CUSTIS® ИТ-РЕШЕНИЯ ДЛЯ РАЗВИТИЯ

DDD в современной архитектуре: как отражать модель в код

Максим Цепков
Главный архитектор решений CUSTIS

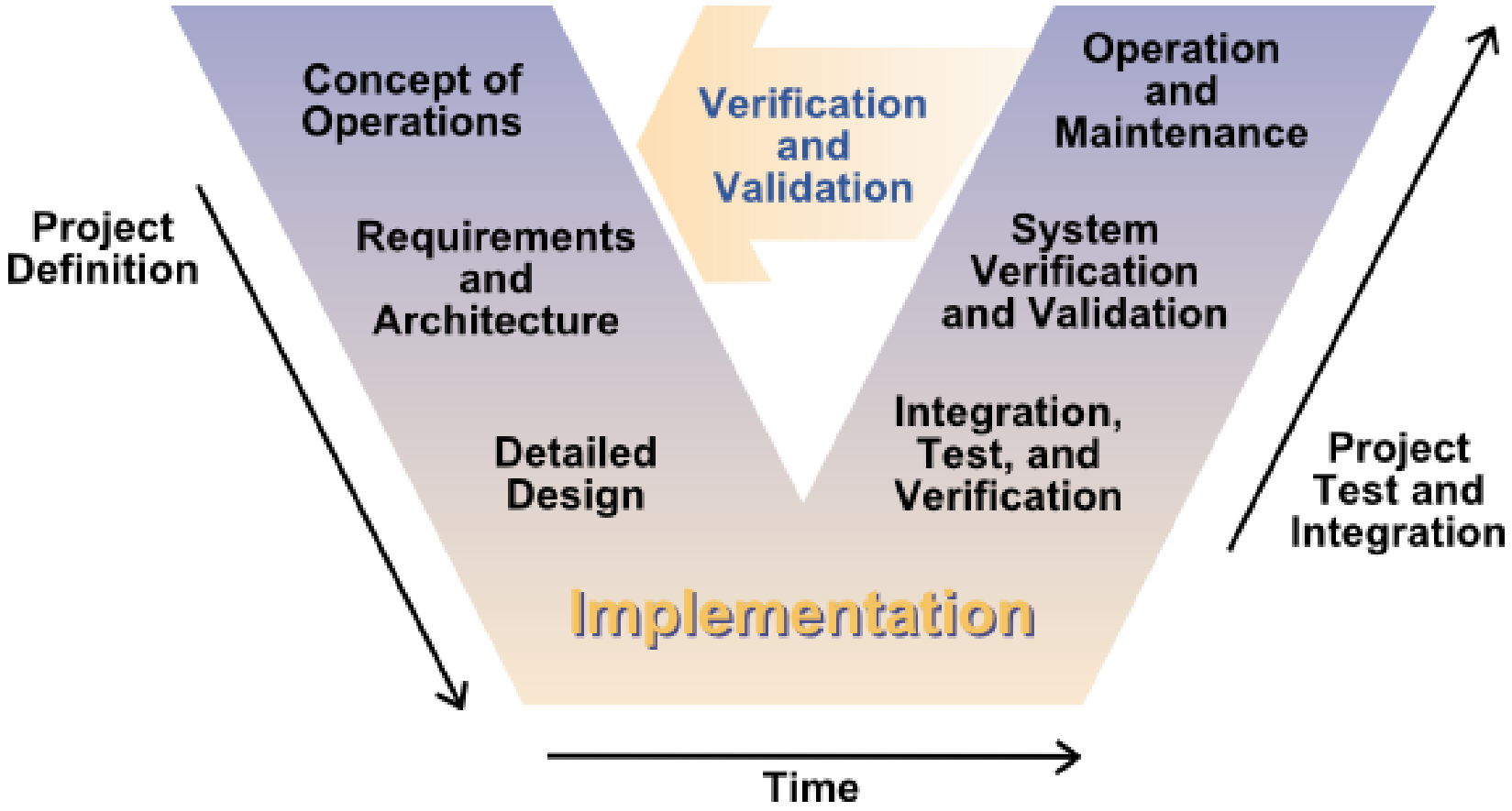
Навигатор по миру Agile, бирюзовых организаций и спиральной динамики

Podlodka Techlead Crew
4 августа 2021

CUSTIS

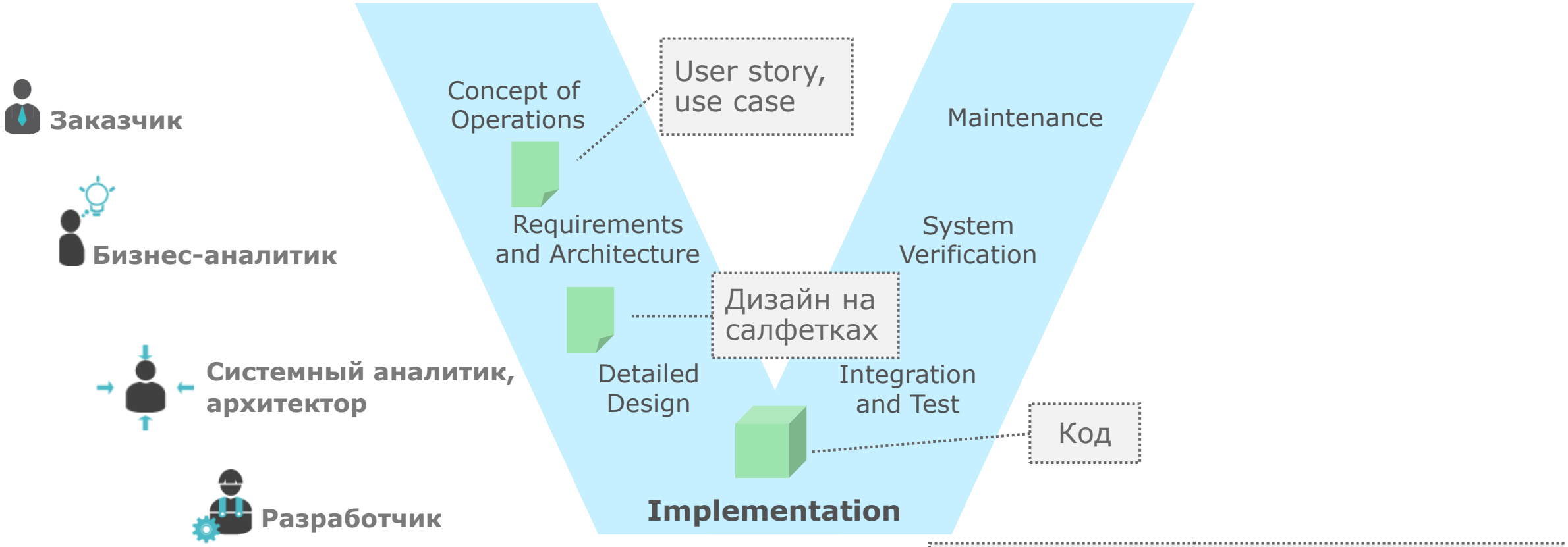
Domain-driven design — что это и зачем?

