Проектирование конфигурационного управления

Лекция 6 (22)

Управление конфигурацией

Овчинников П.Е.

МГТУ «СТАНКИН»,

ст.преподаватель кафедры ИС

01.03.2021

Конфигурация = документированность

Модель процессов

Модель архитектуры ИС

Модель данных ИС

Упрощенная модель отношения

Детализированная модель отношения

Терминология: инженерия

ГОСТ Р 57193-2016 Системная и программная инженерия. Процессы жизненного цикла систем

системная инженерия (systems engineering):

междисциплинарный подход, управляющий **полным техническим и организаторским усилием**, требуемым для преобразования ряда потребностей заинтересованных сторон, ожиданий и ограничений в решение и для поддержки этого решения в течение его жизни

ISO/IEC/IEEE 24765:2017 Systems and software engineering -- Vocabulary

программная инженерия (англ. *software engineering*): приложение

- **систематического**
- **дисциплинированного**
- **измеримого**

подхода к

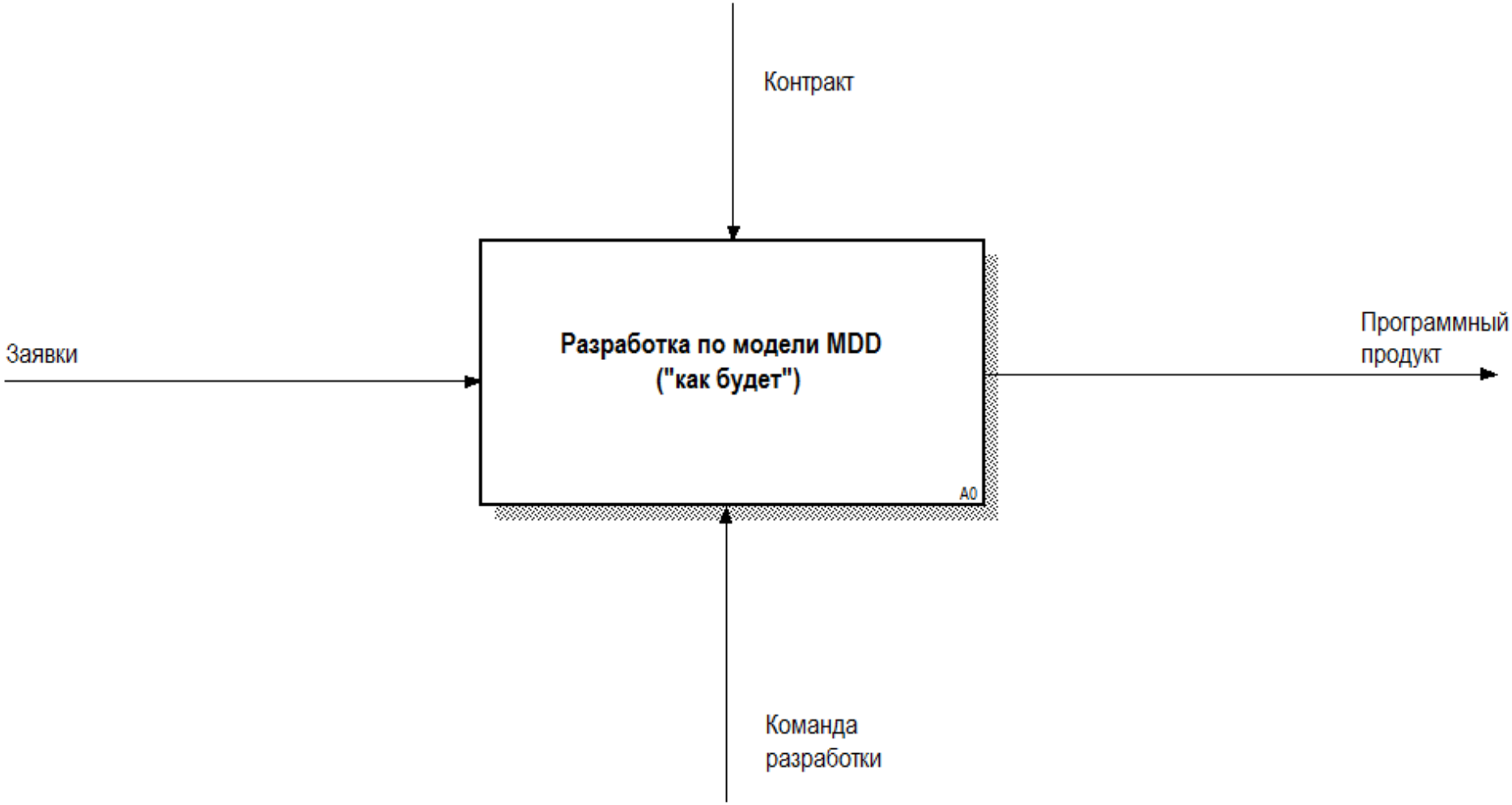
- **разработке**
- **функционированию и**
- **сопровождению**

программного обеспечения,

а также к исследованию этих подходов

то есть, приложение дисциплины инженерии к программному обеспечению

Инженерия в модели MDD



Гибкая разработка: модель MDD

Разработка, управляемая моделями (*model-driven development,* **MDD**, *Model-driven engineering*, **MDE**) —

> **стиль разработки** программного обеспечения, когда **модели** становятся **основными артефактами** разработки, из которых **генерируется код** и **другие артефакты**

Модель —

> **абстрактное описание** программного обеспечения, которое скрывает информацию о некоторых аспектах с целью представления упрощенного описания остальных

Модель может быть **исходным артефактом** в разработке, если она фиксирует информацию в **форме**, пригодной для **интерпретаций людьми** и обработки **инструментальными средствами**

В разработке, управляемой моделями, конфигурация программного и информационного обеспечения всегда точно описана в базах данных, описывающих модели!

MDE

Гибкая разработка: модель MDD

Модель определяет **нотацию** и **метамодель**

Нотация

представляет собой совокупность **графических элементов**, которые применяются в модели и могут быть интерпретированы людьми

