SECON’14

Domain Driven Design
от требований до кода

Максим Цепков

www.facebook.com/mtsepkov

О чем этот доклад?

- Есть разные способы ведения разработки
- Выбор конкретного – зависит от проекта

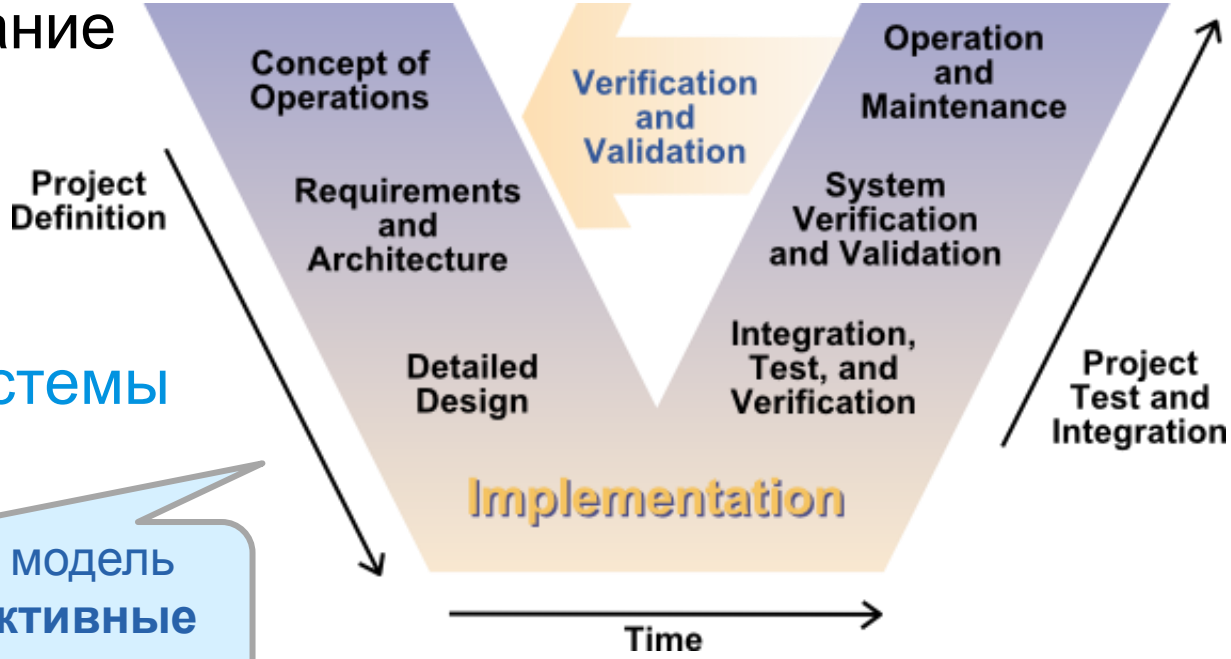
- В сложных проектах уместна работа с моделями
- И DDD – наиболее эффективный способ для этого

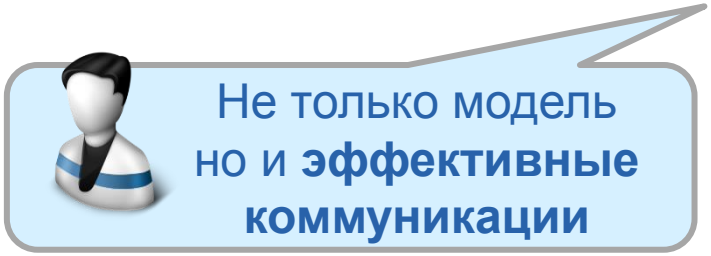
DDD – ключ к построению сложных систем
и их развитию вслед за потребностями бизнеса

DDD – на полном жизненном цикле

- Требования
- Проектирование
- Реализация
- Внедрение

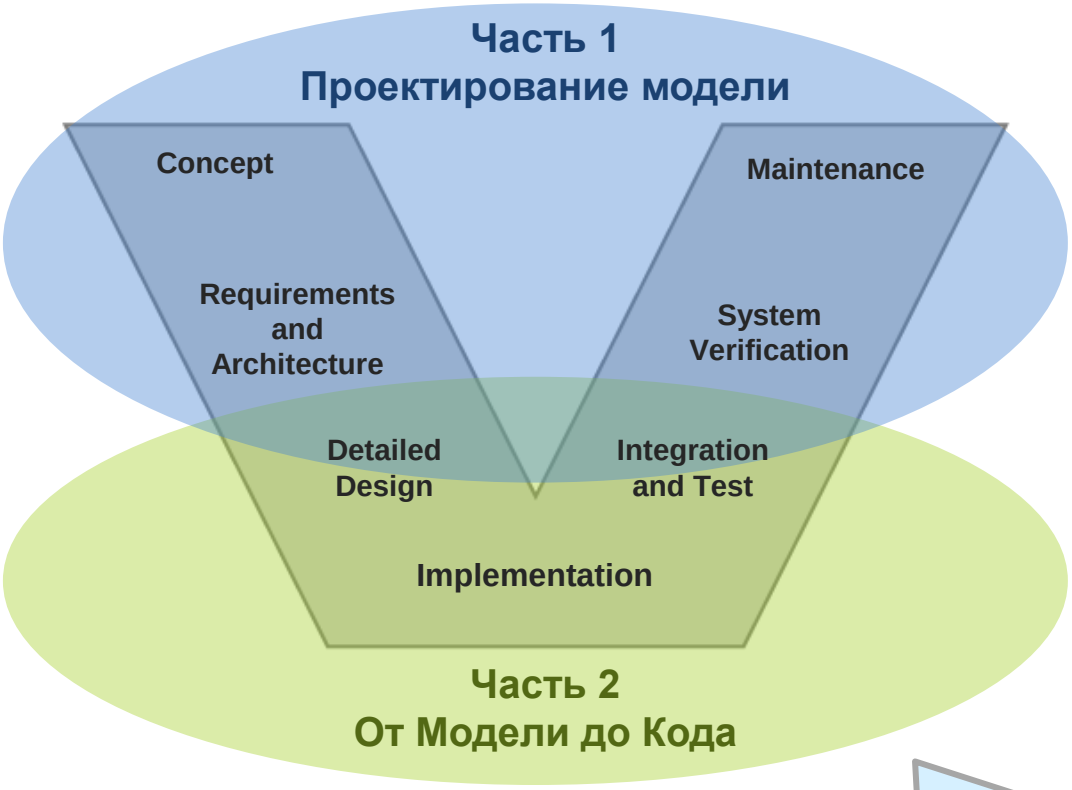
- Развитие системы



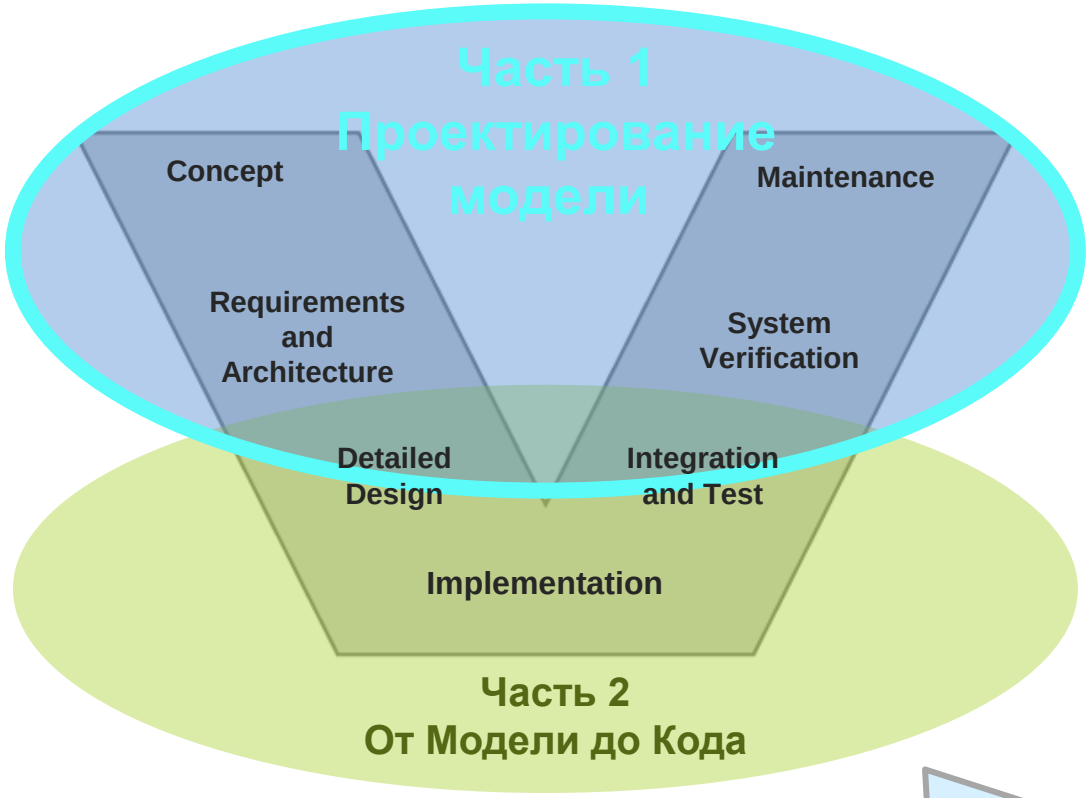


http://en.wikipedia.org/wiki/V-Model_(software_development)

Схема доклада



Часть 1. Проектирование модели



Проектирование в DDD

Теория

DDD – единый язык проекта и одна модель системы

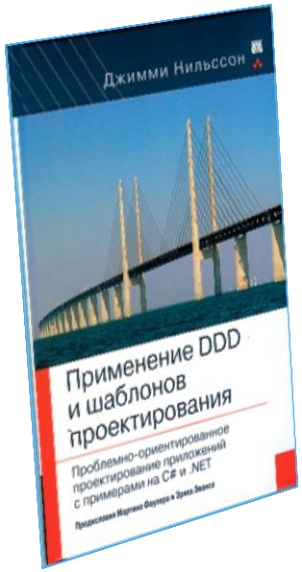
- Модели были давно, но две: бизнес-область и система
- Единый язык проекта создает общее поле понятий
- И позволяет работать с одной, общей моделью

Практика

- Единый язык и построение Модели в конкретных примерах
- Практики проектирования – на этап бизнес-анализа

DDD – как оно начиналось

- 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!)