Путь проекта — V-модель



V-Model (Wikipedia)

Проект можно сделать без моделей



Проблемы:
- нет целостного представления
- реализуемость непонятна

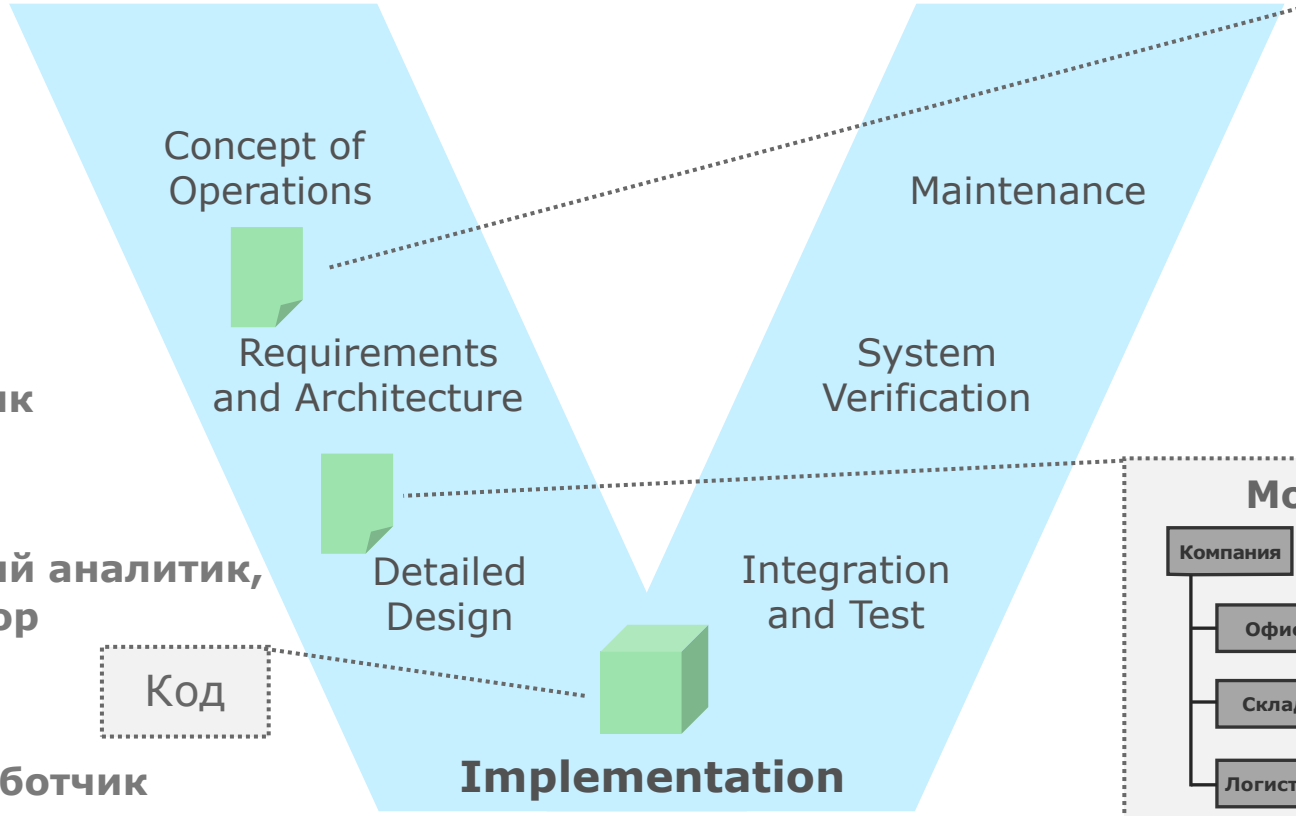
Классическое решение модели

Заказчик

Бизнес-аналитик

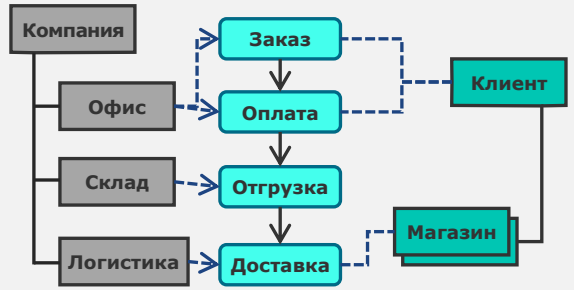
Системный аналитик, архитектор

Разработчик



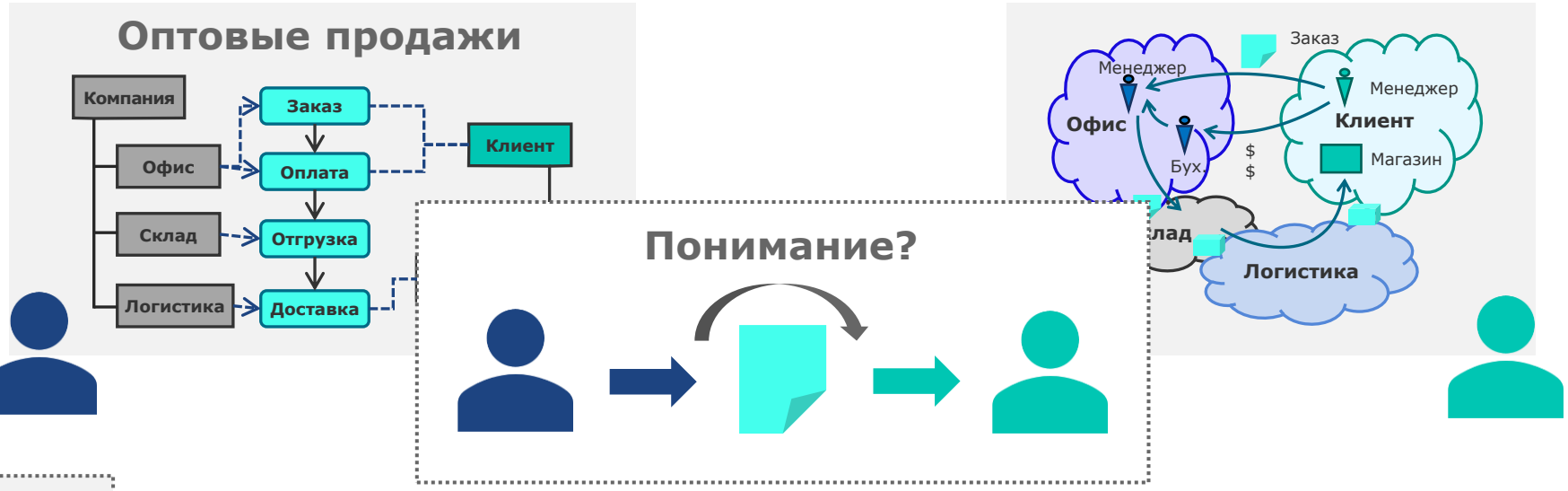