Метамодель

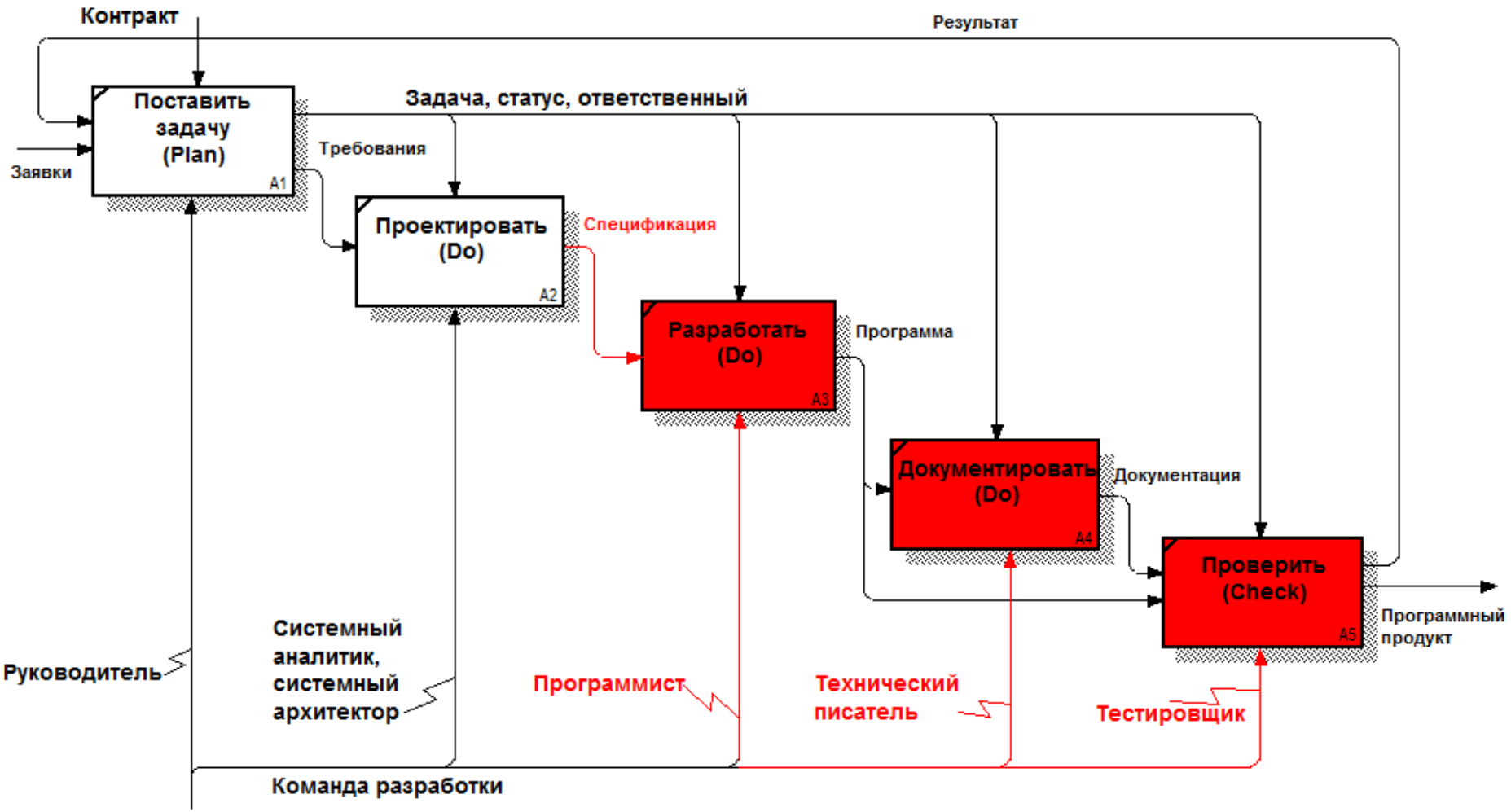
описывает используемые в модели понятия и фиксирует информацию в виде метаданных, которые могут быть обработаны инструментальными средствами

Наиболее известными современными MDE-инициативами являются:
1.разработка Object Management Group (OMG) под названием model-driven architecture (MDA)
2.экосистема Eclipse для инструментов моделирования и программирования (Eclipse Modeling Framework)

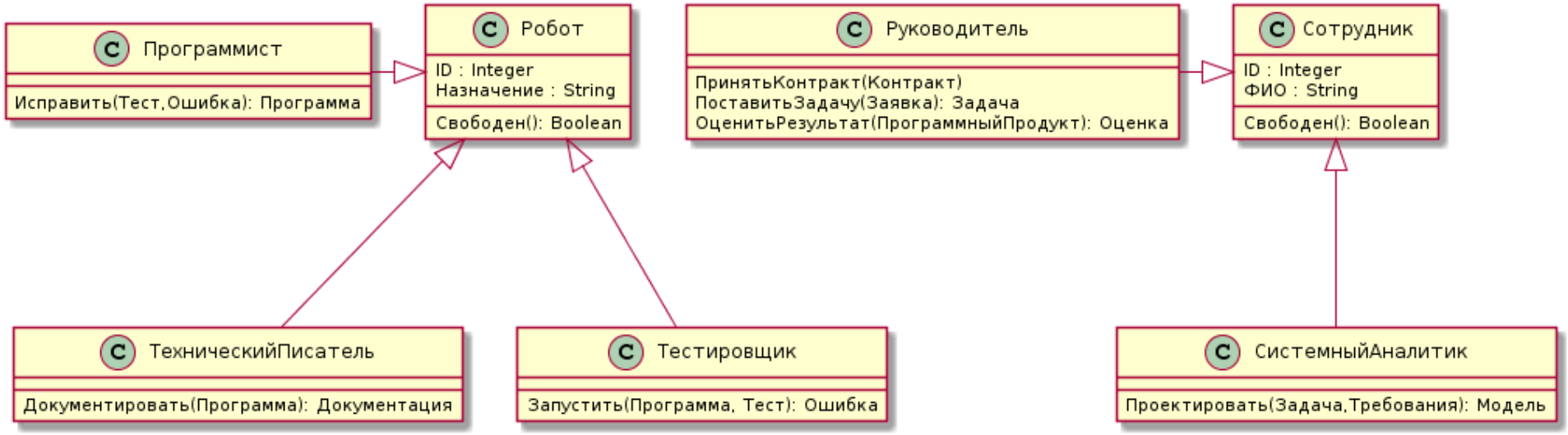
В отечественной ИТ-индустрии наиболее известным продуктом, поддерживающим MDD, является платформа «1С:Предприятие»

Объект проектирования: метамодель

Модель MDD и команда



Модель MDD и команда



http://www.intuit.ru/studies/courses/3467/709/lecture/21825
http://www.intuit.ru/studies/courses/3467/709/lecture/21825?page=2

