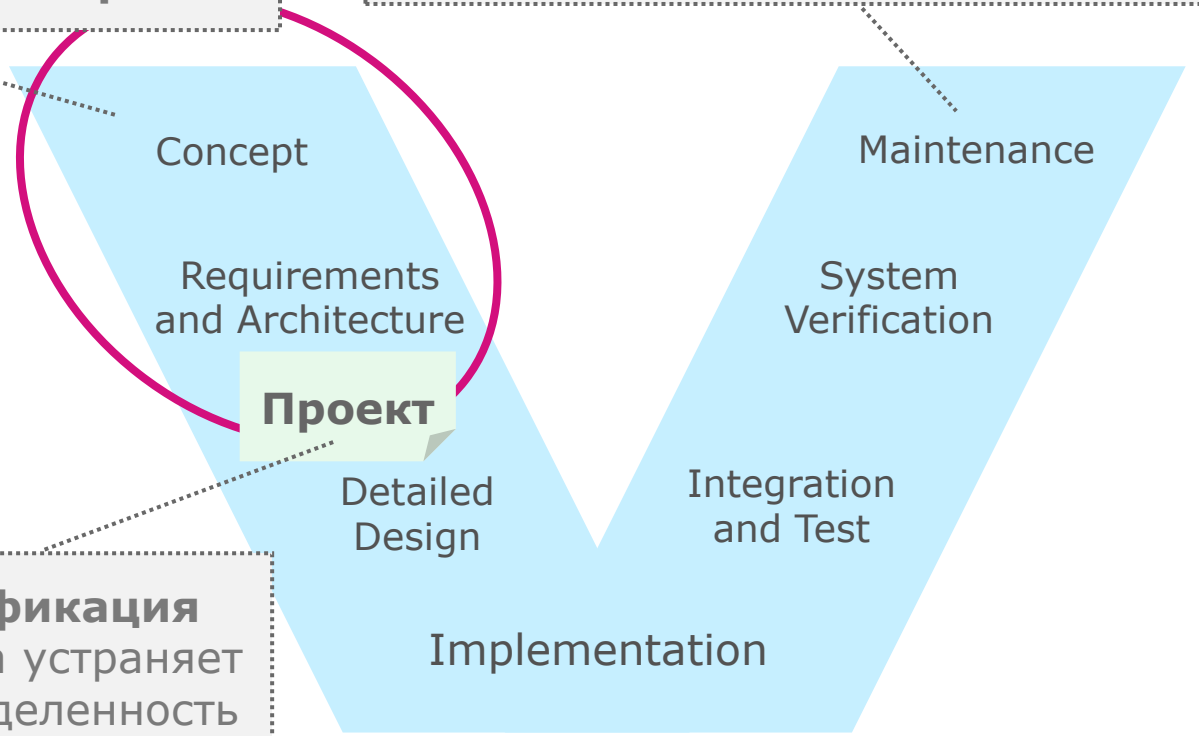


Старый известный образ

Неопределенность — в проектирование!

Проектирование — **НИОКР**, результат **не гарантирован**

Исполнение — **производство**, результат обеспечивается регламентами, включая контроль рисков

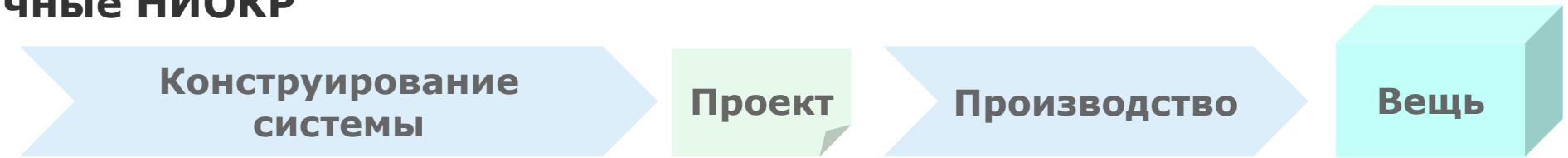


Спецификация проекта устраняет неопределенность

PM **контролирует** результат проектирования, а затем управляет исполнением в проектном треугольнике

Разработка кода – часть проектирования

Обычные НИОКР



ИТ-разработка

Jack W. Reeves. What is software design, 1992 (перевод)

Почему RUP не достиг успеха

Разработка кода — проектирование низкого уровня, а не производство

Технологии только развиваются, незрелые — TRL 4-5. При использовании доводим технлогию внутри разработки: нельзя предсказать трудозатраты и сроки, для предсказания нужны TRL не ниже 8 (опыт Boeing)

ООП не дал методов разработки, которые бы позволили технологично и понятно писать код (хотя и улучшил ситуацию)

Закон Конвея: архитектура софта воспроизводит структуру коммуникаций в команде разработки — организация неотделима, спроектированная конструкция по факту не воплощается

В разработке критичен человеческий фактор, это поняли в 1980-х: Фредерик Брукс («Мифический человеко-месяц», 1975) и Том ДеМарко («Человеческий фактор», 1987). Преодолеть не получилось

Время персоналок

Кризис появления персоналок

Компьютеры стали доступны среднему и мелкому бизнесу

- Потребовалось кратно больше софта, а выпуск разработчиков сохранился
- Процессы в этом бизнесе слабее нормированы, чем в крупном, предметная область является запутанной, а не сложной (Cynefin), нельзя построить модель
- Задача меняется во время разработки, потому что бизнес развивается быстро

Решением стали Agile-методы ведения ИТ-проектов:

- признать, что разработка — это эксперимент и возможны провалы
- ограничить размер эксперимента-итерации, двигаться малыми шагами
- доводить эксперимент до проверки на конечном потребителе
- проблемы коммуникации решить, снизив (или устранив) специализации