Модель предметной области

Модель приложения



Проблема: каждый уровень отдельно — сложно изменять и поддерживать соответствие

Проблема двух моделей



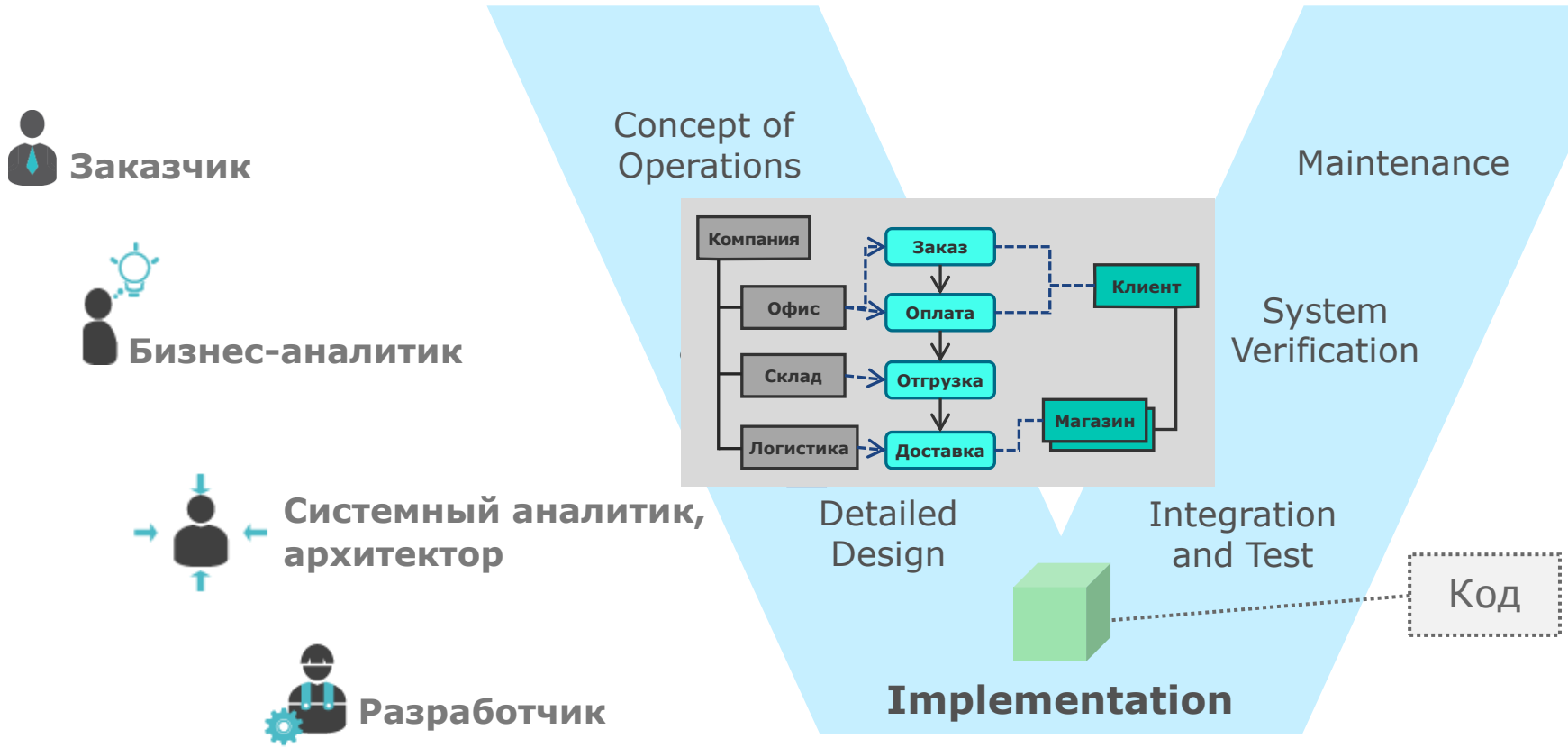
Разработка системы

Работа в старой системе — подготовка места для новой

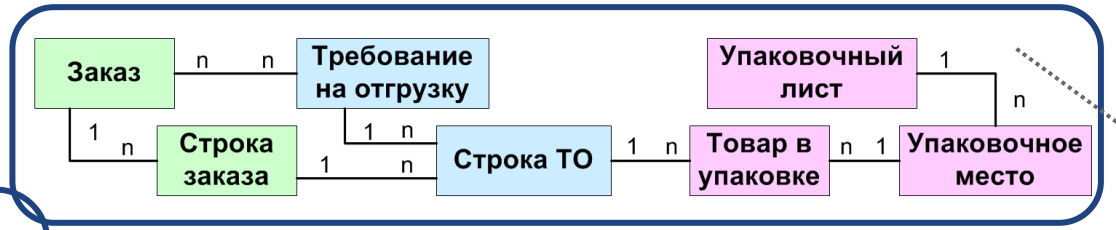
Долгое внедрение — совмещение контуров системы и ее места