О едином языке

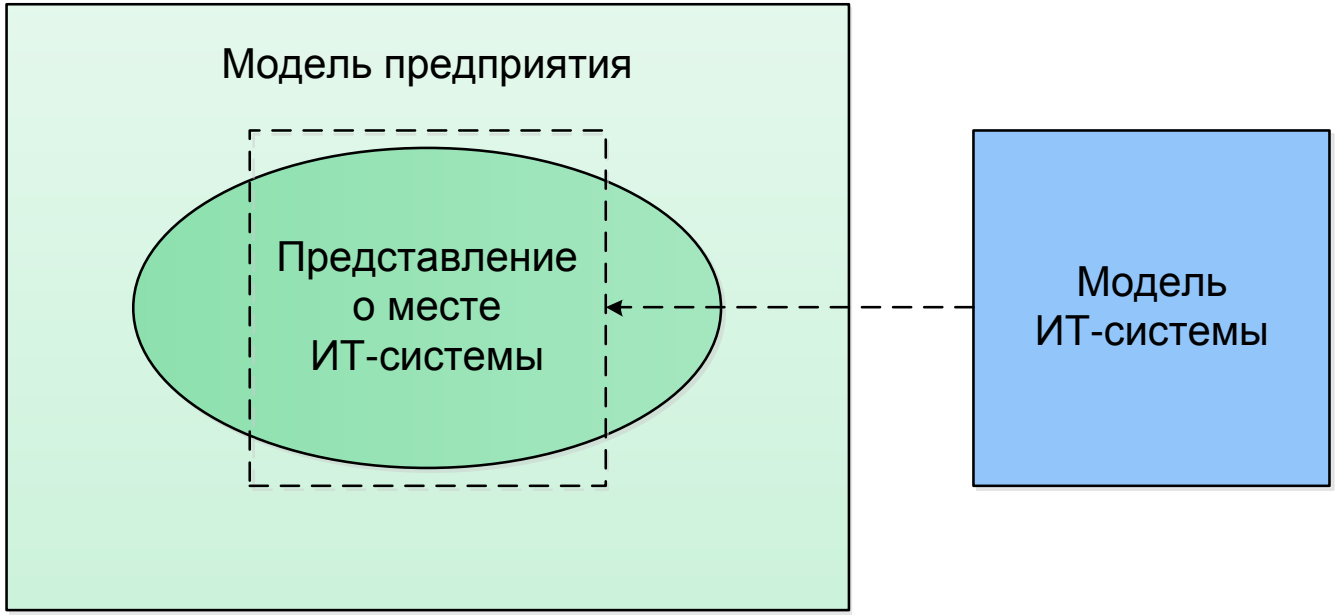
Основная идея DDD

Единый язык (ubiquitous language)

- Построен на основе терминов предметной области
- Его понимают 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

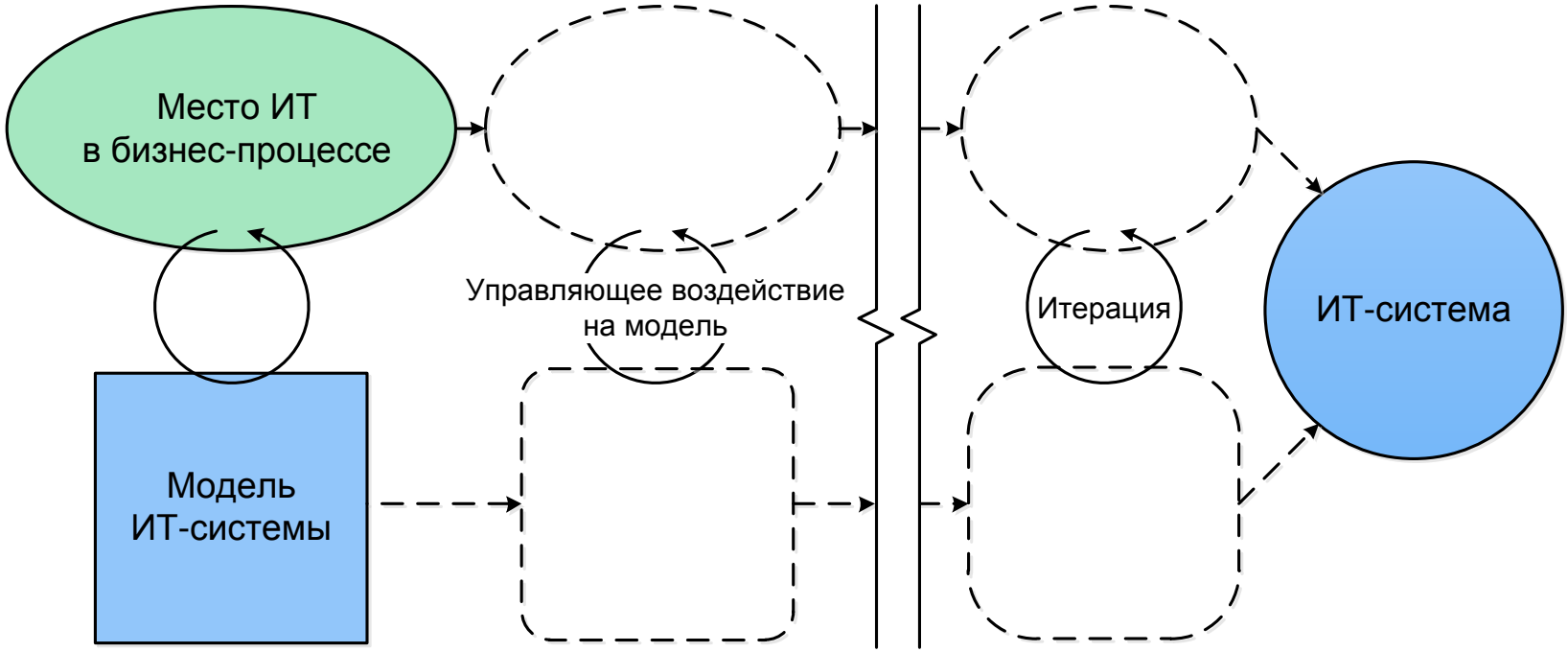
Зачем нужен единый язык?



Модель системы не соответствует представлению бизнеса о ее месте в модели предприятия.

«Не то чтобы совсем не попал,
но только не попал в шарик…»

Итерационное развитие модели



Единый язык позволяет совместить модель системы с представлениями бизнеса о ее месте

Единая модель

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

Модель предметной области **становится** моделью системы

Что мы достигаем?

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

А где здесь ООП?

ООП – парадигма моделирования

Парадигма моделирования определяет

- Элементы языка
- Способ их соединения в сложные конструкции
- Визуальный образ для представления
- Способ отражения модели в код