Модель MDD и команда

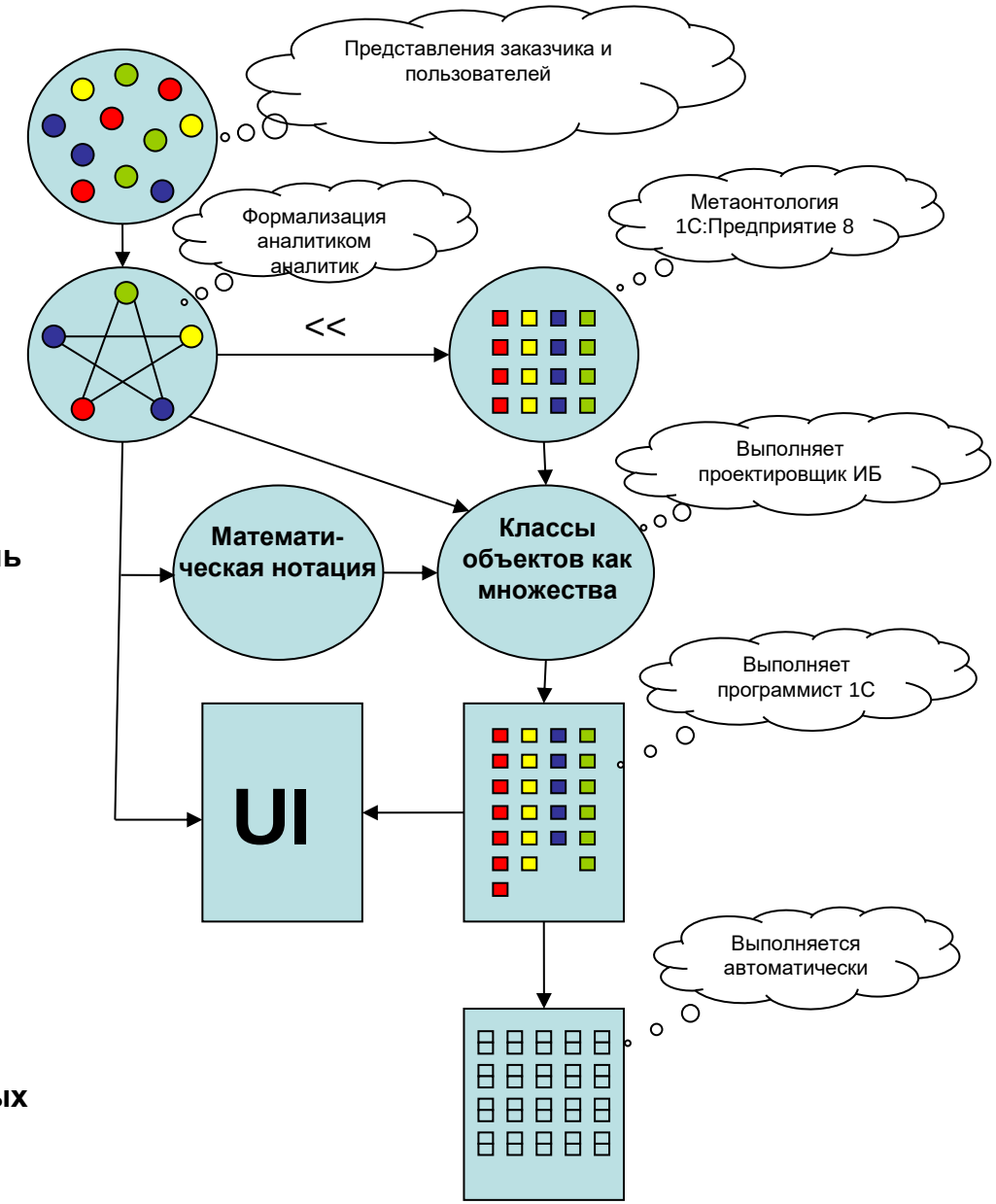
1. **Семантическая сеть**

2. **Онтология**

3. **Концептуальная модель**

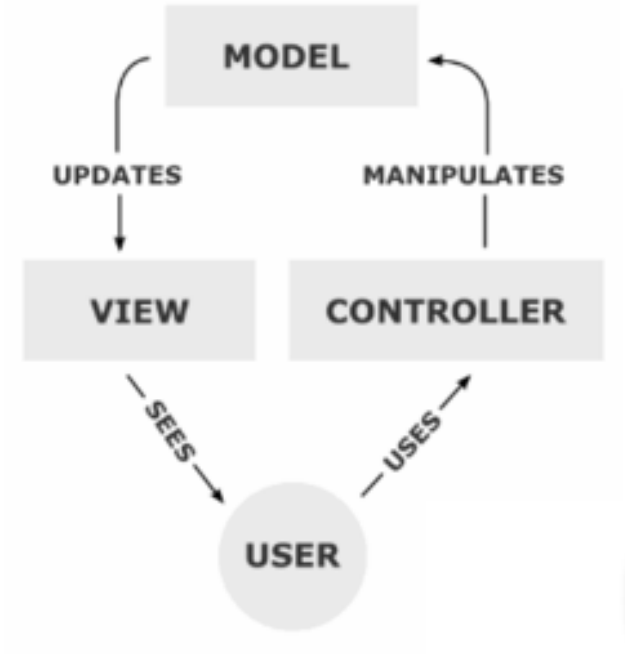
4. **Объектная модель информационной базы 1С:Предприятие**

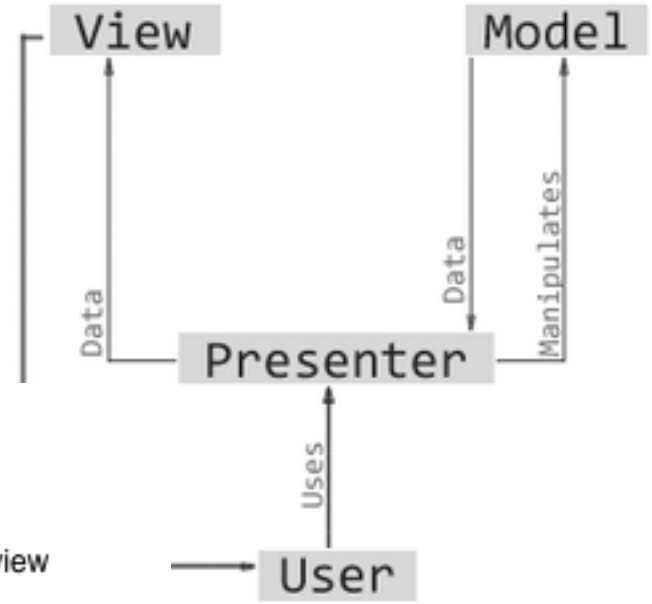
5. **Логическая модель реляционной базы данных**

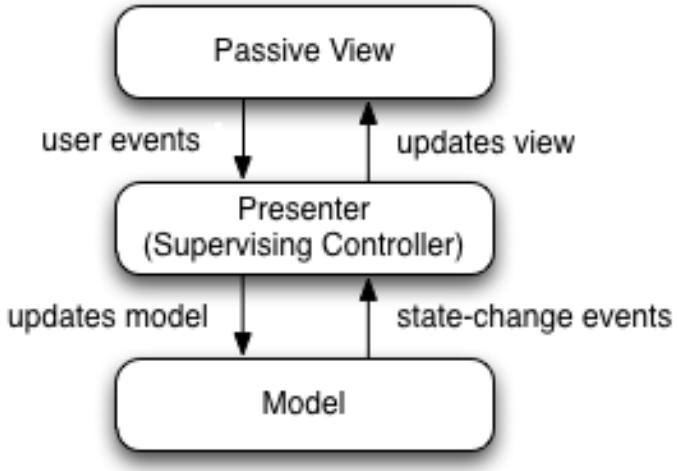


Частичная реализация принципов MDD

Паттерны MVC



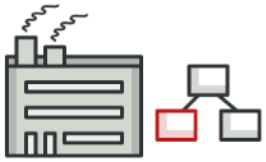




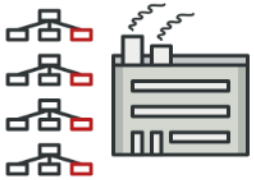
[Model-View-Controller]

Частичная реализация принципов MDD

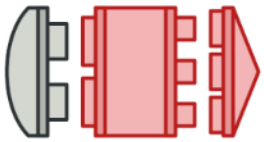
Паттерны ООП



Фабричный метод
Factory Method



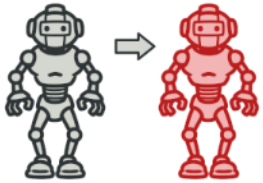
Абстрактная фабрика
Abstract Factory



Адаптер
Adapter

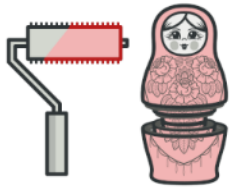
Мост
Bridge

Строитель
Builder



Прототип
Prototype

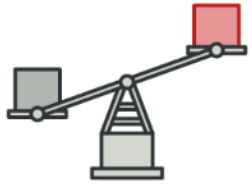
Компоновщик
Composite



Декоратор
Decorator

Одиночка
Singleton

Фасад
Facade



Легковес
Flyweight

[Архитектурные паттерны]