Работа по-новому

DDD: единая модель, единый язык, прозрачное отражение в код

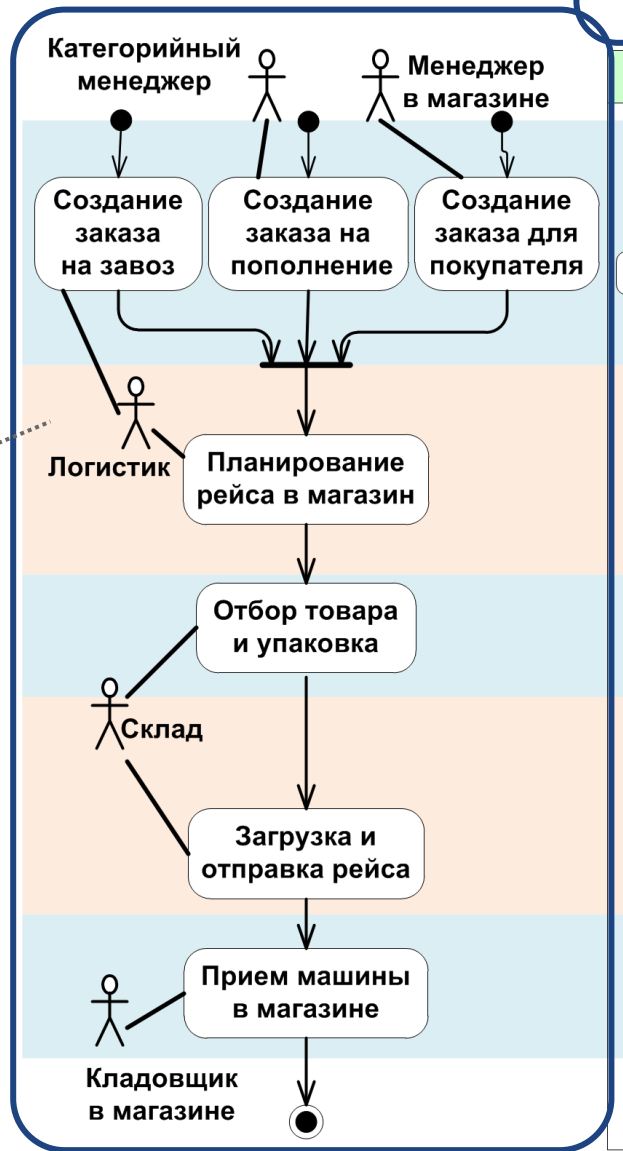


От бизнес-процесса до документов



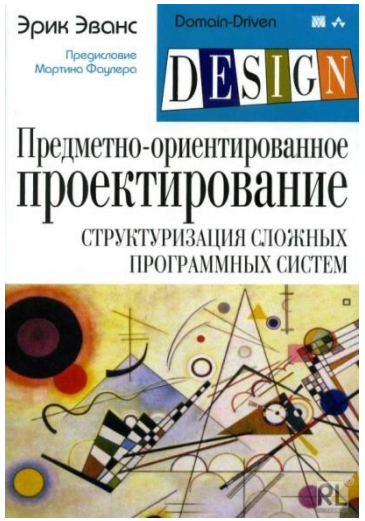
Выделяем объекты-документы

Бизнес-процесс: диаграмма активности



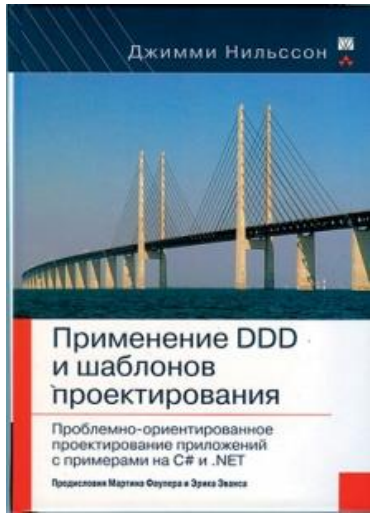
Переходы фиксируют выполнение бизнес-функции, а состояния соответствуют передаче по этапам обработки

DDD: источники



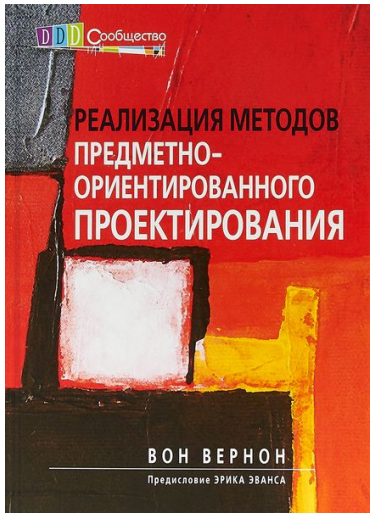
Концептуальная книга Эрика Эванса

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



Практическая книга Джимми Нильссона

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



Обновление от Вона Вернона

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

Единый язык (ubiquitous language)

- 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

- 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

Модель предметной области — модульная

- Предметную область делим на ограниченные контексты
- Для каждого из них — свой единый язык и своя модель
- Целостность удерживает карта контекстов
- Для изоляции и выделения общего в моделях работают методы ООП
- Модели учитывают позицию фирмы: поставки и продажи — разные контексты и модели, хотя сделка одна — оптовая купля-продажа

Подробнее — мои доклады «От монолитных моделей предметной области — к модульным» и «Domain-driven design: от справочников и документов до отчетов»

CUSTIS

Отражение модели в код

Проблемы использования одной модели

- Необходимость понимания модели заказчиком существенно ограничивает моделирование
- Технические подробности и сложные формальные методы становятся недопустимы
- А без них модель не может служить проектом для реализации, нужно дополнительное проектирование

Единственная модель на едином языке —
и преимущество DDD, и источник проблем

Большие и сложные объекты

Бизнес-объект включает все аспекты деятельности

- Клиент — субъект делового мира, партнер по сделкам, взаимоотношения — юридические и управленческие
- Заказ — полный жизненный цикл от начальных переговоров до исполнения, включая юридическое и другое оформление

Бизнес-объекты тесно связаны между собой

☹ При отражении в код объекты получается очень похожими на антипаттерн Big Objects

☹ Сложно понимать, развивать, тестировать

Техническое усложнение модели

- Принципы ООП побуждают к декомпозиции, что увеличивает число объектов
- Применение шаблонов (стратегии и др.)
- Технические и инфраструктурные атрибуты и классы
- Ограничения на объекты со стороны фреймворков и обходные пути для их преодоления