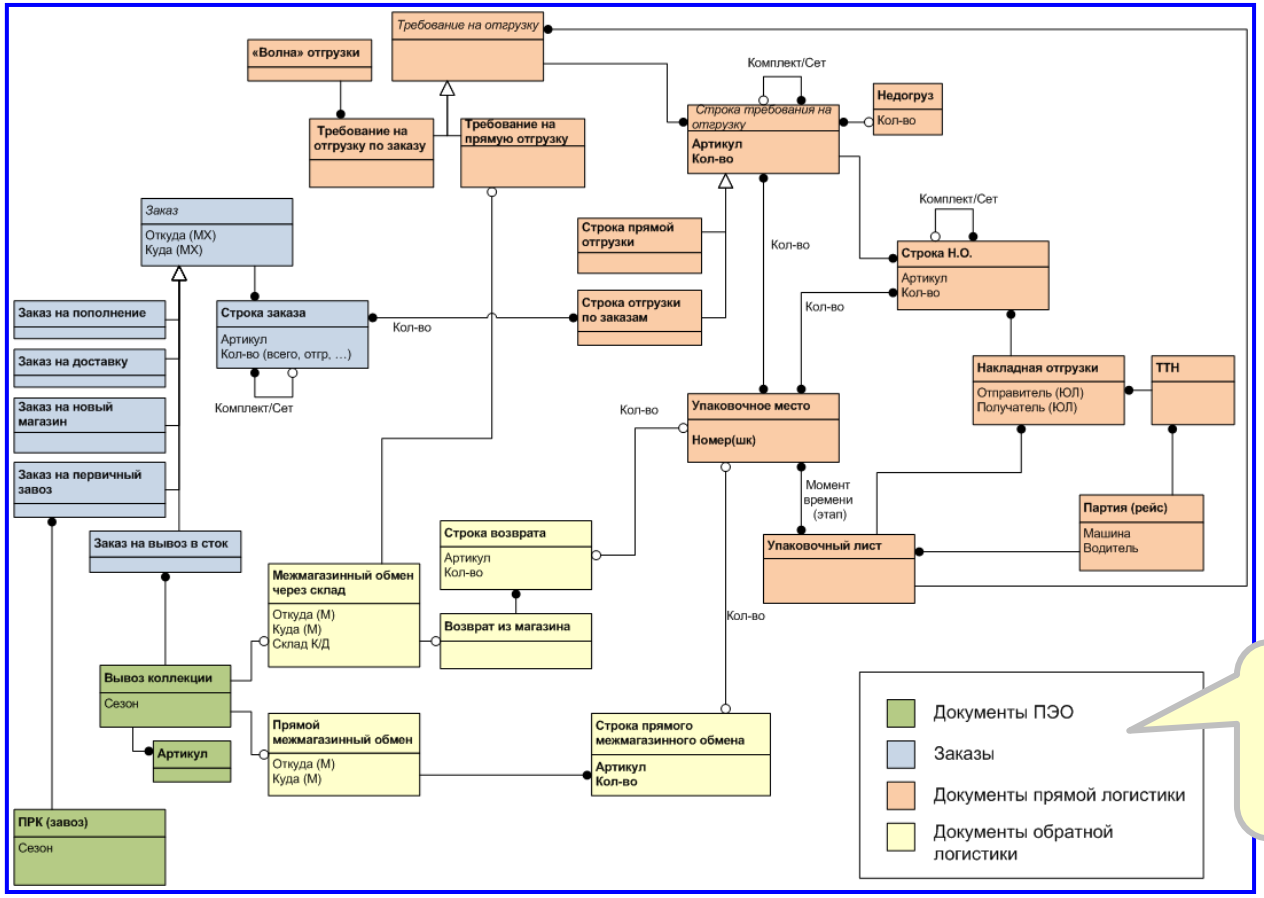
Объекты с атрибутами и методами

Диаграмма классов и другие UML-диаграммы

Типы, соответствующие бизнес-объектам

Модель должен понимать заказчик

Классы соответствуют бизнес-объектам – заказчик видит знакомые названия



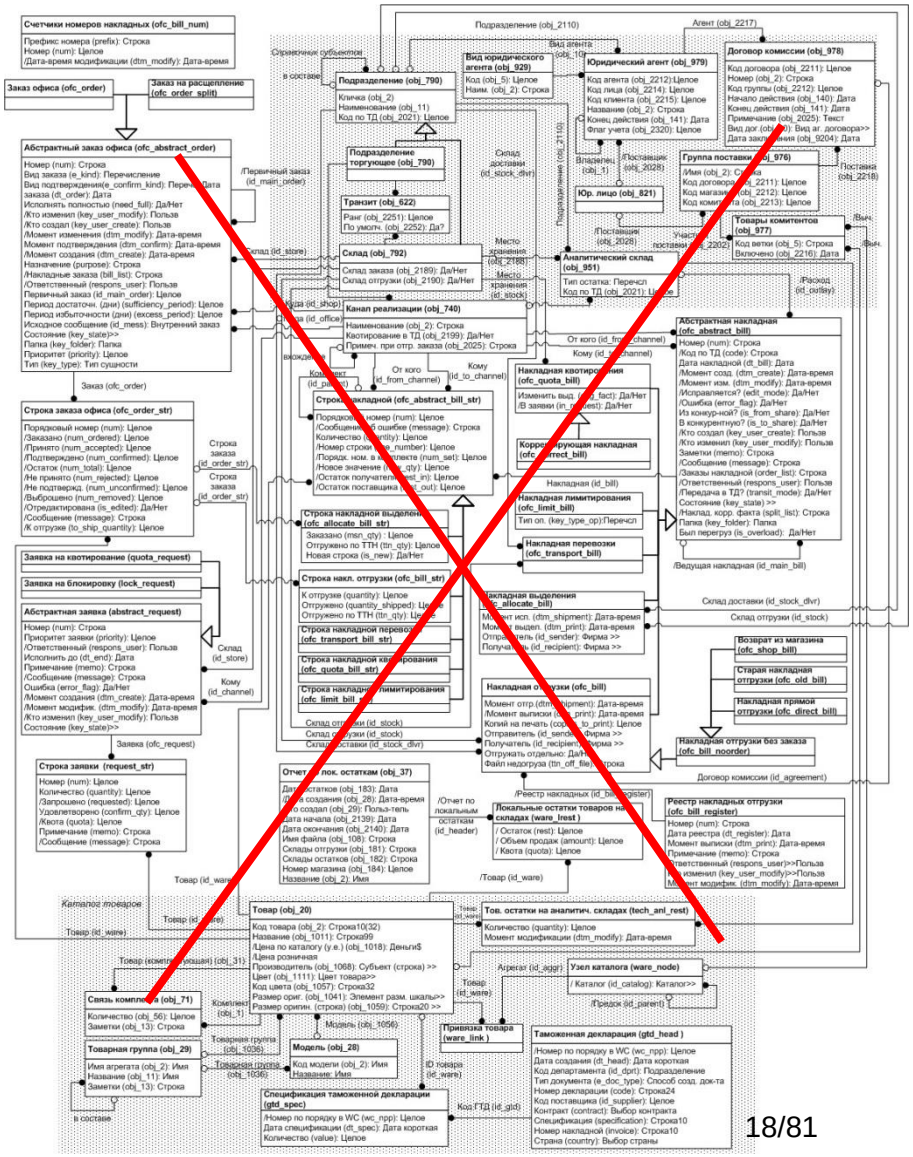
Используем цветовые выделения

Не нужно усложнять диаграмму

☹ Не нужно

- Стараться изобразить все классы на одной диаграмме
- Отображать вспомогательные атрибуты
- Использовать технические коды
- Использовать сложную нотацию

☺ Диаграммы должны понимать заказчики, а не только разработчики



Пример:
Виза на документы

Задача

Задача касается конкретного документа

В процессе обработки документа (сделки, договора, контракта) обеспечить проверку и одобрение определенными сотрудниками или отделами

Выбор проектного решения

В чем проблема?

- Шаблоны обладают разной гибкостью и сложностью
- Для выбора нужно понимать требуемый баланс между гибкостью и сложностью решения

Традиционный подход

- На этапе сбора требований аналитики формулируют задачу для конкретного документа
- Исходя из этого в системе проектируется решение

☹ Выбранное решение отражает текущую ситуацию

Решение надо принимать с учетом потенциального изменения бизнес-процессов

Примеры неудачных решений

- Проверка сделки казначейством и отделом рисков – не получается выполнять параллельно
- Юристы отозвали одобрение кредита, а служба безопасности на него опиралась – связи между визами не контролируются системой
- Настройку виз для одобрения договоров с недвижимостью передали в IT из-за сложности

Решение в рамках DDD

- Представляем шаблоны, описанные применительно к конкретному документу, показываем разницу
- Бизнес-специалисты оценивают потенциальную потребность в реализации бизнес-процессов

☺ Решение принимается с учетом перспективы

В чем Единый язык?

- Для решения модель надо описать понятно бизнесу
  - Можно описать обобщенные решения для документов, «состояния» и «визы», и на них ссылаться
  - Можно описывать каждый случай отдельно
- Термины должны быть понятны Заказчику:

  Например, «визированием» могут считать одобрение документа, требующее только просмотра, а если требуется дополнительная работа, то это называется «обработкой» или «проверкой»
- Общий шаблон надо «перевести» на язык проекта

Результат

- ☺ Мы используем известные шаблоны решений
- ☺ Заказчик оценивает вариант решения не только с точки зрения текущих потребностей, но и из предположений о развитии бизнес-процессов
- ☺ Проектируя изменения бизнес-процессов, заказчик представляет потенциал гибкости системы и принимает решения с учетом этого

Пример:
Обобщенный документооборот

Обобщенный документооборот

- Документ обрабатывается в несколько этапов.
- На каждом этапе определенные сотрудники могут совершать определенные действия.
- Для передачи на следующий этап должны выполняться определенные условия.

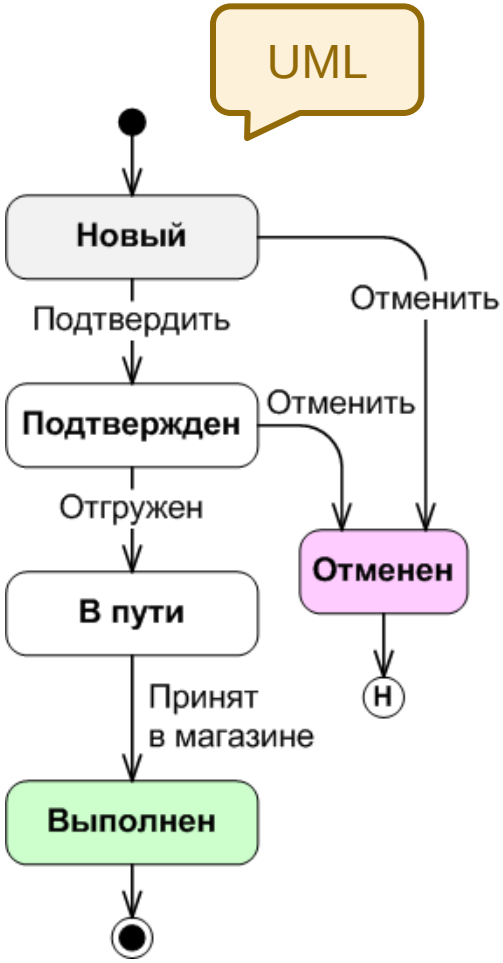
Модель для документооборота

- Документу приписываем состояние.
- Состояние определяет этап документооборота:
  - какие действия можно совершать над документом;
  - кто отвечает за обработку документа;
  - кто имеет права на совершение тех или иных действий.
- Возможные изменения состояний документа образуют граф переходов.