Изменения постановок

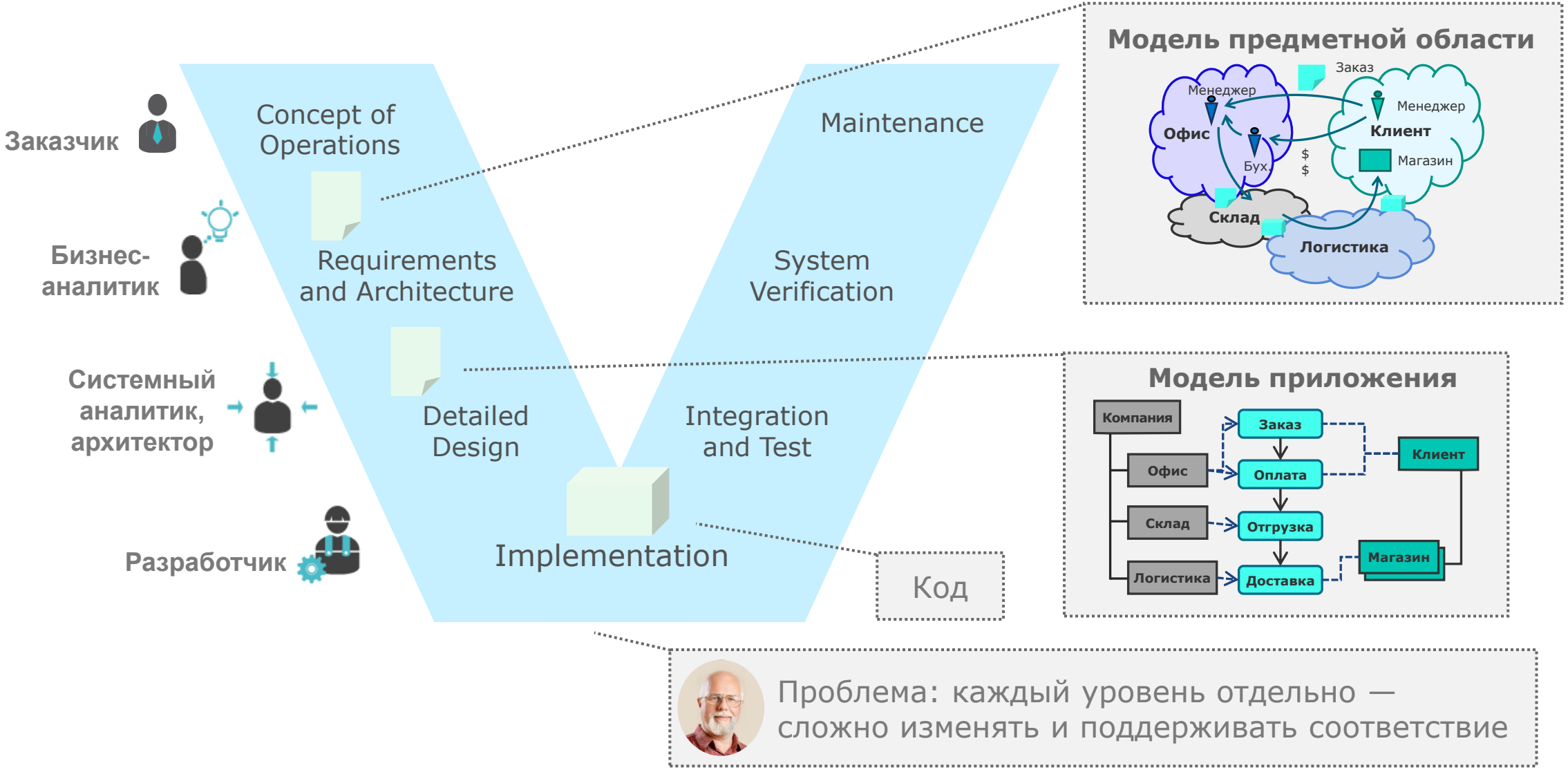
Новые требования к постановкам

- Система проектируется и реализуется инкрементально, а не целиком
- Требования должны быть на языке, отражающем ценность для пользователя
- Но можно не заботиться о точности, так как демо быстро дает обратную связь
- Дизайн делают в процессе разработки и фиксируют по необходимости