☹ Сложная модель непонятна заказчику, простая — не отражает реализацию

Решение: технологические рельсы

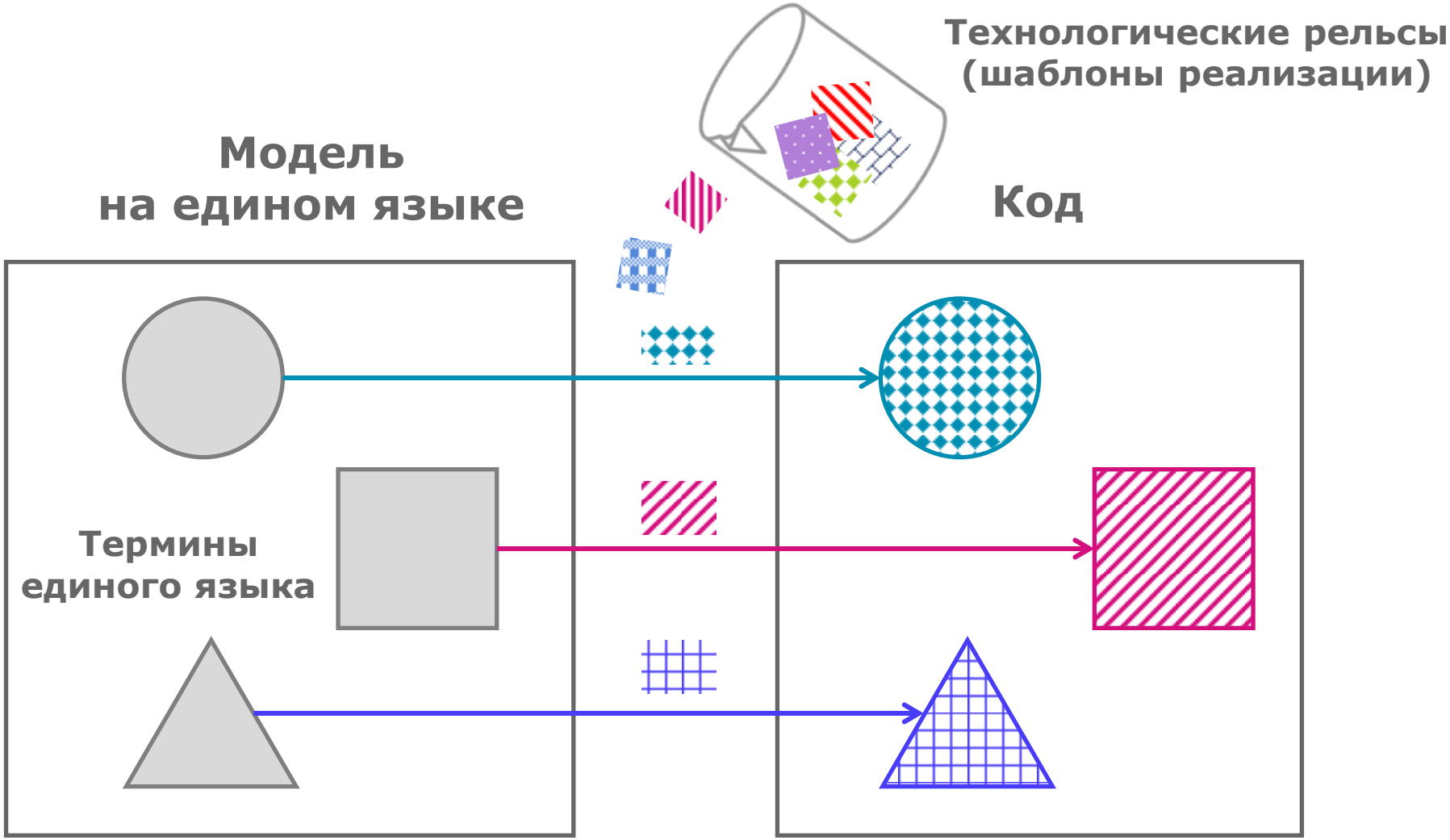
Технологические рельсы — это способ перевода модели на едином языке в код

- Платформы, фреймворки и библиотеки
- Способы их организации в единое приложение
- Способы отражения модели приложения в код
- Типовые задачи и design patterns для их реализации

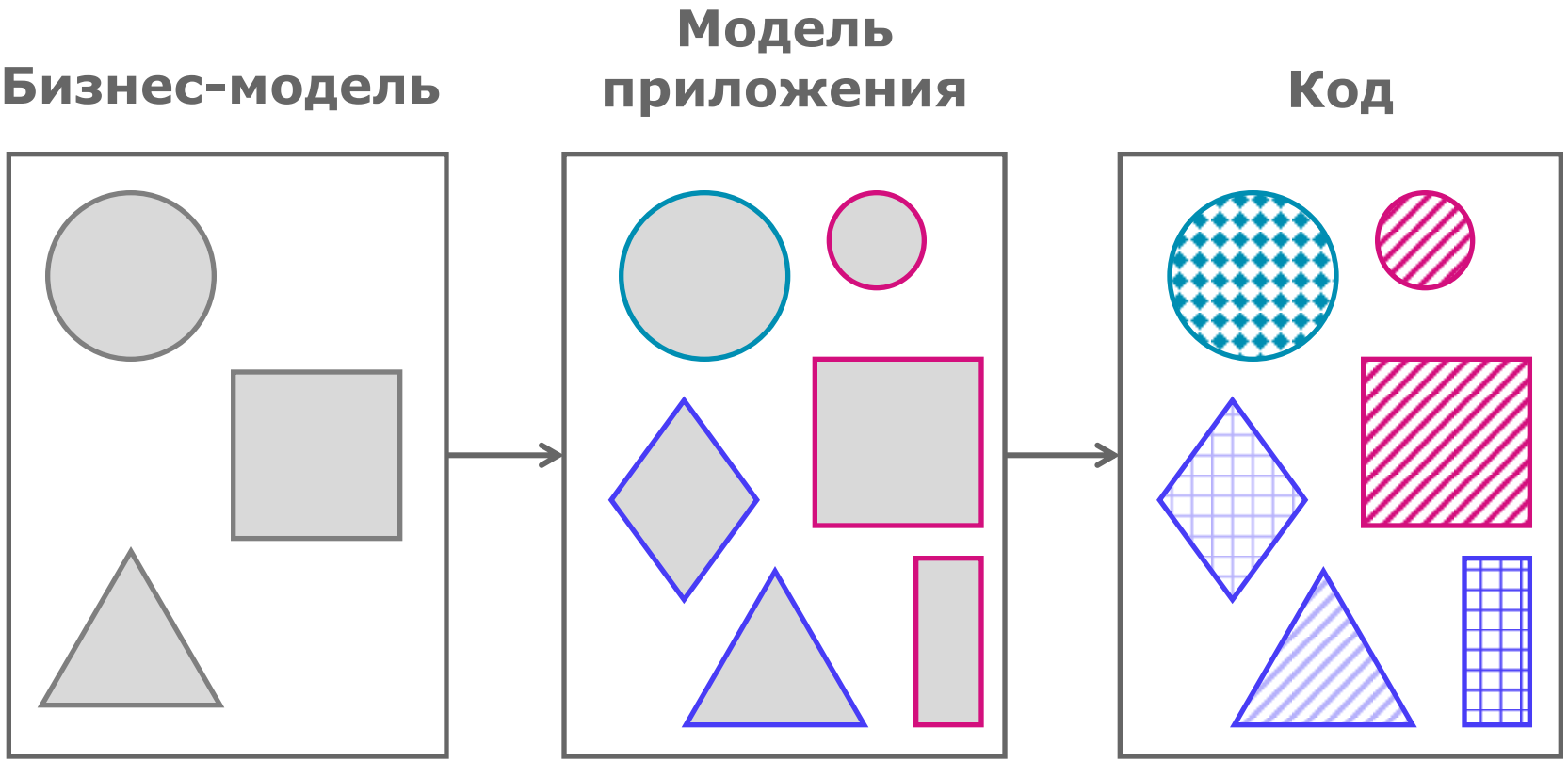
Формулируются на всех уровнях — от архитектуры до частных задач посылки сообщения