Вместо множества атрибутов используем один, определяющий фазу жизненного цикла

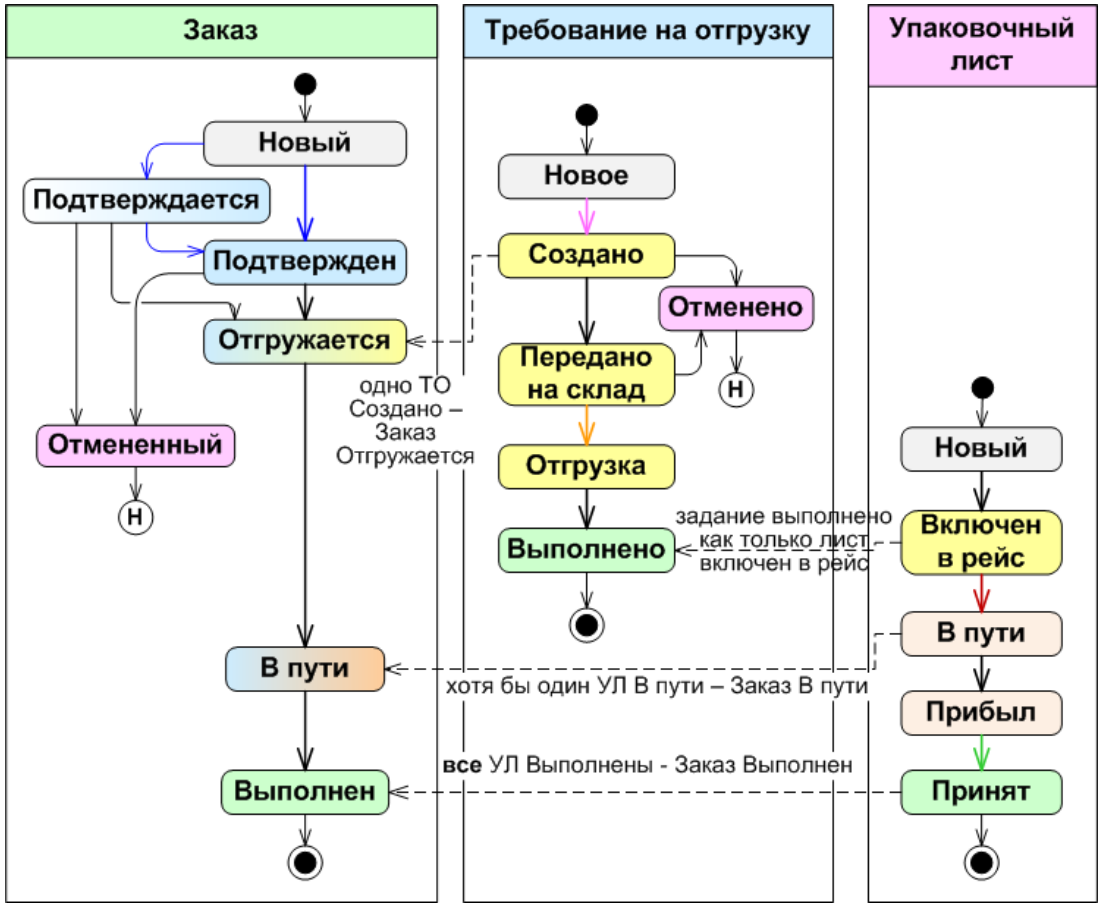
Язык модели

- Документ – объект предметной области.
- Действие над ним – вызов его метода.
- Среди всех методов выделяем переходы и связываем их с состояниями.
- Граф состояний – State machine diagram.



Язык ООП «с расширениями».
Названия состояний и переходов – на языке бизнеса.

Наглядно представляет сложный документооборот



Практики проектирования – на этап бизнес-анализа

Частные практики

- Стратегии
- Политики
- Выделение общих объектов
- Абстракция через интерфейсы

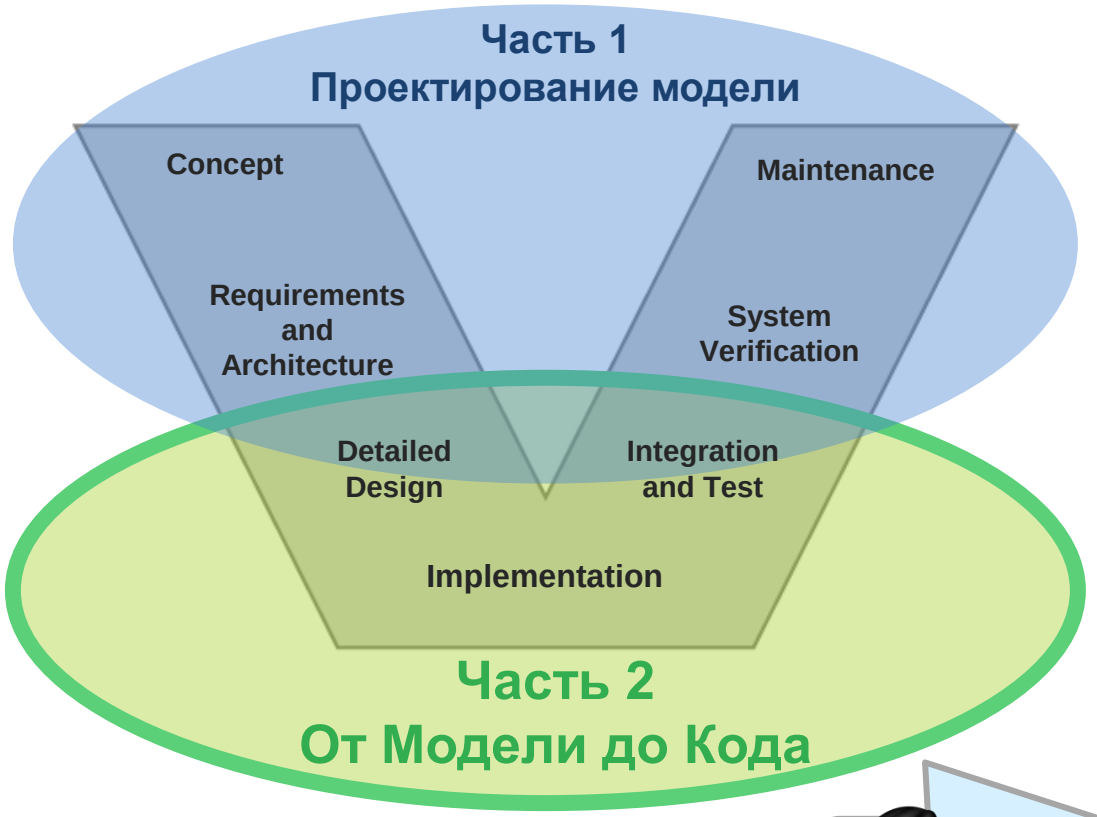
Технические практики наполняются бизнес-смыслом и служат для общения с Заказчиком

Контексты

Перенос практик декомпозиции на бизнес-анализ

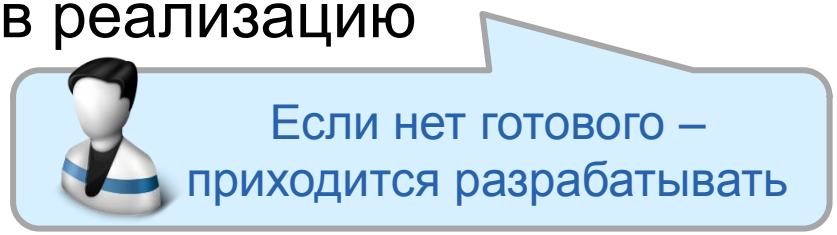
- Понятие ограниченных контекстов
- Различные варианты выделение общего
  - Общее ядро
  - Абстрактное ядро
  - Подключаемые компоненты
  - Крупноблочная структура
  - Уровни разделения обязанностей
- Изоляция и карта контекстов

Часть 2. От модели до Кода



Как отражать модель в реализацию

- Язык, реализующий парадигму
- Framework, часто с диаграммами, например, MS Workflow Foundation для реализации документооборота
- DSL, лучше графический, с компилятором или интерпретатором
- Диаграммы и понятия единого языка и шаблоны их отражения в реализацию



Простое применение DDD

- На основе требований создаем объектную модель
  - структура классов
  - поведение объектов
  - интерфейсы
- Согласуем ее с заказчиком
- Реализуем в коде на основе шаблонов Domain model и Rich objects

Что получается?