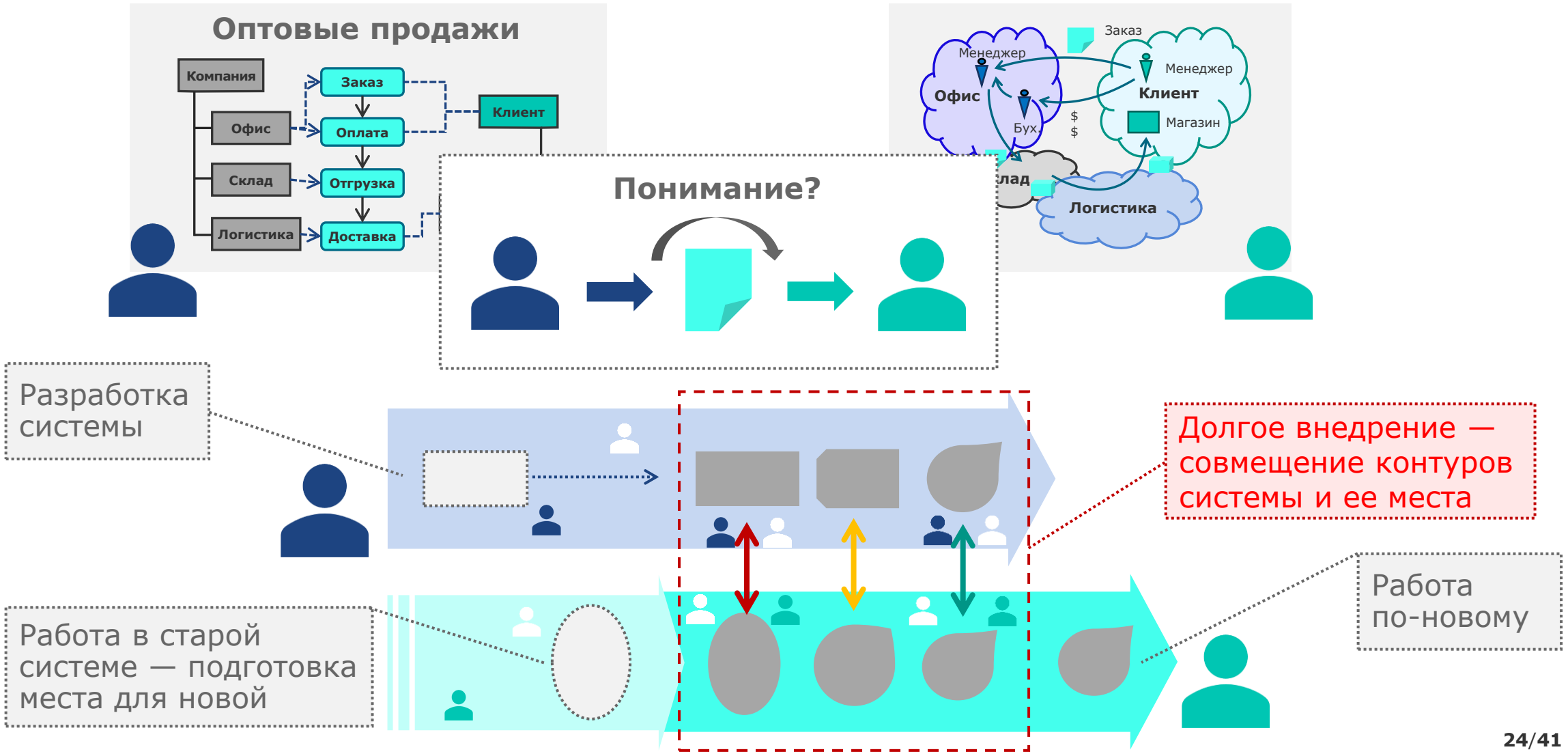
Решения

- Формат **user story** — малые, ценные для пользователя фичи приложения
- Проблема: невозможно оценить трудоемкость без дизайна
- Альтернатива: **DDD**, модель приложения, **понятная заказчику**