Частичная реализация принципов MDD

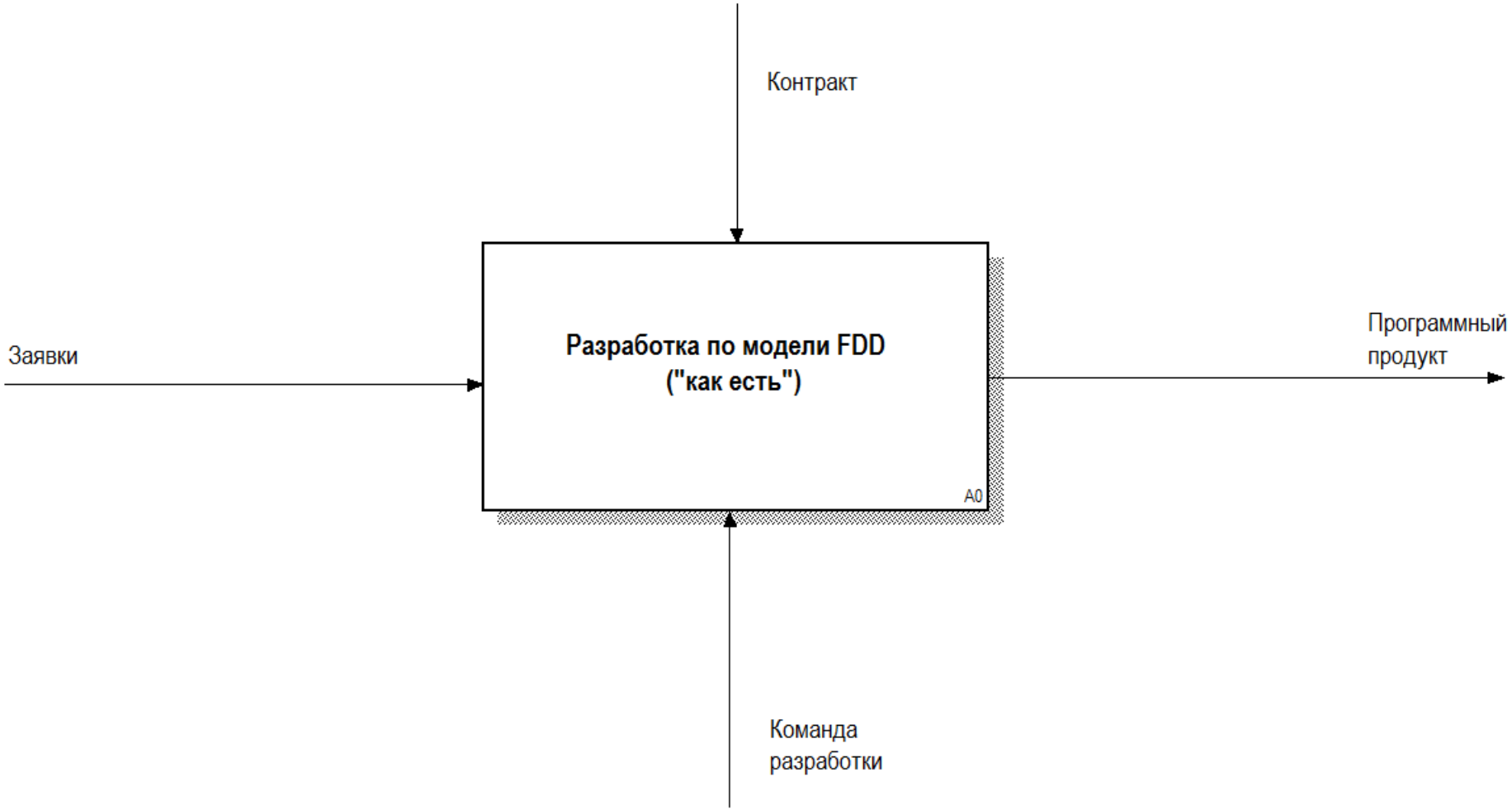
Постпроцессоры

PostCSS — программа, которая автоматизирует рутинные операции с CSS с помощью расширений, написанных на языке JavaScript

Используется при разработке Википедии, Facebook и GitHub

Один из самых часто загружаемых с npm инструментов для работы с CSS

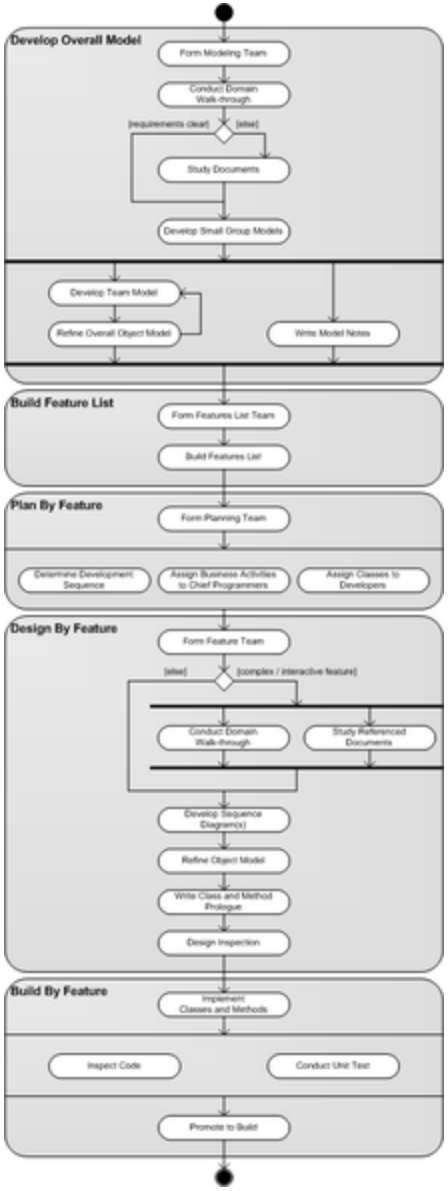
Инжиниринг в модели разработки FDD



Гибкая разработка: модель FDD

Feature driven development (**FDD**, *разработка, управляемая функциональностью*) — итеративная методология разработки программного обеспечения, одна из *гибких методологий разработки (agile)*

FDD представляет собой попытку объединить наиболее признанные в индустрии разработки программного обеспечения методики, принимающие за основу важную для заказчика функциональность (свойства) разрабатываемого программного обеспечения. Основной целью данной методологии является разработка реального, работающего программного обеспечения систематически, в поставленные сроки.