Достоинства объектной модели

- ☺ Соответствует парадигме современных языков
- ☺ Понятна разработчикам и аналитикам
- ☺ Имеет эффективные визуальное представление
- ☺ Соответствует реальному миру и понимается заказчиком – если проектировать бизнес-объекты

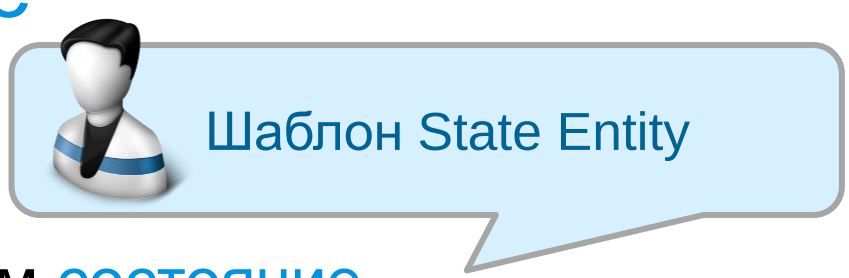
Достоинства DDD с ООП

- ☺ ООП – общепринятый и (на простом уровне) интуитивно понятный метод
- ☺ Модель, сделанная в объектной парадигме, может быть представлена заказчику и согласована с ним
- ☺ Современные языки поддерживают ООП, модель хорошо реализуется с использованием шаблонов Domain model и Rich objects

Этот способ **работает,** но **ограниченно.** Простого ООП не хватает, а изучать сложный заказчики не считают правильным

Документооборот и State Entity

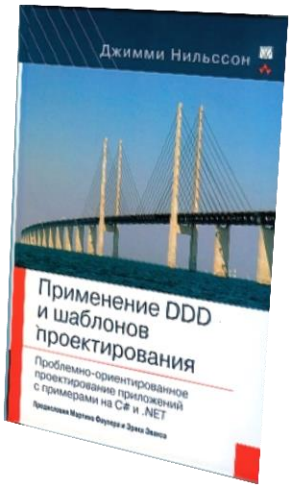
Напомним решение



- Документу приписываем состояние
- Состояние определяет этап документооборота:
  - какие действия можно совершать над документом
  - кто отвечает за обработку документ
  - кто имеет права на совершение тех или иных действий
- Возможные изменения состояний документа образуют граф переходов

Варианты шаблона реализации

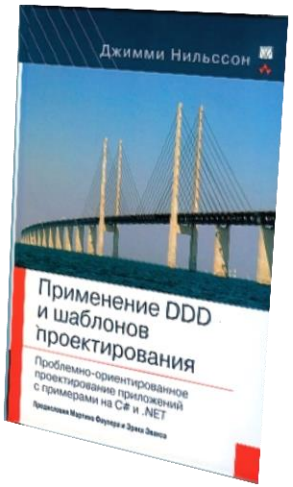
Из книги Нильссона



Варианты шаблона реализации

Из книги Нильссона

1. Императивно в коде конкретных методов



В методе `AddOrderLine()` это можно реализовать следующим образом:

```
private void AddOrderLine(OrderLine orderLine)
{
  if (_currentState == OrderState.Registered || _currentState ==
OrderState.Granted)
    _currentState == OrderState.NewOrder;
  else if (_currentState == OrderState.NewOrder)
    // Не выполнять переход.
  else
    throw new ApplicationException(...

  // Выполнить полезные действия...
}
```

Варианты шаблона реализации

Из книги Нильссона

1. Императивно в коде конкретных методов
2. Императивно в едином методе смены состояния

```
private void _ChangeState(OrderState newState)
{
  if (newState == _currentState)
    retun; // Не считать ошибкой переход состояния

  switch (_currentState)
  {
    case OrderState.NewOrder:
      swith (newState)
      {
        case OrderState.Registered:
        case OrderState.Cancelled:
          _currentState = newState;
          break;

        default:
          throw new ApplicationException(...
          break;
      }
    case OrderState.Registered:
      switch (newState)
      {
        case OrderState.NewOrder:
        case OrderState.Granted:
        case OrderState.Cancelled:
          _currentState = newState;
          break;

        default:
          throw new ApplicationException(...
          break;
      }
...
// И так далее...
  }
}
```

Теперь метод AddOrderLine() выглядит следующим об

```
public void AddOrderLine(OrderLine orderLine)
{
    _ChangeState(OrderState.NewOrder);

    // Выполнить полезные действия...
}
```

Варианты шаблона реализации

Из книги Нильссона

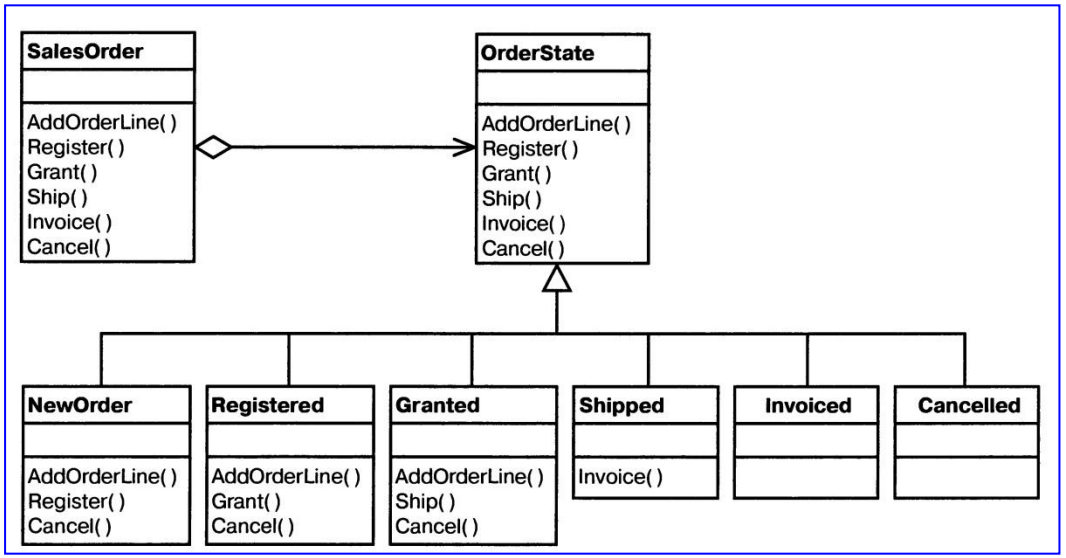
1. Императивно в коде конкретных методов
2. Императивно в едином методе смены состояния
3. Декларативно – через таблицу переходов и состояний

Текущее состояние	Метод	Новое состояние
NewOrder	Register()	Registered
NewOrder	Cancel()	Cancelled
Registered	AddOrderLine()	Cancelled
Registered	Grant()	Granted
Registered	Cancel()	Cancelled
...		...

Варианты шаблона реализации

Из книги Нильссона

1. Императивно в коде конкретных методов
2. Императивно в едином методе смены состояния
3. Декларативно – через таблицу переходов и состояний
4. Через иерархию классов-состояний



Что выбрать?

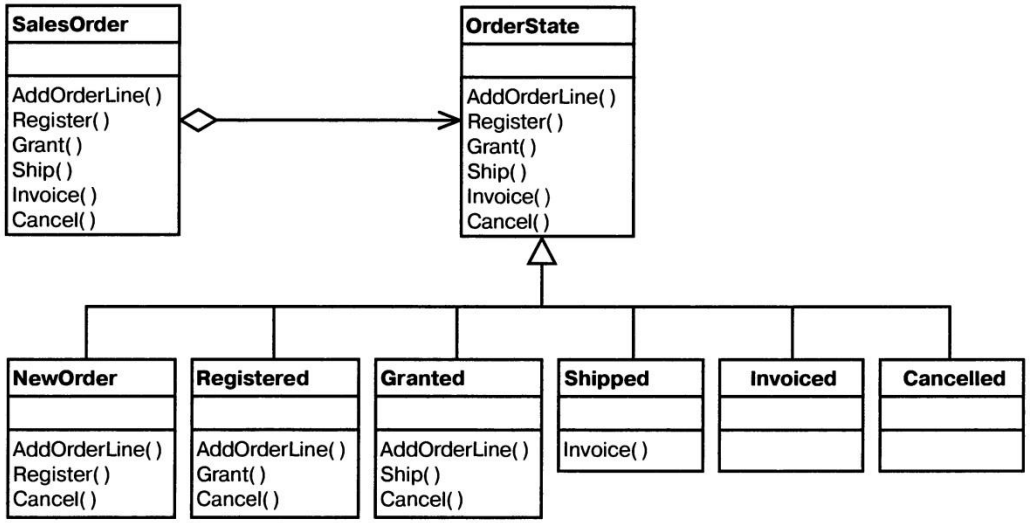
Модель должна прозрачно отражаться в реализацию

- Используем декларативное описание (3)
- Или комбинацию (1) и (3):
  - императивно изменяем состояние в методе перехода
  - контролируем, что изменение соответствует декларативной разметке

- Таблица метаданных – декларативное описание – однозначно соответствует диаграмме состояний

Почему не иерархия состояний?

- Иерархия классов-состояний – в объектной модели
  - Полнее выразить реализацию в диаграмме классов
  - Но поведение документа – за рамками, оно только в графе переходов
  - Поэтому для единого языка, понимаемого заказчиком – не подходит



Почему не иерархия состояний?