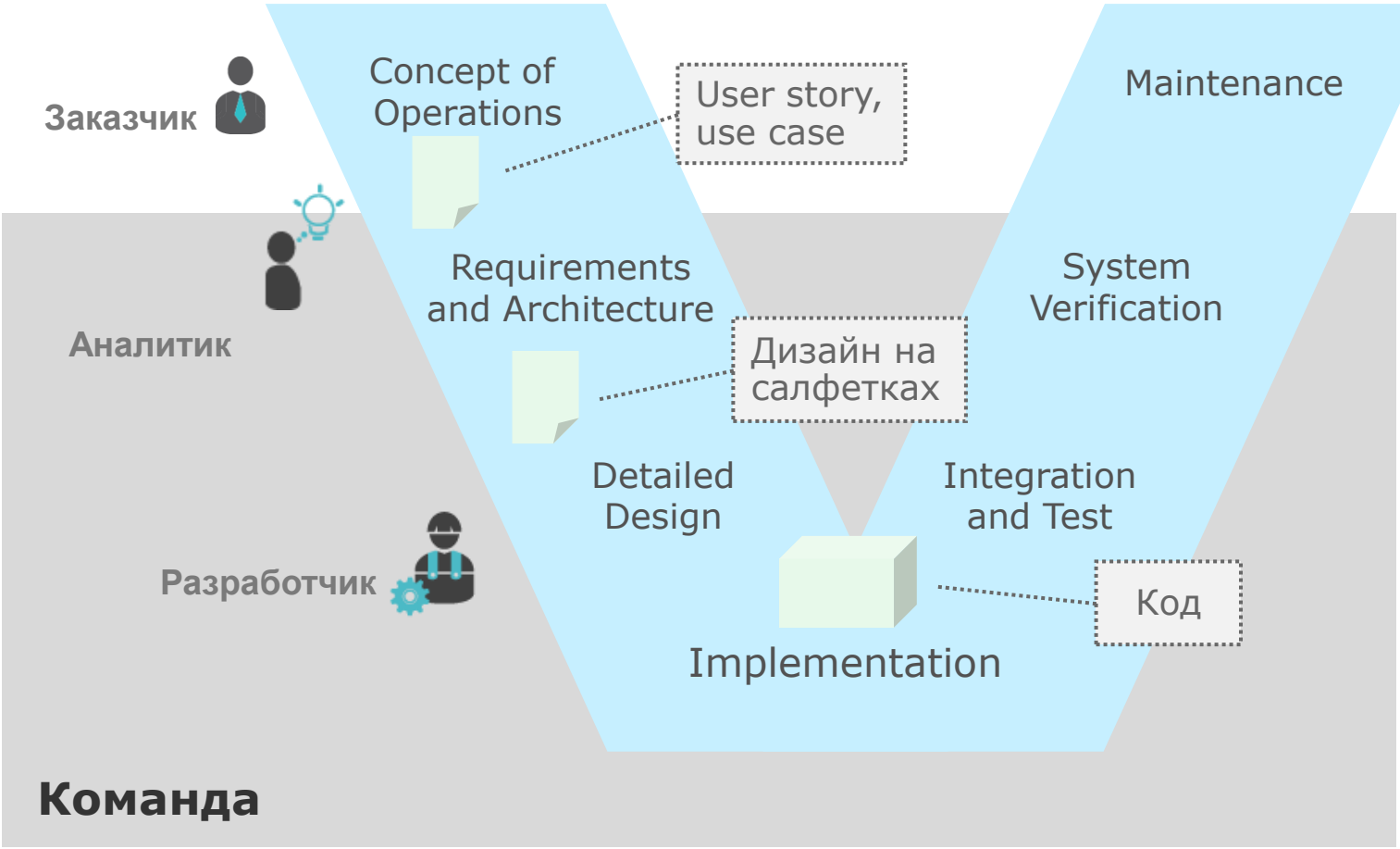
Классический водопад



Проблема двух моделей



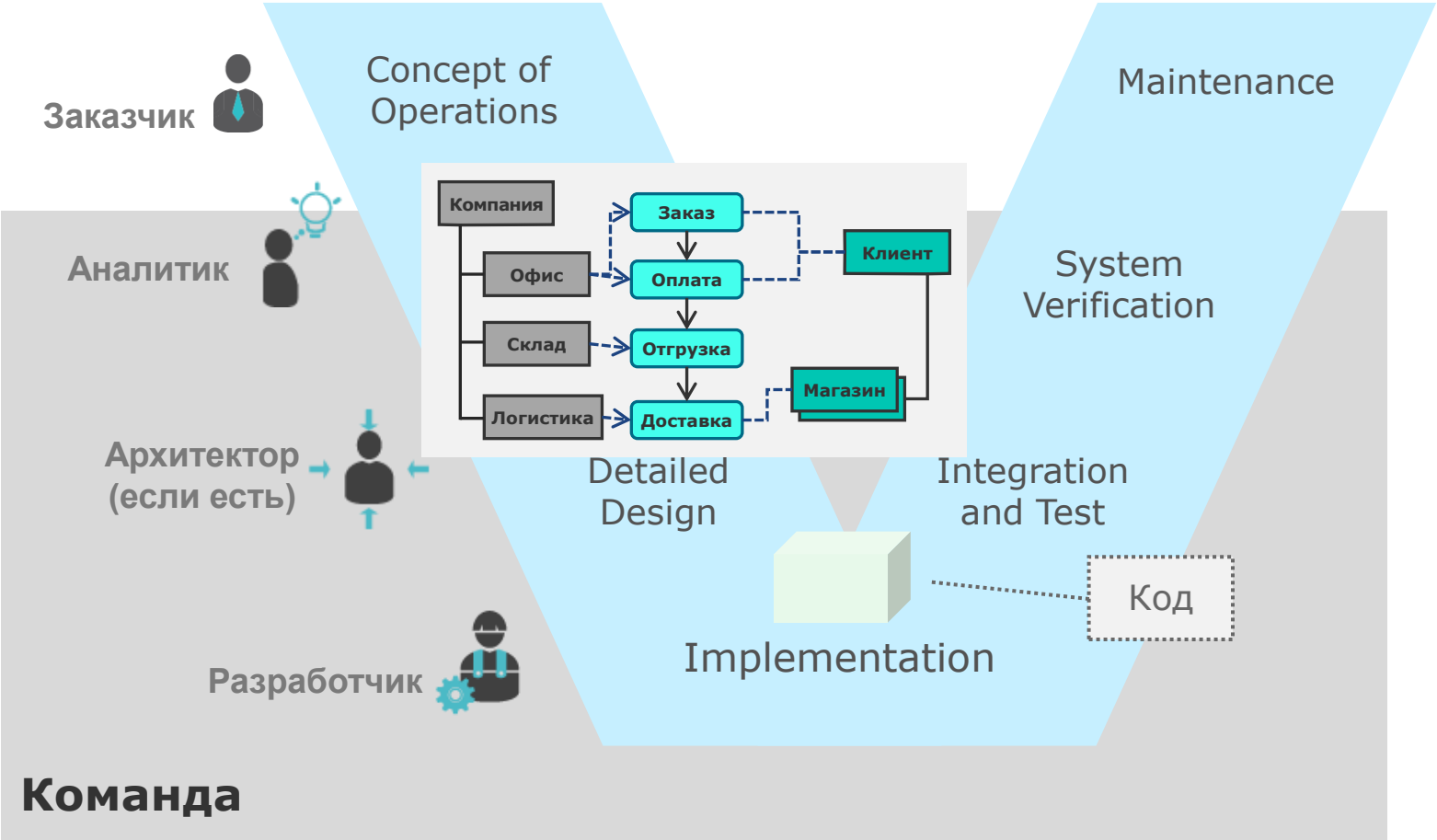
Легкий цикл для каждой фичи



Проблемы:

- нет целостного представления
- реализуемость непонятна

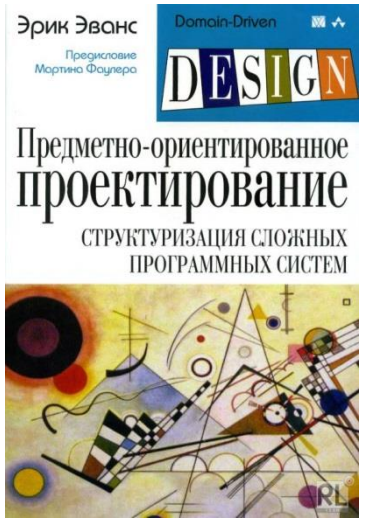
DDD — единые язык и модель, прозрачно отражаются в код