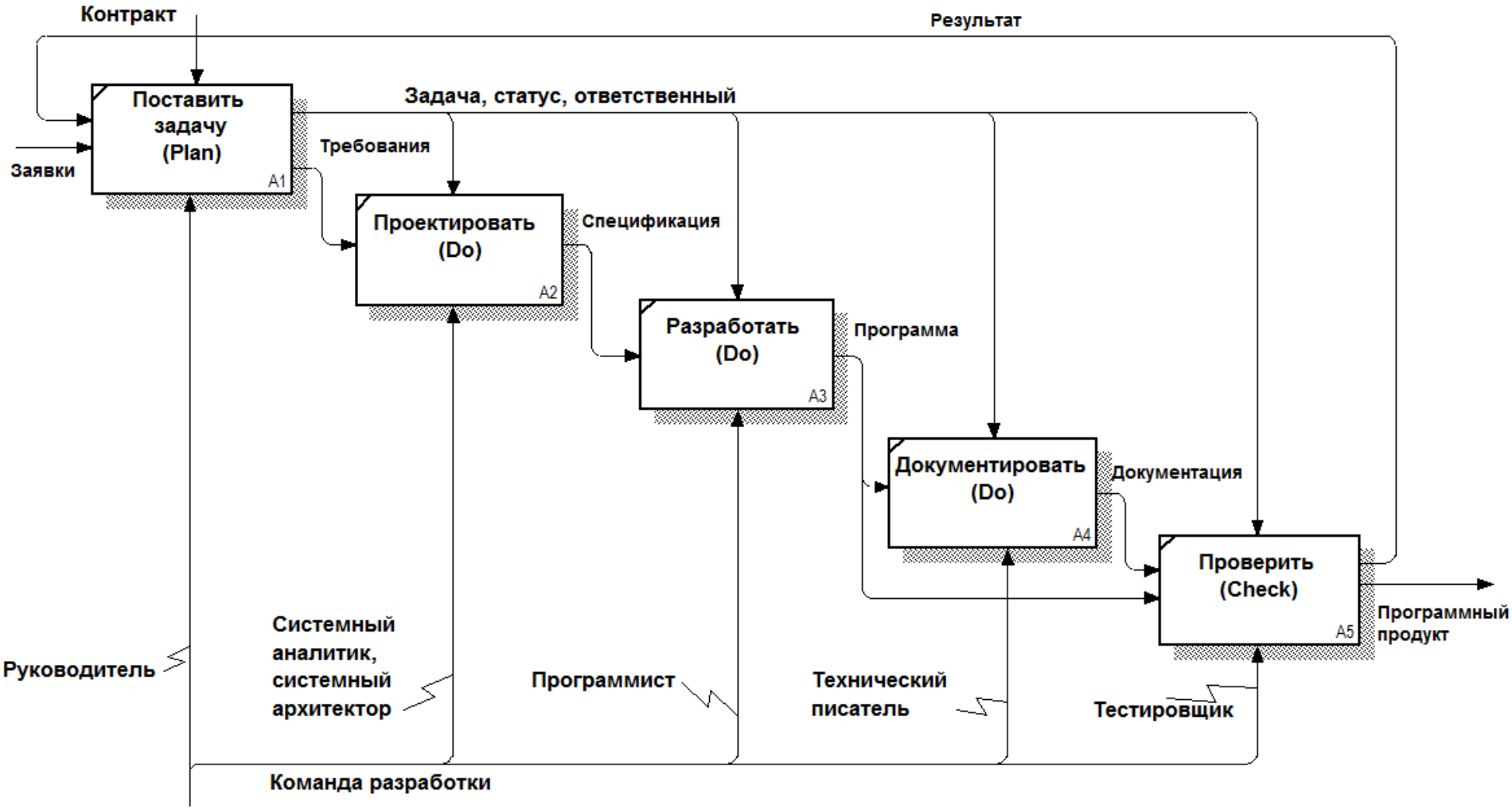


FDD включает в себя пять базовых видов деятельности:

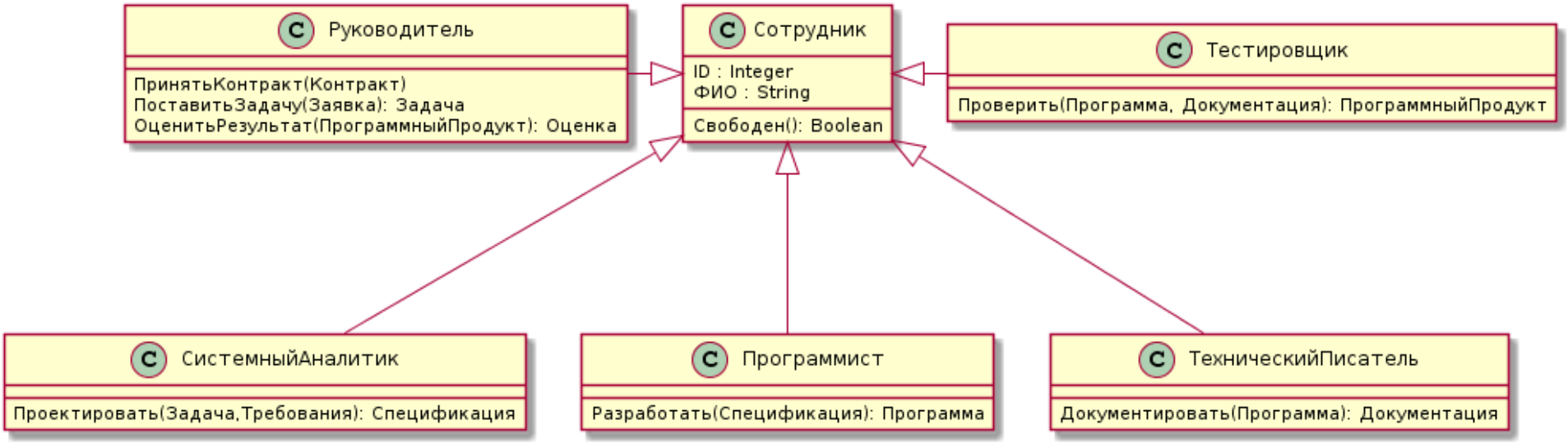
1. разработка **общей модели**
2. составление **списка необходимых функций** системы
3. **планирование** работы над каждой функцией
4. **проектирование** функции
5. **реализация** функции

https://ru.wikipedia.org/wiki/Feature_driven_development
http://www.intuit.ru/studies/courses/3505/747/lecture/26289?page=1#sect3
http://www.intuit.ru/studies/courses/2188/174/lecture/4730?page=2

Модель FDD и команда



Модель FDD и команда



http://www.intuit.ru/studies/courses/3467/709/lecture/21825
http://www.intuit.ru/studies/courses/3467/709/lecture/21825?page=2

Реинжиниринг

Обра́тная разрабо́тка (**обратное проектирование, обратный инжиниринг**, **реверс-инжиниринг**; англ. *reverse engineering*) — исследование некоторого готового устройства или программы, а также документации на него с целью понять принцип его работы; например, чтобы

- обнаружить недокументированные возможности (в том числе программные закладки),
- сделать изменение или
- воспроизвести устройство, программу или иной объект с аналогичными функциями, но без прямого копирования

Применяется обычно в том случае, если **создатель** оригинального объекта **не предоставил** информации о **структуре** и **способе создания** (производства) объекта

Правообладатели таких объектов могут заявить, что проведение **обратной разработки** или использование её результатов нарушает их исключительное право по закону об авторском праве и патентному законодательству

Реинжиниринг

Реинжиниринг программного обеспечения — процесс создания новой функциональности или устранения ошибок, путём революционного изменения, но используя уже имеющееся в эксплуатации программное обеспечение

Как правило, утверждается, что «легче разработать новый программный продукт». Это связано со следующими проблемами:

- реинжиниринг, чаще всего, дороже разработки нового программного обеспечения, так как требуется убрать ограничения предыдущих версий, при этом сохранив с ними совместимость
- реинжиниринг не может сделать программист низкой и средней квалификации — даже профессионалы часто не могут качественно реализовать его, поэтому требуется работа программистов с большим опытом переделки программ и знанием различных технологий
- разработчику бывает сложно разбираться в чужом исходном коде — это вынуждает адаптироваться к восприятию незнакомого стиля программирования, расходует время на всесторонний анализ и освоение реализованных в проекте концепций, используемых в нём сторонних библиотек, требует скрупулёзно исследовать принцип действия всех плохо документированных участков кода — и всё это лишь осложняет процесс перехода продукта на новые архитектурные решения
- кроме того, сам характер деятельности требует дополнительной мотивации: по сравнению с созданием новых продуктов, переработка уже имеющихся не всегда приносит столь же наглядные и впечатляющие результаты, зачастую отягощает грузом технического долга и оставляет мало места для профессионального самовыражения.