«Применяем Domain Model и Rich Objects» — частный случай технологических рельсов

По рельсам ☺


Технологические рельсы
(шаблоны реализации)
Модель
на едином языке
Код
Термины
единого языка

А как без рельсов



CUSTIS

Технологические рельсы для современных приложений

Отражение бизнес-объектов

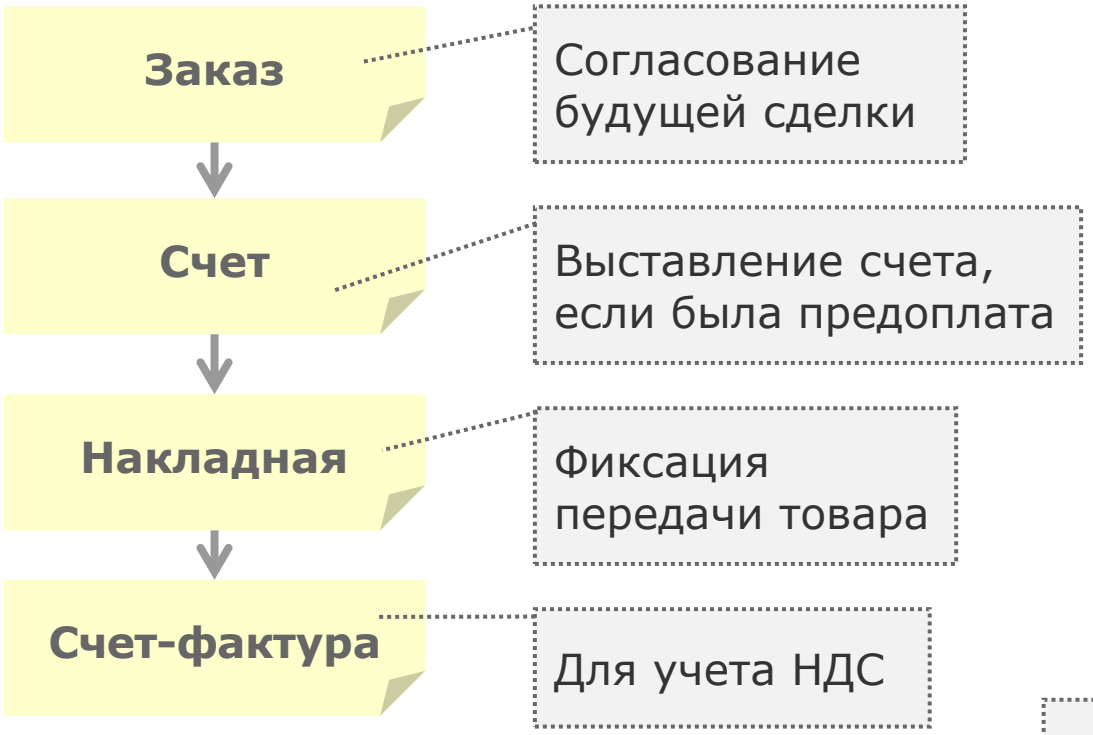
- Анемичный транспортный объект или несколько идентичных
- Прозрачное отражение транспортного объекта в БД (с ORM или без)
- Класс-контроллер бизнес-логики на бэкенд или несколько — для одного объекта, фабрика для создания, для обработки коллекций
- Классы-контроллеры логики для фронтенда и для клиентов

Технологические рельсы

- Генерация всех объектов данных и БД по одному описанию (JSON, XML)
- Решение для workflow на основе state entity
- Библиотека или базовый класс для naked-objects интерфейсов
- API, уведомления, печать и другая инфраструктура — на описаниях

Пример: оптовая торговля

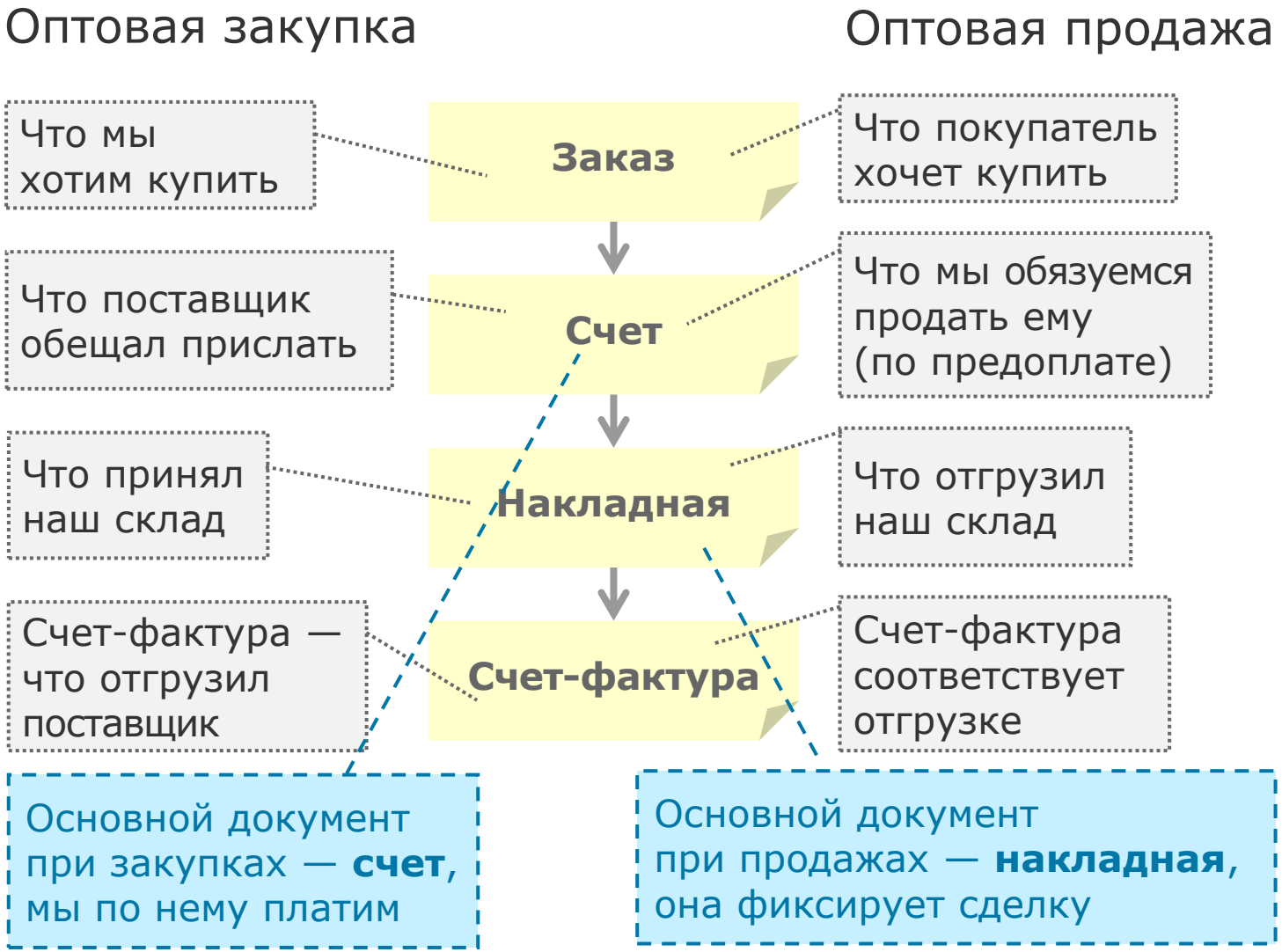
Сделка купли-продажи — нормативный workflow