CUSTIS

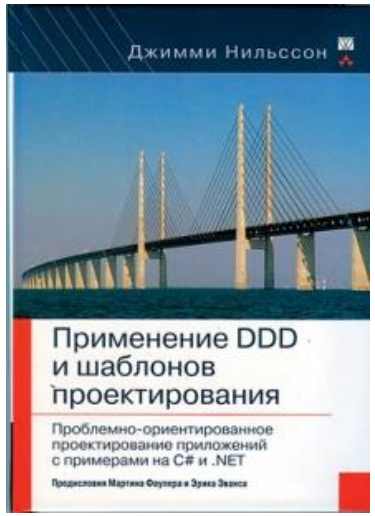
Domain driven design

DDD: источники



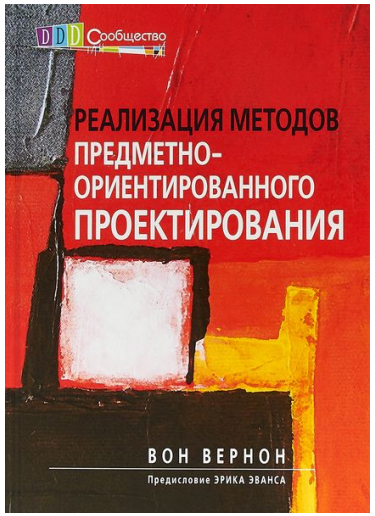
Концептуальная книга Эрика Эванса

- на английском — в 2003
- на русском — только в 2010



Практическая книга Джимми Нильссона

- на английском — в 2006
- на русском — в 2007 (почти сразу!)



Обновление от Вона Вернона

- на английском — в 2013
- на русском — в 2016

Единый язык (ubiquitous language)

CUSTIS

- Построен на основе терминов предметной области
- Его понимают и ИТ-специалисты, и эксперты бизнеса
- На нем описана модель ИТ-системы и ее место в бизнес-процессах

Понятия единого языка: **клиент, накладная, платеж, долг** — из предметной области

Единая модель

CUSTIS

- Аналитик собирает требования и строит модель — сначала предметной области, затем системы
- Артефакты модели описывают систему и ее использование в бизнес-процессах предприятия
- Разработчик реализует модель
- Артефакты модели можно проследить в коде

Модель предметной области становится **моделью системы**

Плюсы единой модели

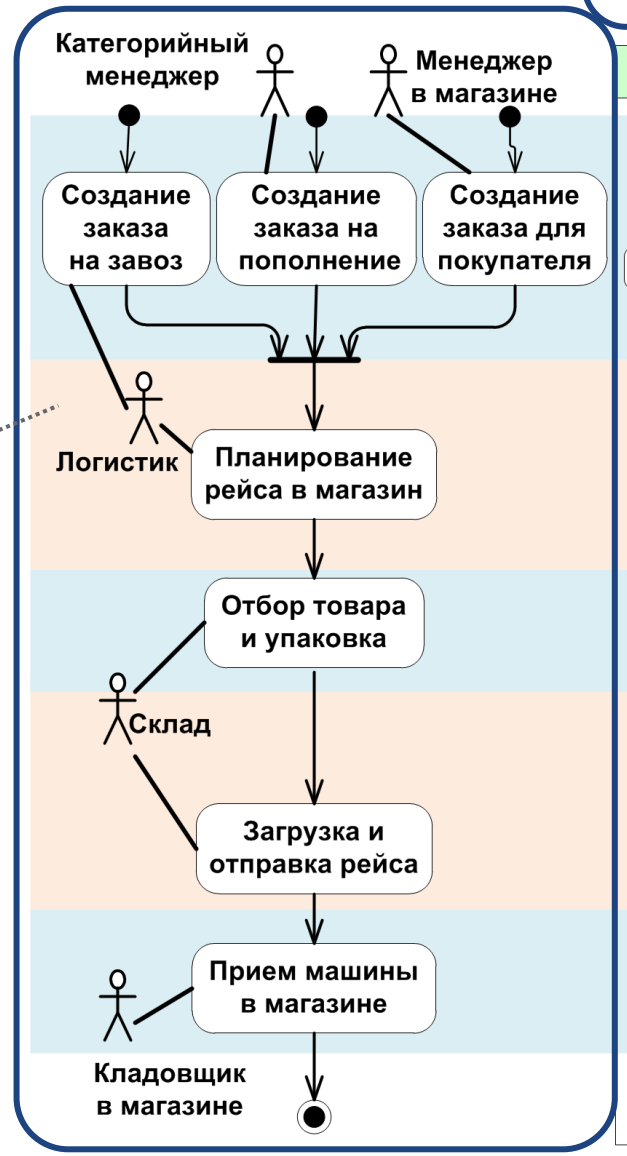
- ☺ Верификация постановок бизнес-специалистами
- ☺ Общее понимание требований на стороне бизнеса
- ☺ Обсуждение модели бизнесом и ИТ, поиск баланса в сложных вопросах
- ☺ Перенос моделей из других предметных областей
- ☺ Бизнес представляет потенциальные возможности системы и сложность различных доработок
- ☺ На этапе эксплуатации — эффективное общение бизнес-пользователей и ИТ без квалифицированных переводчиков-аналитиков