Реинжиниринг в FDD

Гарантированно есть список **решенных задач** и **возможно** есть **набор тестов**, связанных или не связанных с задачами

Шаг 1. Восстановление требований
Вход: **задачи**
Выход: пользовательские **истории**

Эпик, тема, история (мнение)

Реинжиниринг в TFD/TDD/BDD

Гарантированно есть список **решенных задач** и **точно** есть **набор тестов**, связанных с задачами

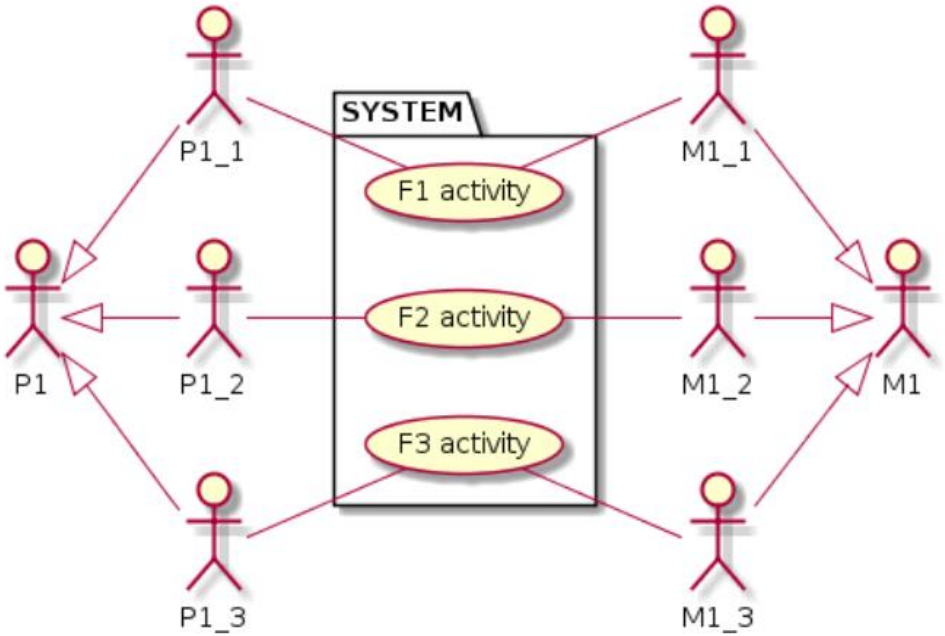
Шаг 1. Восстановление требований
Вход: **тесты**
Выход: пользовательские **истории**

Эпик, тема, история (мнение)

Реинжиниринг в TFD/TDD/BDD

Гарантированно есть список **решенных задач** и **точно** есть **набор тестов**, связанных с задачами

Шаг 2. Восстановление прецедентов
Вход: **истории**
Выход: **прецеденты**



Эпик, тема, история (мнение)

Прецеденты: IDEF0 → Use Case

Описание решения

Общее решение состоит в следующей ассоциации элементов диаграммы IDEF0 с элементами диаграммы прецедентов:

- стрелки механизмов преобразуются в "actor";
- декомпозируемые механизмы становятся родительскими "actor";
- имена блоков становятся именами прецедентов;
- все блоки дочерней диаграммы объединяются в один пакет с именем родительской.

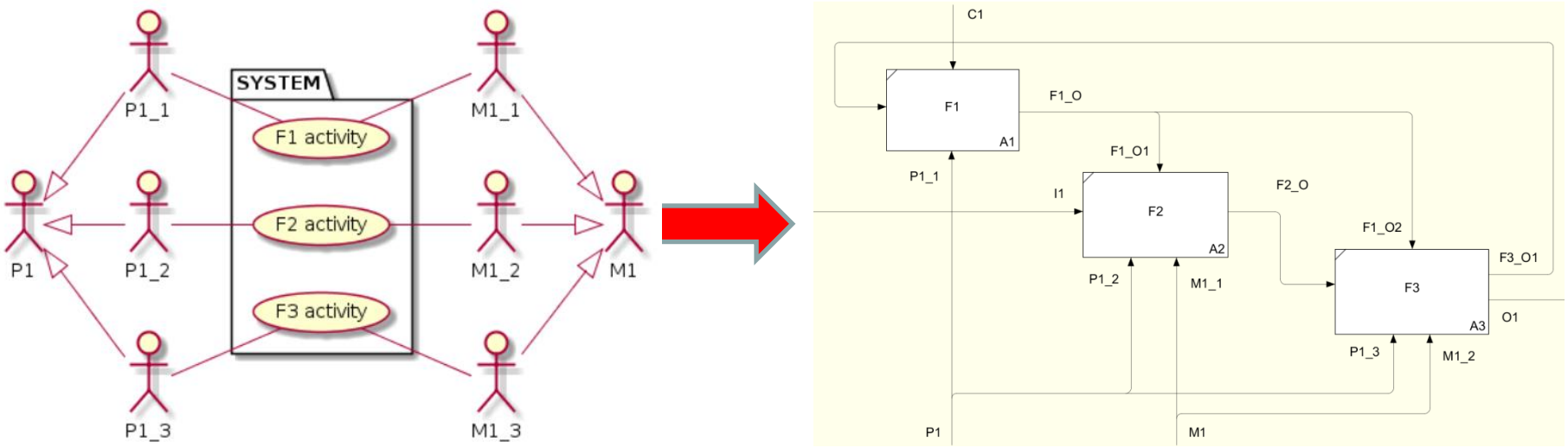
Особенности преобразования

При преобразовании диаграмм IDEF0 в диаграммы прецедентов UML **теряется** информация **обо всех** информационных и материальных потоках - о входах, выходах и управлении

При обратном преобразовании все эти потоки должны быть **восстановлены** или **спроектированы** заново

Реинжиниринг в FDD/TFD/TDD/BDD

Шаг 3. Восстановление функциональной структуры
Вход: **прецеденты**
Выход: **модель процессов**