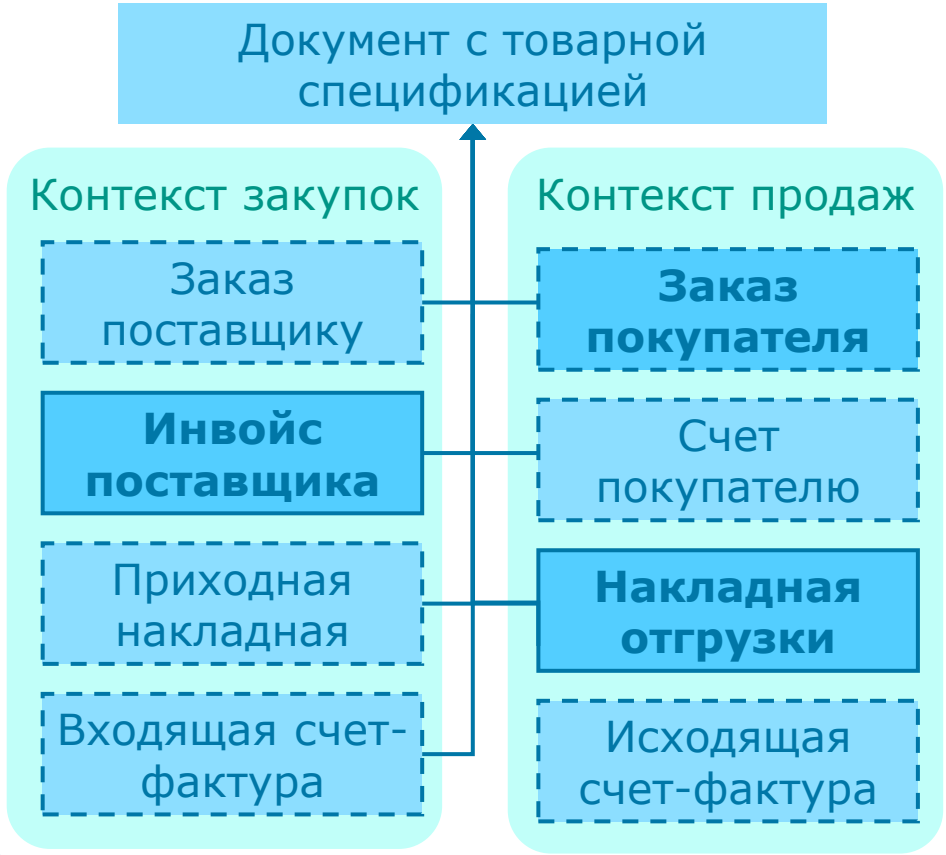


Вариант объектной структуры

Вариант — плохой, не учитывает различие контекстов и позицию компании в сделках

Разница контекстов сделки





Именование объектов

- Бизнес-объекты, их атрибуты и методы называются по-русски
- Для них в модели должны быть зафиксированы английские названия
- Политика: перечисление и/или простые справочники
- Все названия в коде — на основе английских названий в модели
- Надписи в интерфейсе — на основе русских названий в модели
- Технологичная поддержка для переименования объектов и атрибутов
  - Смена русского названия (полу)автоматически отражается в интерфейсе
  - Смена английского названия через (полу)автоматический рефакторинг
  - Регламент для изменения — преобразование перечисления в справочник
- Однородное решение для получения пользовательских названий экземпляров объектов, используемых в сообщениях и логах

Workflow на основе шаблона state entity

- Принцип: единственный атрибут документа отражает его состояние в workflow, возможные действия над ним и права на их совершение
- Граф переходов отражает возможные изменения состояния
- Граф переходов отражает реализацию документом бизнес-процесса, согласован и понятен бизнесу
- Реализация собственным движком или библиотекой типа Activity
- Интерфейсы единообразно опираются на состояния документов, включая проверку на интерфейсах и логику доступных действий

Джимми Нильссон разбирает четыре способа реализации workflow. Я считаю state entity наиболее технологичным, см. мой доклад «Domain Driven Design — от требований до кода»

Выделение базовых абстракций

Для разработчика логично выделение базовых абстракций

- Справочники — объекты без workflow, но с выделением черновиков и устаревших
- Документы — объекты с workflow, реализующие бизнес-процесс
- Классы документов: «договор», «документ с товарной спецификацией» и т. п.
- Обобщенная печать для любых документов
- Обобщенная интеграция для любых объектов и т. п.