Постановка по DDD

Снабжение магазинов

Выделяем объекты-документы

Бизнес-процесс: диаграмма активности

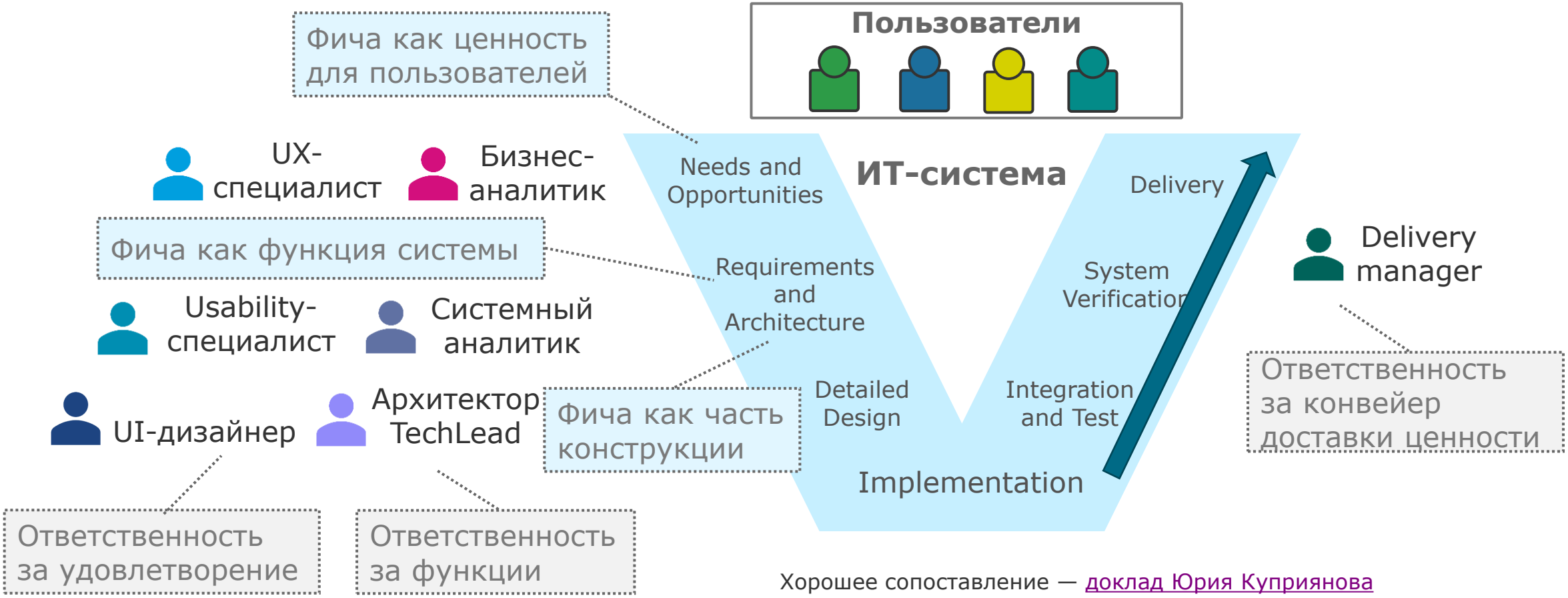


Переходы фиксируют **выполнение** бизнес-функции, а **состояния** соответствуют **передаче по этапам** обработки

Время public web

Проектируем интерфейсы

Стейкхолдеры ставят цель: удовлетворить группы пользователей. Это породило отдельную линейку специализаций и соответствующие учебники



Хорошее сопоставление — доклад Юрия Куприянова «Анализ требований с точки зрения UX» на AnalystDays 2015

Время public web — технологии

Кризис технологий:

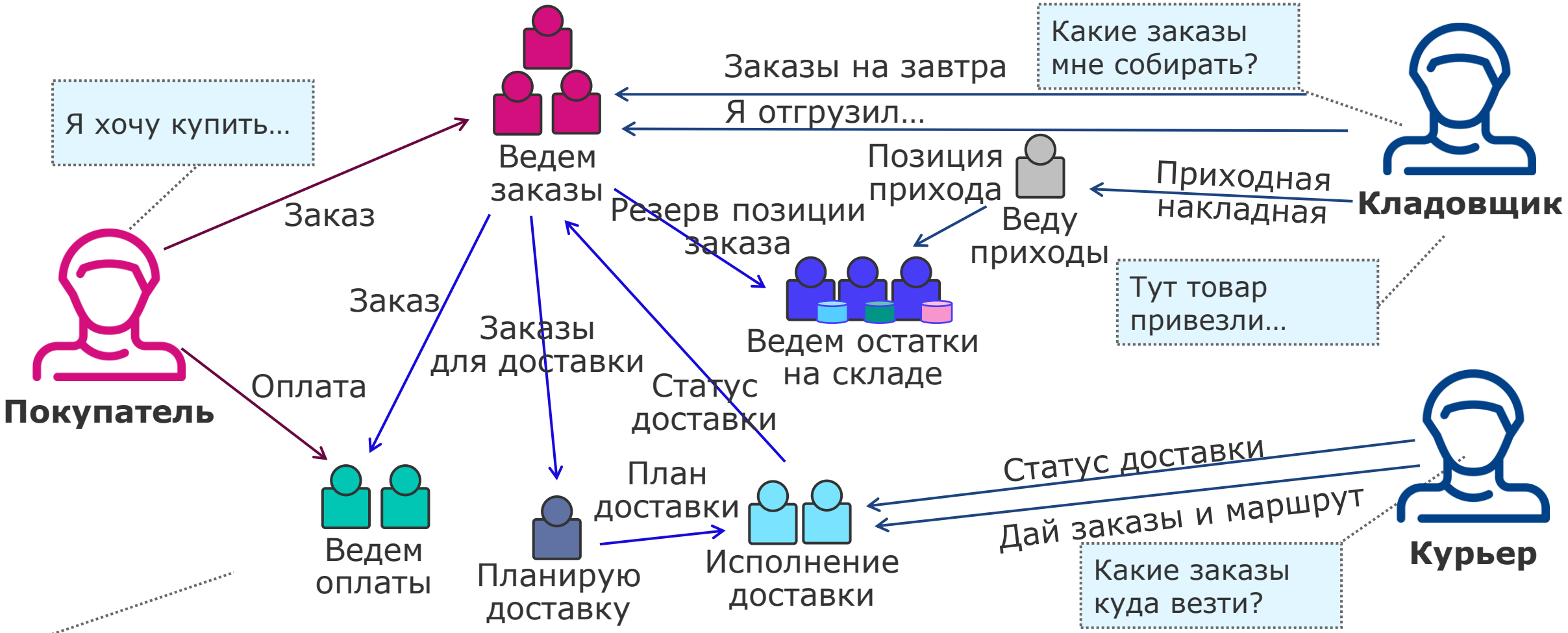
- объектные языки не дали хорошего, устойчивого и понятного кода
- производительности одного сервера недостаточно, распределенную систему не получается сделать базовым софтом, а надо делать в конкретном продукте
- реляционные базы данных не обеспечивают производительность