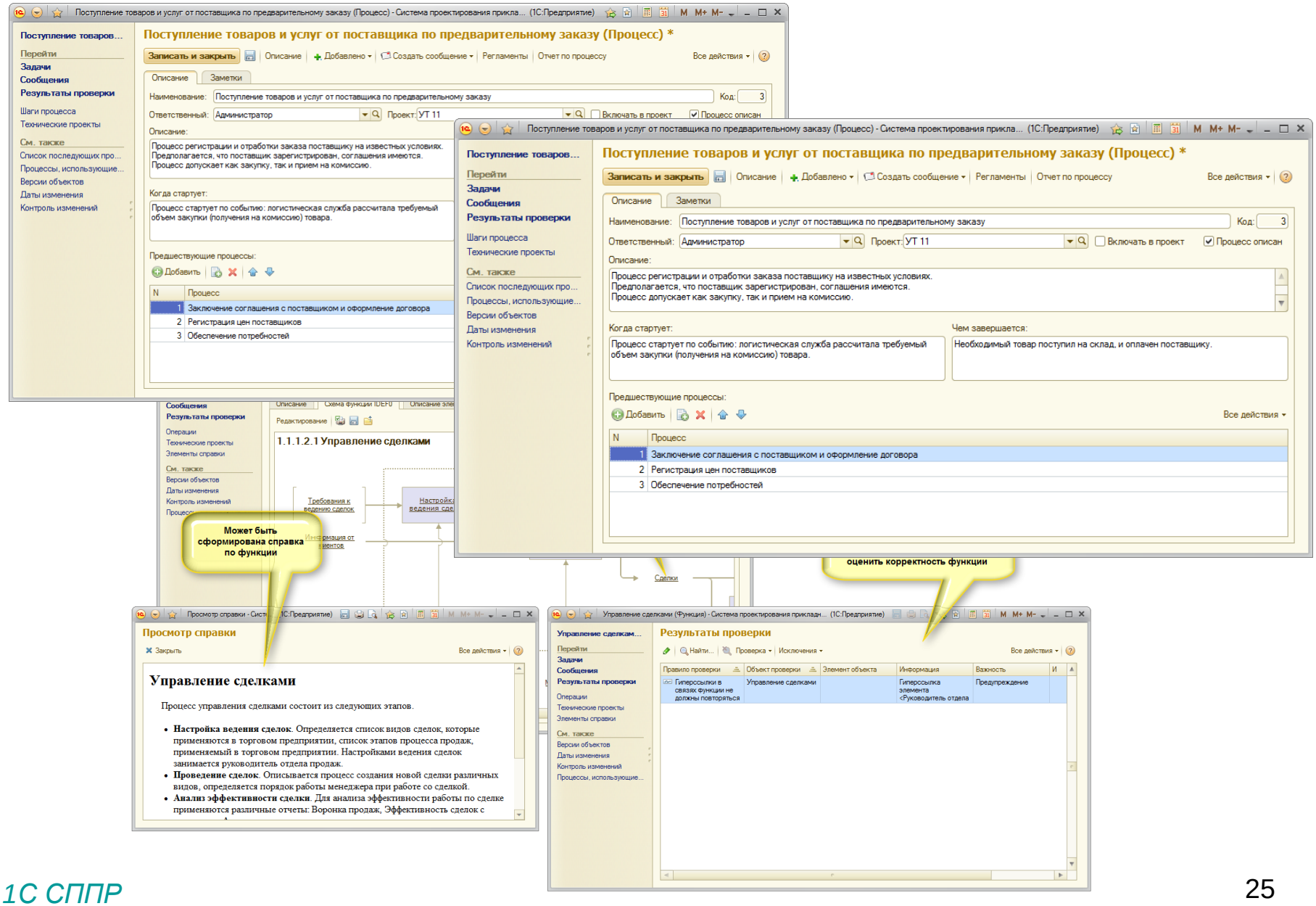


При выполнении преобразования необходимо дополнить (обогатить) модель данными обо всех потоках

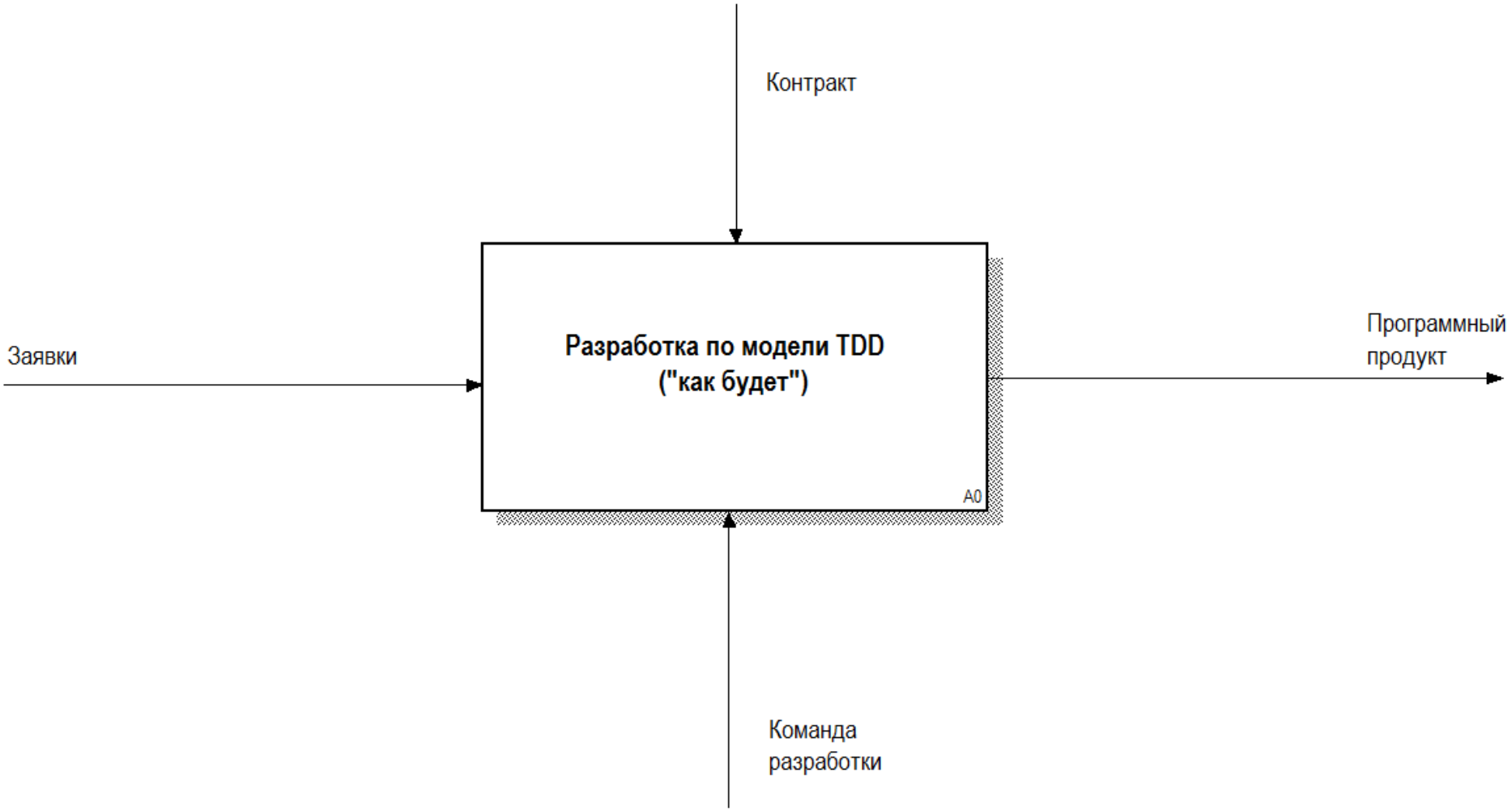
В разработке, управляемой функциональностью или тестами, конфигурация программного и информационного обеспечения всегда может быть восстановлена с трудозатратами, сравнимыми с проектированием!