- Иерархия классов-состояний – в объектной модели
  - Полнее выразить реализацию в диаграмме классов
  - Но поведение документа – за рамками, оно только в графе переходов
  - Поэтому для единого языка, понимаемого заказчиком – не подходит
- Реализация через метаданные
  - Таблица переходов отражает поведение документа в коде
  - И однозначно соответствует диаграмме переходов
  - Диаграмма переходов понимается заказчиком – единый язык
  - При этом иерархия состояний становится излишней
  - Однако, реализация требует выхода из объектной модели

Сравним модели…

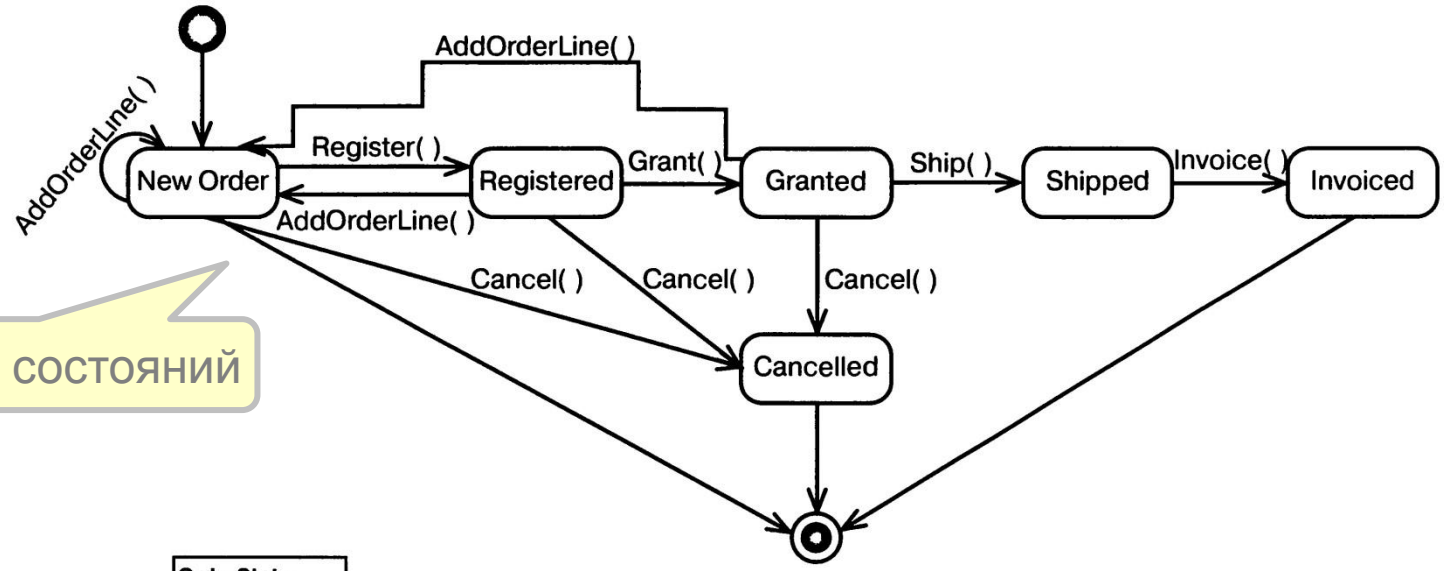


Диаграмма состояний

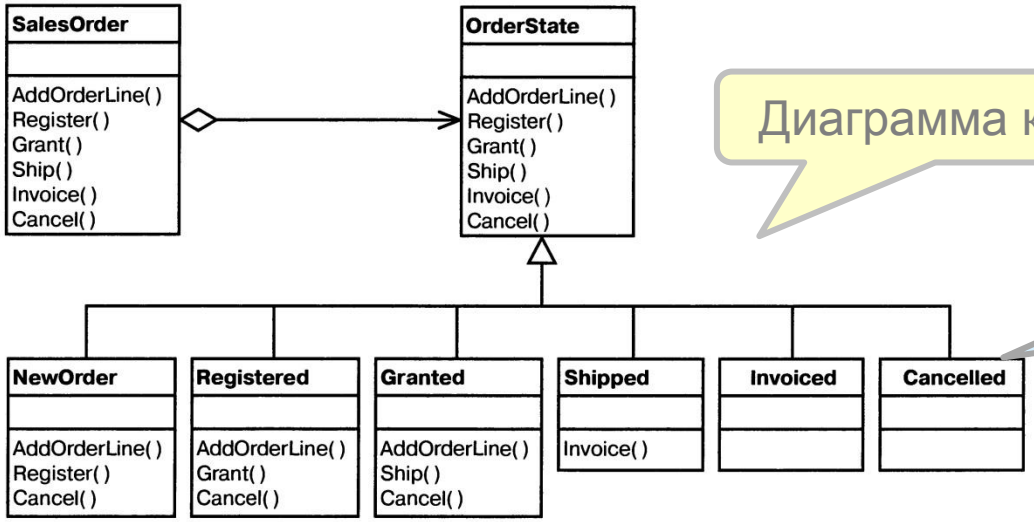


Диаграмма классов

Это – лишнее

Реализация на разных платформах

- Отдельный проект
  - Инициализация таблицы переходов для каждого документа
  - Общая функция проверки перехода для всех методов (вместе с логом)
  - Таблица допустимых действий для каждого состояния (тоже с логом)

- Собственный объектный framework в Oracle
  - Таблица переходов и прав в метаданных
  - Вызов процедур-методов динамически с проверками

Реализация на разных платформах

Собственный фреймворк на C#

- Разметка методов метаданными – состояния, права
- Описание в метаданных графа состояний и условий
- Обработка на посткомпиляции при создании реализации

```
[Method(AutoSave = true)]
[StateRestriction(RequestForShipmentState.New)]
[StateTransition(RequestForShipmentState.New, RequestForShipmentState.Created)]
[GrantInvocation(RmsRole.Manager)]
public virtual void PrepareForShipment()
{
    State = RequestForShipmentState.Created;
    ...
```

Проблемы при стандартном отражении модели в код

Проблемы при использовании 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 – как 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 IT-объекты получается очень похоже на антипаттерн Big Object

☹ 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 не понимается заказчиком, простая – не отражает реализацию

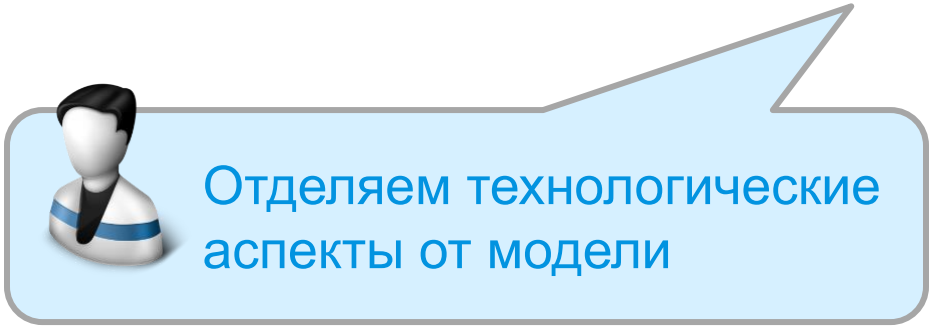
Ограниченная область DDD-модели

- Подход разрабатывался для слоя бизнес-логики
- Использование объектных языков побуждает ограничиться объектными моделями

Хочется

- Использовать модель при проектировании интерфейсов
- Расширять способы моделирования

Решение – технологические рельсы



Что такое технологические рельсы

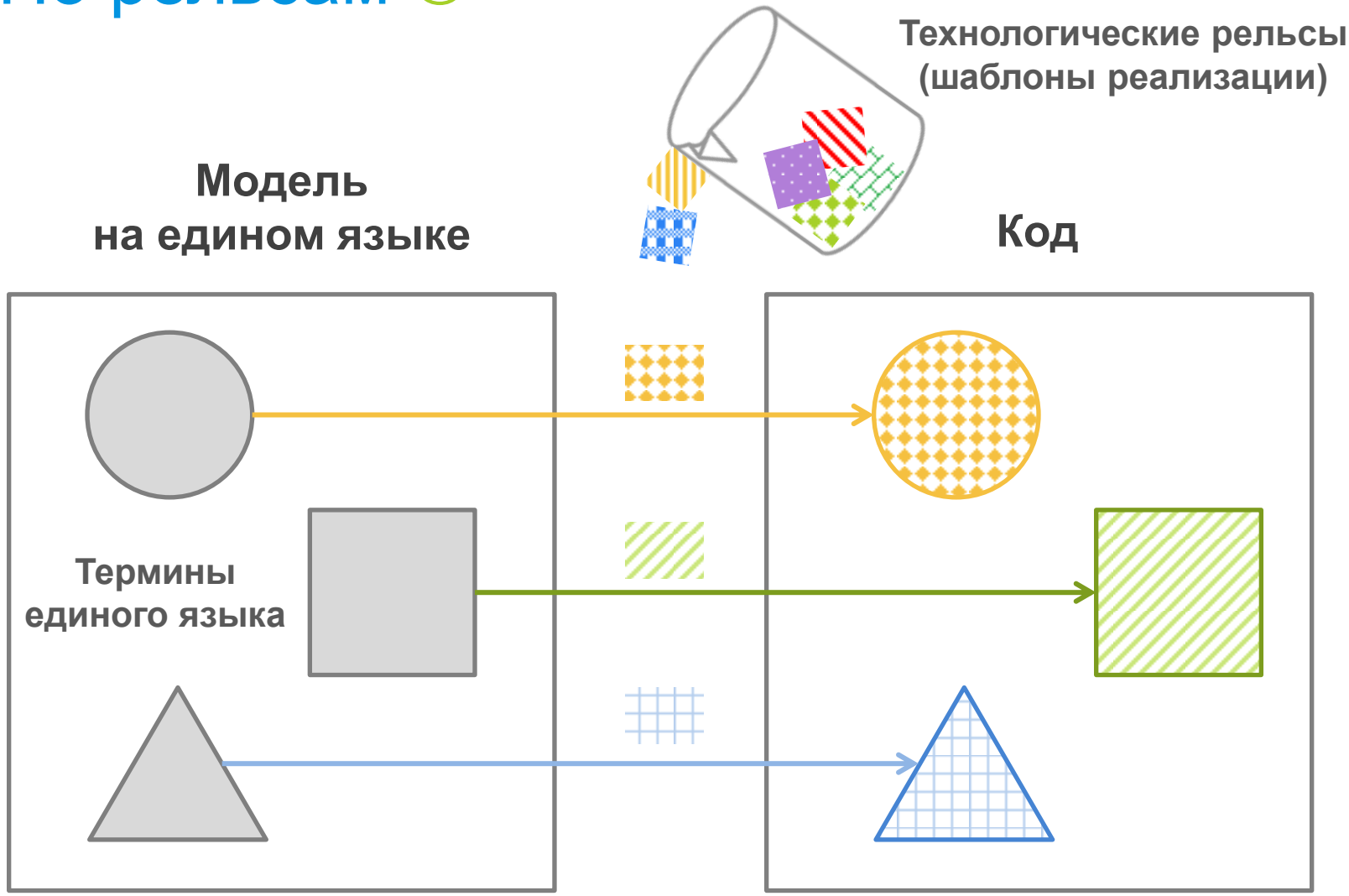
Технологические рельсы – это способ перевода модели на едином языке в код