Технологические ответы:

- переход на микросервисы и акторную модель с оркестрацией и хореографией многих асинхронно действующих агентов
- базы данных NoSQL, множество баз данных для одного приложения без транзакций и консистентности вместо единой реляционной СУБД с SQL
- мультипарадигмальные языки — объектная, реляционная и функциональная парадигма вместе (с C# в 2008)
- конвейер непрерывной поставки продукта (DevOps)

Если работаете через user story — то нет проблем. А вот модель, описывающую устройство такой системы, надо создавать принципиально иначе

Модель гномиков в приложении



Готовых методов нет. Эта модель — моя идея, впервые рассказал на AnalystDays 2020, а эта схема — из доклада на TechLead 2021

Ловушки новых технологий

Новые технологии имеют большой потенциал, но проработаны недостаточно

Стейкхолдеры знают этот потенциал и часто думают, что их использование — как путешествие по хорошей дороге, а по факту это форсирование болота

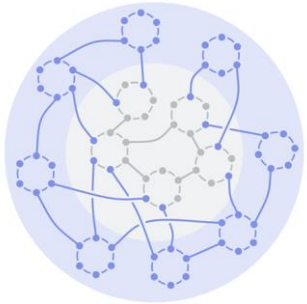
Пара примеров разного уровня:

- чат-боты вместо операторов в службах поддержки, вы все с ними встречались
- делаем мобильное место продавца вместо десктоп-приложения — оказывается, что для печати на ближайшем принтере нет готовой технологии, и в браузере тоже нет. Получается исследовательская задача

Время public web — продуктовое мышление

Софт не поддерживает создание бизнес-продукта, а образует этот продукт, другие активности становятся поддержкой, как в Ozon, Uber и других

При успехе MVP, создаваемого одной командой, число разработчиков увеличивается, получается много полуавтономных команд в сетевой структуре с оркестрацией между ними по графику и продуктам работ



Конструкция фрактальна:

- каждая команда порождает гипотезы о перспективных фичах, проверяет их на пользователях, развивает
- успех гипотез приводит к росту команды и ее разделению по направлениям, а не по специализациям
- на уровне компании так же порождаются направления и создаются команды под них

Рамка работы — развитие социотехнической системы, образуемой продуктом (схема OMG Essence)

CUSTIS

Понимает ли разработчик смысл приложения?

Не понимает

Понимает

Создает

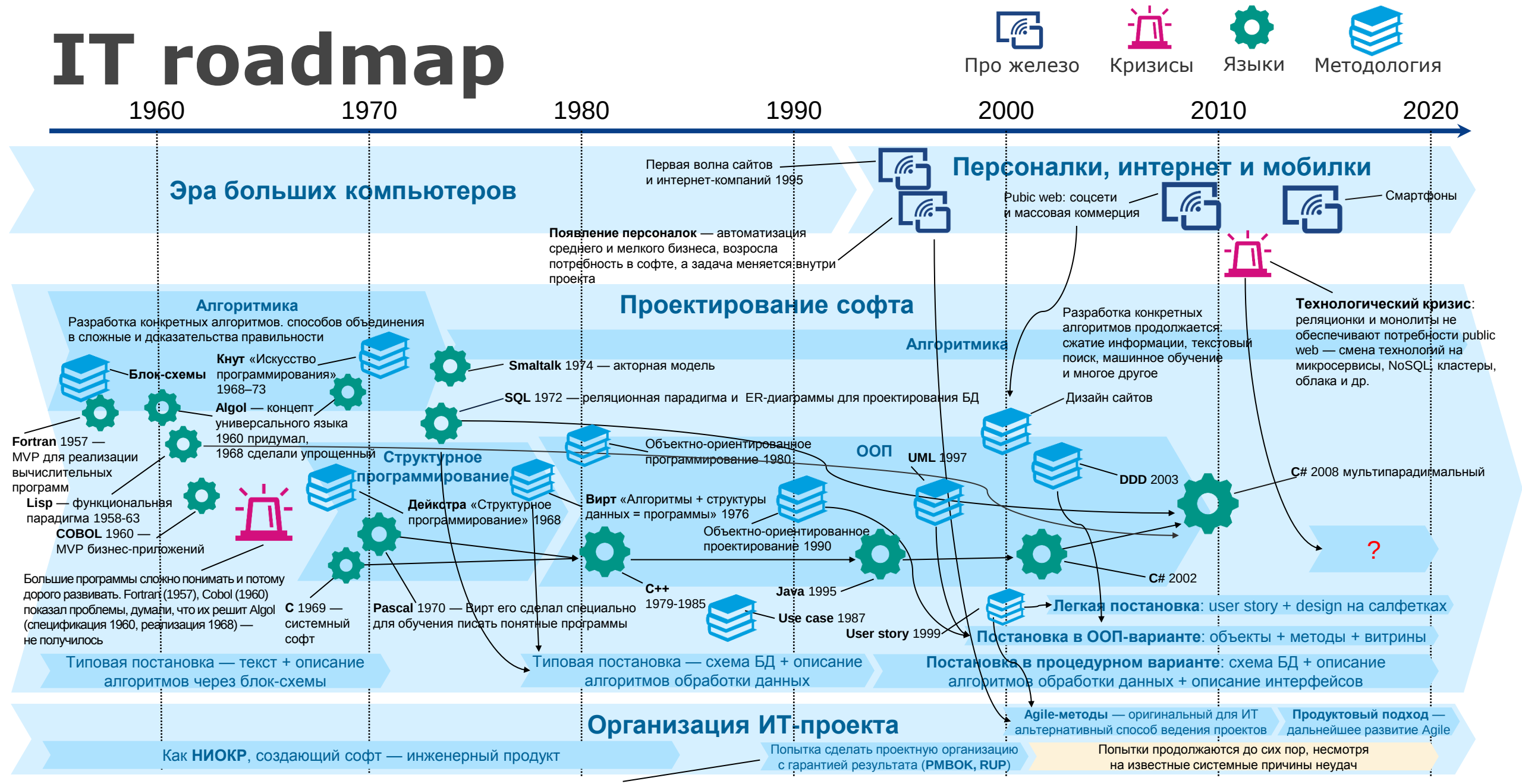
Понимать смысл софта — лишнее, аналитики растолкуют на техническом языке, что сделать