Для бизнеса эти абстракции не существуют, он мыслит реализацию объектов как не зависящих друг от друга

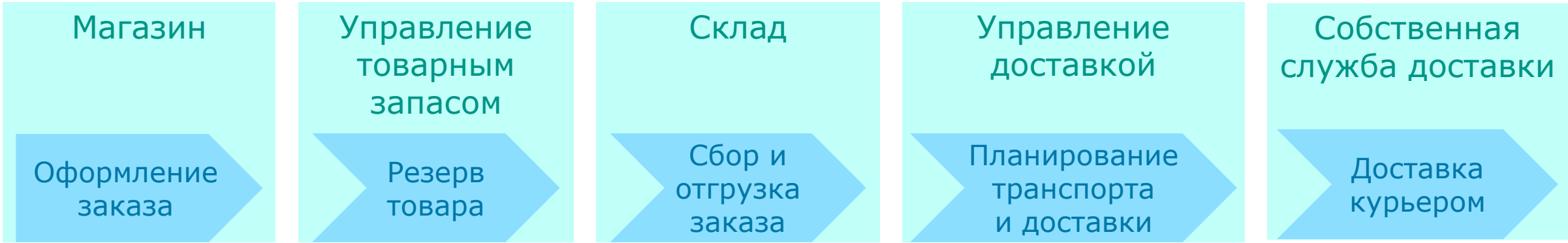
☺ Базовые абстракции можно выделять как инфраструктурные объекты

☹ Проблема: вынесенное в базовый объект ограничивает изменения

Пример: если печать ТОРГ-12 или счета-фактуры реализовать в классе «Документ с товарной спецификацией», то контексты закупок и продаж оказываются связаны в инфраструктурном объекте

Гибкость инфраструктуры

Пример: продажа с доставкой в розничной торговой сети



Инфраструктура интеграции поддерживает движение объекта по бизнес-процессу

- В одних системах может быть общий анемичный объект «Заказ покупателя», в других он преобразуется на входе и выходе из собственного представления
- Универсальное представление в шине дает узкое горло
- Надо уметь добавить к заказу «план проезда» — скан рисунка покупателя, который нужен только курьеру

Единство кода для бэк, фронт и интеграции

Логика, которая присутствует в нескольких слоях приложения:

- Бизнес-логика доступности действий на формах и проверка прав
- Бизнес-логика проверки значений для атрибутов и удобного ввода

Проблемы, требующие технологичного решения:

- Как обеспечивается соответствие логики в разных слоях — вручную, одинаковым языком реализации, генерацией на основе метаописаний или конфигураций, как-то иначе
- Как поддерживается работа бэка с многими версиями фронта и API

DDD и сервисная архитектура приложения

Предметная область разбивается на ограниченные контексты

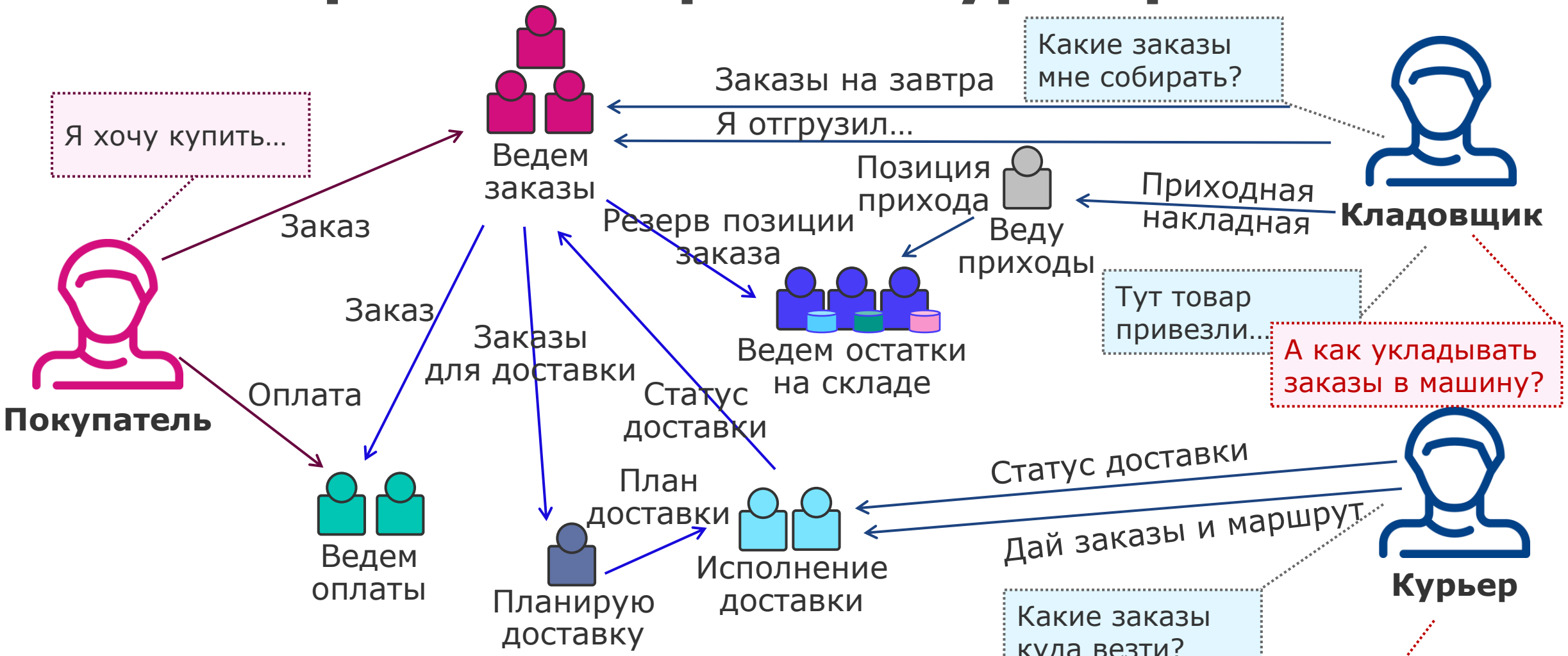
Каждый контекст реализуется одним или несколькими сервисами, возможен композит — несколько сервисов и общая БД

- Ведение покупателей может быть включено в ведение заказов или отдельно

Не все сервисы основаны на объектной модели

- Примеры: учетные сервисы ведения остатков, сервисы работы с расписаниями

DDD и сервисная архитектура приложения



Из доклада «Визуальное проектирование масштабируемых приложений»

Что же достигает DDD?

Единый язык + единая модель:

- Надежная основа коммуникации всех участников проекта при принятии решений
- Успешно заменяют мелкую россыпь требований
- Позволяют эффективно развивать сложную систему

DDD в современной архитектуре требует более сложного отражения в код, чем для монолитов, но технологические рельсы решают задачу

Максим Цепков

http://mtsepkov.org
maks.tsepkov@ya.ru

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