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

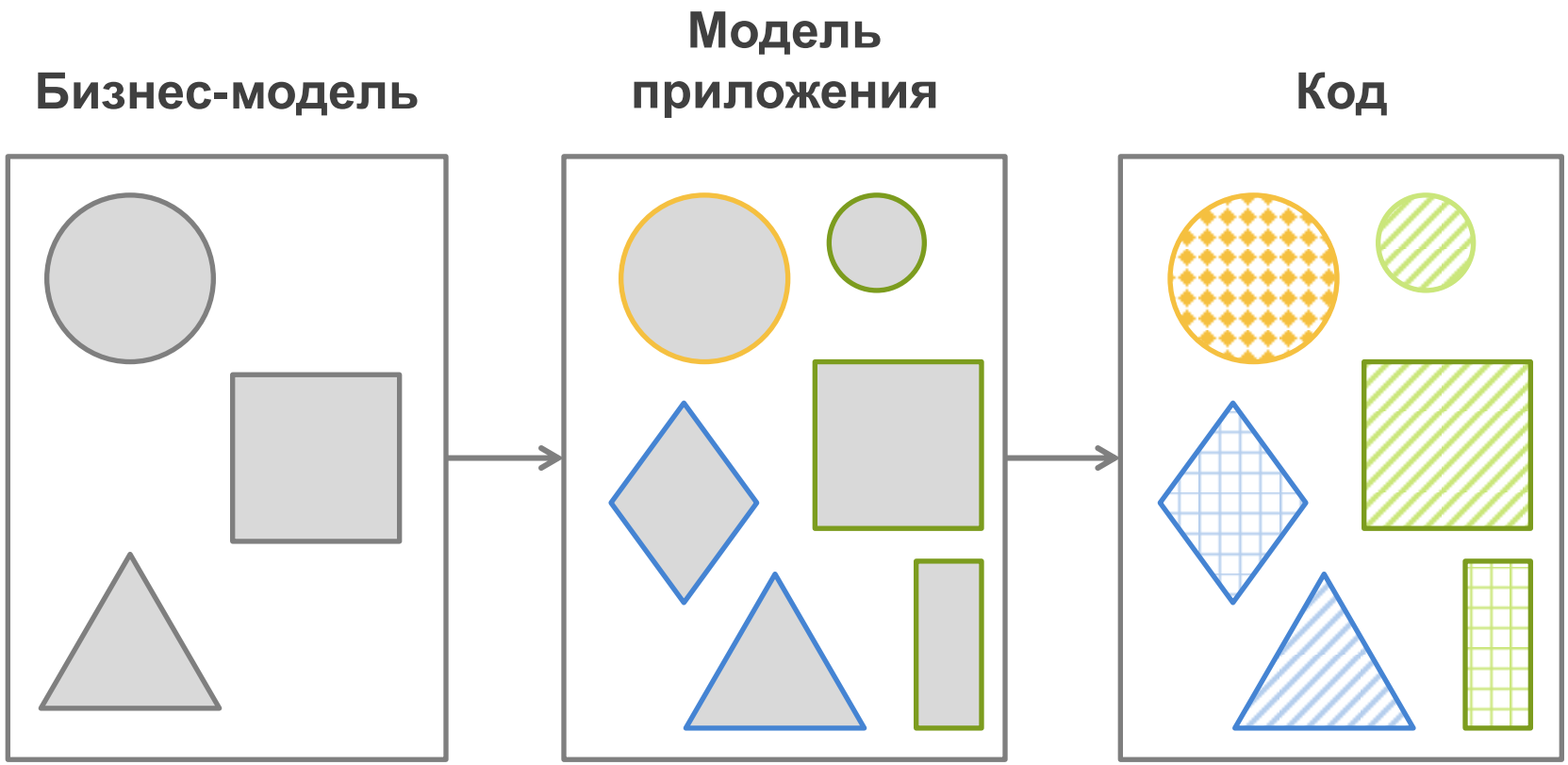
Формулируются на всех уровнях – от архитектуры до частных задач посылки сообщения

«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



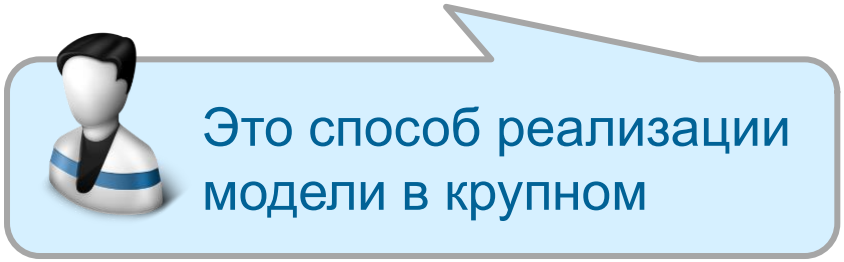
Виды технологических рельсов

- Архитектурные
- Инфраструктурные
- Рельсы предметной области

Архитектурные рельсы

Структура приложения в крупном, например:

- Java с Hibernate и Spring
- Создаем anemic-объекты с разметкой Hibernate для каждого бизнес-класса, используем их так же, как DTO для интерфейса
- Бизнес-логику каждого класса реализуем в отдельном статическом классе
- Документооборот описываем через состояния объекта, для реализации используем StateEntity с декларативной таблицей состояний…
- …



Инфраструктурные рельсы

Способы реализации стандартных задач: печатных форм, отправки сообщений, интеграции, например:

- Все печатные формы реализуются с помощью библиотеки X
- Доступ к библиотеке – через сервис-локатор
- Для бизнес-объекта с печатной формой создаем класс PrintForm<TClass>, в котором каждой форме соответствует отдельный метод…
- На интерфейсе команды печати появляются автоматически, исходя из созданных классов…

Рельсы предметной области

Способ реализации конструкций, характерных для предметной области проекта, например документа с товарными строками:

- Строки документов хранятся в отдельных таблицах, однако в объектной модели доступ к ним – исключительно через шапки документов
- Для реализации типовой работы класс документа наследуем от WareDoc<TDocHeader, TDocItem>, что обеспечивает стандартный функционал…
- Если документ не хранит цены и стоимость, а берет их из справочников, реализуем стратегию PriceCalc
- …

Пример.
Печать накладной

Рельсы для печатной формы

Задача: Для накладной надо печатать форму ТОРГ-12

Задача – типовая, решение – единообразное

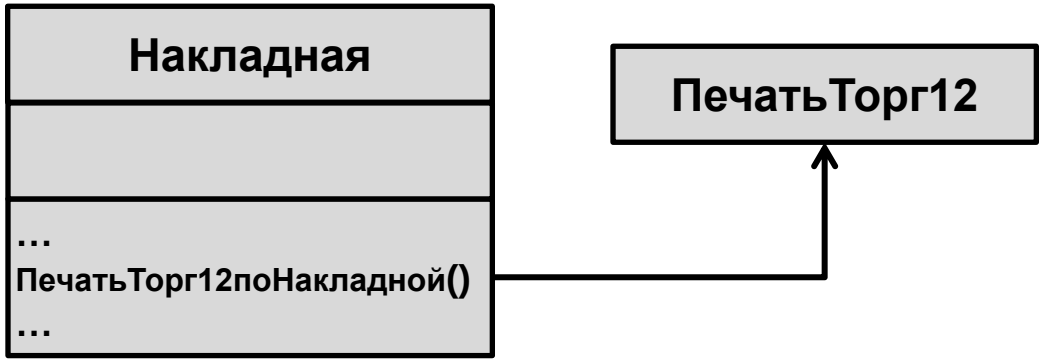
Требования к решению:

- Отделить печать от бизнес-логики
- Обеспечить независимое тестирование
- Желательно обобщенное решение

Декомпозиция кода и независимое тестирование – признаки хорошего стиля

Печать форм: решение 1

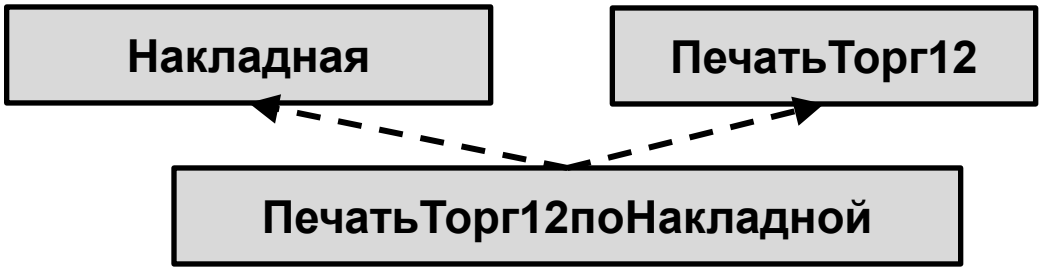
Обобщенный сервис ПечатьТорг12
и метод НапечататьТорг12 у класса Накладная



- ☺ Простое и очевидное решение
- ☹ Увеличивается класс Накладная, в нем смешивается разнородная бизнес-логика
- ☹ Класс Накладная зависит от сервиса печати, это сильно затрудняет тестирование

Печать форм: решение 2

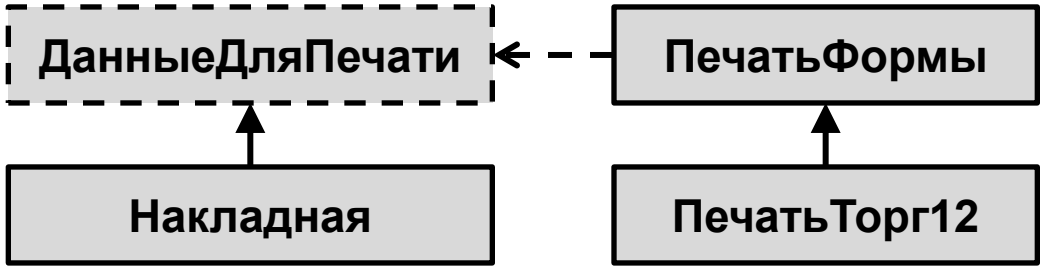
Обобщенный сервис ПечатьТорг12 и сервис ПечатьТорг12поНакладной



- ☺ Логика печати вынесена из бизнес-объектов
- ☹ Для тестирования ПечатьТорг12поНакладной необходима Накладная со всей инфраструктурой
- ☹ Таких классов печати становится довольно много

Печать форм: решение 3

Обобщенный сервис ПечатьФормы и сервис ПечатьТорг12поНакладной



☺ Логика печати вынесена из бизнес-объектов

☺ Печать и бизнес-логика тестируются независимо

Решение применяем ко всем задачам печати, тогда в модели достаточно перечислить формы

Расширение модельного ряда

Недостатки объектной модели

По опыту разработки корпоративных приложений

☹ Плохо представляет цикл жизни объекта

☹ Плохо подходит для отражения потоков ресурсов

☹ Плохо подходит для систем связанных показателей

Если для области автоматизации эти аспекты важны, можно применять другие парадигмы, дополняя объектную или обходясь без нее

Диаграммы состояний

- У класса-документа есть атрибут **состояние**, описывающий текущее место в документообороте
- В модели для документов создаем диаграмму состояний, на ней же показываем роли
- В классе для каждого перехода реализуем метод, помечая его метаданными
- В начале и в конце метода перехода вызываем PreTransit и PostTransit, которые обеспечивают логирование и контролируют состояние документа

Модель для интерфейсов

- Реализуем конструктор обобщенных интерфейсов на основе метаданных бизнес-объектов – таблиц с фильтрами для просмотра, диалогов для изменения
- Обеспечиваем точки расширения для реализации дополнительной логики

☺ В постановках на 90% интерфейсов – списки полей

☺ Реализация требуется только для особых случаев

Как сделать необъектную модель?

- Выбрать или придумать парадигму моделирования
  - Объекты обмениваются сообщениями
  - Ресурс выделяется действующим лицам
- Определить визуальный образ единого языка
  - Диаграммы взаимодействия и синхронизации
  - Образ разрезания пиццы
- Разработать правила отражения модели в код – иначе элементы модели не найти в реализации
- Лучше, если отражение будет по шаблонам

Задачи ведения учета

- Присутствуют в большинстве enterprise-приложений
- Плохо описываются в терминах объектов

Решение

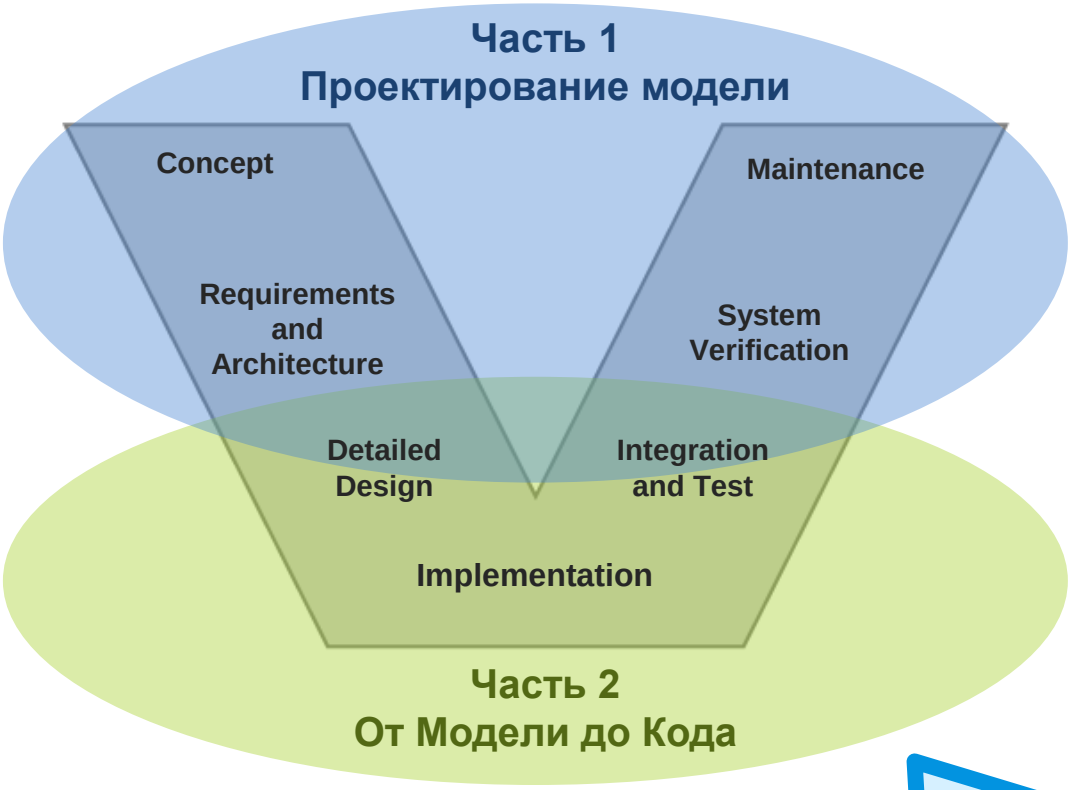
- Специальные диаграммы учета
- Базовое учетное ядро – обобщенный учет
- Шаблон реализации диаграмм учета на учетном ядре

☺ Учет описывается в адекватных терминах

☺ Диаграммы обеспечивают эффективное понимание

Подробнее о диаграммах состояний и реализации учета –
в докладе на ADD-2011 «Необъектные модели предметной области»

Заключение – профит



Что достигнуто?

Технологические рельсы:

- сочетают описание модели в понятных заказчику терминах со сложными техническими решениями
- помогают соблюдать ограничения, накладываемые фреймворками, платформами и библиотеками на организацию программного кода, поддерживают их совместную работу и однородность решений в рамках проекта

Все это обеспечивает эффективную разработку и является залогом долгого и успешного развития ИТ-системы, уже находящейся в эксплуатации

Аналитик и архитектор

- В небольших проектах аналитик и архитектор – один человек
- В крупных проектах аналитик создает модель, а архитектор проектирует техническую реализацию
- В классическом процессе каждый из них работает со своей моделью
- В DDD – совместная работа над одной моделью

☹ Конфликты – ответственность плохо разграничена

Разработка «с рельсами»

- Аналитик – создает модель и согласует ее
- Архитектор – отвечает за технологические рельсы, которые обеспечивают эффективную реализацию
- Конструкции единого языка – способ синхронизации модели с технологическими рельсами

☺ Разграничение ответственности

☺ Эффективность проектных решений обеспечивается технологическими рельсами

☺ Параллельная работа над проектом

Что же обеспечивает DDD?

Единый язык + Единая модель:

- обеспечивают надежную основу для общения всех участников проекта при принятии решений;
- успешно заменяют мелкую россыпь требований;
- позволяют эффективно развивать сложную систему.

Это требует дополнительных усилий:

- формирование единого языка;
- понимание разработчиками предметной области.

По опыту, результат окупает усилия.

Спасибо! Вопросы?

Максим Цепков http://www.facebook.com/mtsepkov