Разработчик понимает цели софта и предлагает технические решения и идеи

Разработчик должен взаимодействовать с заказчиком и понимать его язык, цели и использование софта

Разработчик в команде проекта активно участвует в формировании целей софта

IT roadmap


Про железо
Кризисы
Языки
Методология
1960
1970
1980
1990
2000
2010
2020
Эра больших компьютеров
Первая волна сайтов и интернет-компаний 1995
Персоналки, интернет и мобилки
Pubic web: соцсети и массовая коммерция
Смартфоны
Появление персоналок — автоматизация среднего и мелкого бизнеса, возросла потребность в софте, а задача меняется внутри проекта
Технологический кризис: реляционки и монолиты не обеспечивают потребности public web — смена технологий на микросервисы, NoSQL, кластеры, облака и др.
Проектирование софта
Алгоритмика
Разработка конкретных алгоритмов. способов объединения в сложные и доказательства правильности
Блок-схемы
Кнут «Искусство программирования» 1968–73
Algol — концепт универсального языка 1960 придумал, 1968 сделали упрощенный
Fortran 1957 — MVP для реализации вычислительных программ
Lisp — функциональная парадигма 1958-63
COBOL 1960 — MVP бизнес-приложений
Большие программы сложно понимать и потому дорого развивать. Fortran (1957), Cobol (1960) показал проблемы, думали, что их решит Algol (спецификация 1960, реализация 1968) — не получилось
Smaltalk 1974 — акторная модель
SQL 1972 — реляционная парадигма и ER-диаграммы для проектирования БД
Алгоритмика
Разработка конкретных алгоритмов продолжается: сжатие информации, текстовый поиск, машинное обучение и многое другое
Дизайн сайтов
Структурное программирование
Дейкстра «Структурное программирование» 1968
C 1969 — системный софт
Pascal 1970 — Вирт его сделал специально для обучения писать понятные программы
Объектно-ориентированное программирование 1980
ООП
Вирт «Алгоритмы + структуры данных = программы» 1976
Объектно-ориентированное проектирование 1990
C++ 1979-1985
Java 1995
UML 1997
Use case 1987
User story 1999
DDD 2003
C# 2008 мультипарадигмальный
C# 2002
?
Легкая постановка: user story + design на салфетках
Постановка в ООП-варианте: объекты + методы + витрины
Типовая постановка — текст + описание алгоритмов через блок-схемы
Типовая постановка — схема БД + описание алгоритмов обработки данных
Постановка в процедурном варианте: схема БД + описание алгоритмов обработки данных + описание интерфейсов
Организация ИТ-проекта
Как НИОКР, создающий софт — инженерный продукт
Попытка сделать проектную организацию с гарантией результата (PMBOK, RUP)
Agile-методы — оригинальный для ИТ альтернативный способ ведения проектов
Продуктовый подход — дальнейшее развитие Agile
Попытки продолжаются до сих пор, несмотря на известные системные причины неудач
Повторялся путь других отраслей — специализация, создание методик, нормирование проектных работ

И в заключение

Каждый период развития ИТ формировал подходы к проектированию и ведению проектов со своими учебниками — эту историю я рассказал

Культура компании формируется людьми, которые работали когда-то и работают сейчас, она может отражать их идеал, а вовсе не текущие потребности и ситуацию компании

Стоит понимать истоки культуры и текущую ситуацию и действовать сообразно, поддерживая культуру или изменяя ее

http://mtsepkov.org
Telegram: @MaximTsepkov

На сайте много материалов по анализу и архитектуре, Agile, ведению проектов, управлению знаниями, мои доклады, статьи и конспекты книг

Вакансии аналитиков

Пишите на hr@custis.ru, подходите с вопросами