Преобразования Use Case

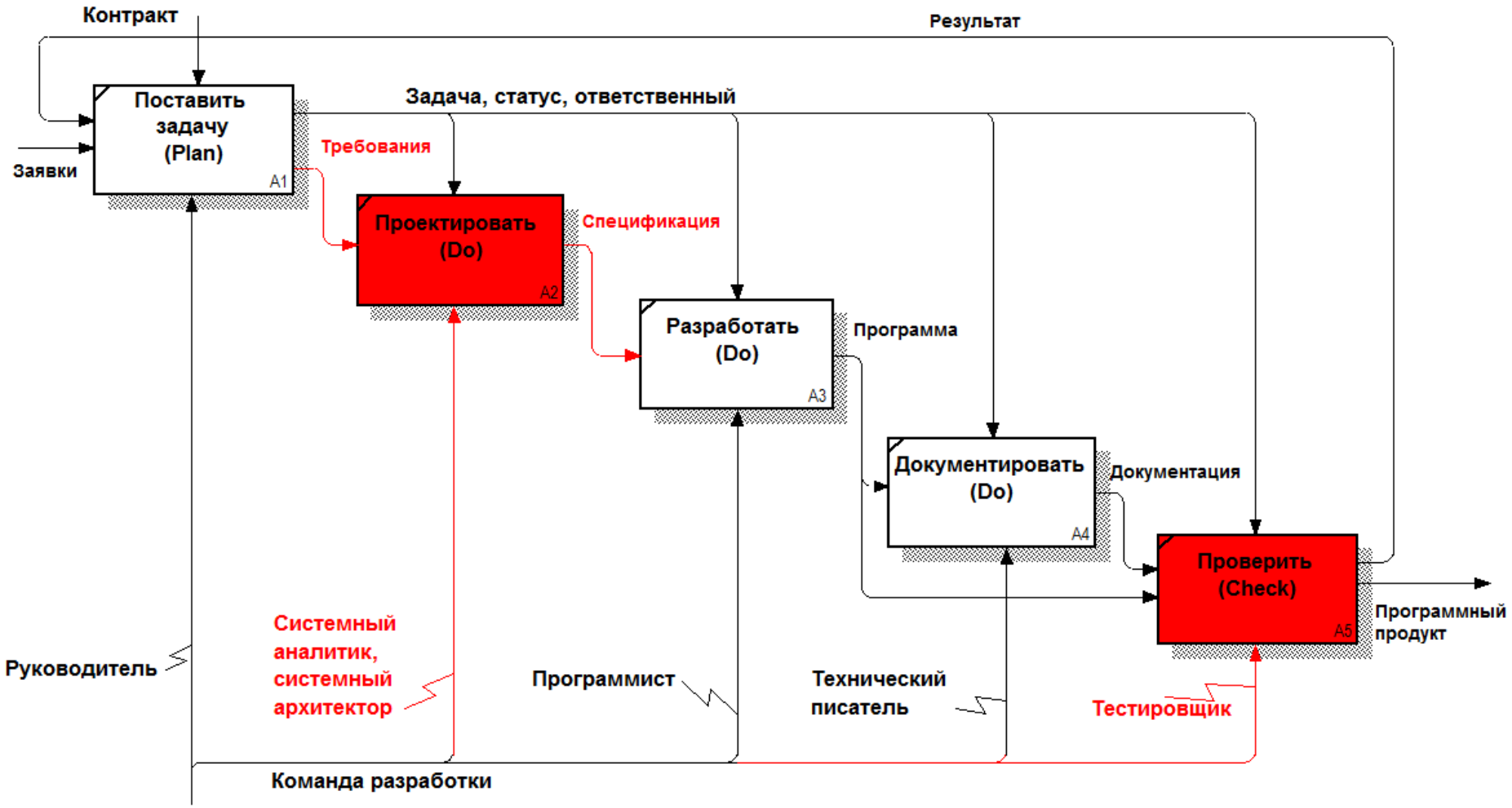
Реинжиниринг в FDD/TFD/TDD/BDD



Модель TDD


Контракт
Заявки
Разработка по модели TDD
("как будет")
A0
Программный
продукт
Команда
разработки

Модель TDD и команда



Рефакторинг в TDD/TFD

Разработка через тестирование (*test-driven development, **TDD***) — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам

TDD цикл включает в себя пять основных шагов:

1. Быстро добавить тест
2. Выполнить все тесты и увидеть, что новый тест "падает"
3. Выполнить небольшое изменение системы
4. Убедиться, что все тесты проходят
5. Выполнить **рефакторинг**, удаляя дублирование

В модели TDD **тест** всегда пишется **прежде** чем создается соответствующий **программный элемент**

Если далее не выполнять шаги 2, 4, 5 то получится модель **TFD** (разработка "вначале тест«, test first development).

Test-driven_development

Рефакторинг в TDD/TFD

Рефакторинг — процесс изменения **внутренней структуры** программы, не затрагивающий её **внешнего поведения** и имеющий целью:

- **облегчить понимание** её работы
- **устранить дублирование** кода
- облегчить **внесение изменений** в ближайшем будущем

Рефакторинг, рассматриваемый защитниками agile как техника "Или – Или", может быть лучше использован как "И" техника: он работает наилучшим образом, когда комбинируется с идеями, отвергаемыми аджилистами, в данном случае с предваряющим анализом

Никакое количество **рефакторинга** неспособно скорректировать **дефектную архитектуру.** Первичная обязанность любого проектировщика – идентифицировать фундаментальные абстракции, составляющие скелет архитектуры

Сделаете это нужным образом, и вам останется еще много работы, которую необходимо выполнить. Но если ошибиться на этом этапе, в конечном счете вам придется (метафору выберите сами) ставить латку на латку, тушить огонь керосином, лепить пластырь на пластырь

Рефакторинг в Agile (Интуит)

Рефакторинг в TDD/TFD

Код с запашко́м (**код с душко́м**, **дурно пахнущий код** англ. *code smell*) — термин, обозначающий код с признаками (запахами) проблем в системе

Был введён Кентом Беком и использован Мартином Фаулером в его книге *Рефакторинг. Улучшение существующего кода*

Запахи кода — это ключевые **признаки** необходимости рефакторинга

Существуют запахи, специфичные как для парадигм программирования, так и для конкретных языков

Основной проблемой, с которой сталкиваются разработчики при борьбе с запахами кода, является то, что критерии своевременности рефакторинга невозможно чётко формализовать без апелляции к эстетике и условному чувству прекрасного

Запахи кода — это **не набор** чётких **правил**, а описание **мест**, на которые нужно **обращать внимание** при рефакторинге. Они легко обнаруживаются, но при этом не во всех случаях свидетельствуют о проблемах

Рефакторинг в TDD/TFD

Комментарии

Часто комментарии играют роль «**дезодоранта**» кода, который появляется в нём лишь потому, что код плохой

Почувствовав потребность написать комментарий, попробуйте изменить структуру кода так, чтобы любые **комментарии стали излишними:**

- если для объяснения действий блока всё же требуется комментарий, попробуйте применить «**Выделение метода**» (Extract Method)
- если метод уже выделен, но по-прежнему нужен комментарий для объяснения его действия, воспользуйтесь «**Переименованием метода**» (Rename Method)
- если требуется изложить некоторые правила, касающиеся необходимого состояния системы, примените «**Введение утверждения**» (Introduce Assertion)

Код с запашком (Интуит)

Документирование программного кода

```
/*!
        \brief Дочерний класс
        \author Norserium
        \version 1.0
        \date Март 2015 года
        \warning Данный класс создан только в учебных целях

        Обычный дочерний класс, который отнас
*/
class Son : public Parent {
public:
        Son();
        ~Son();
};
```

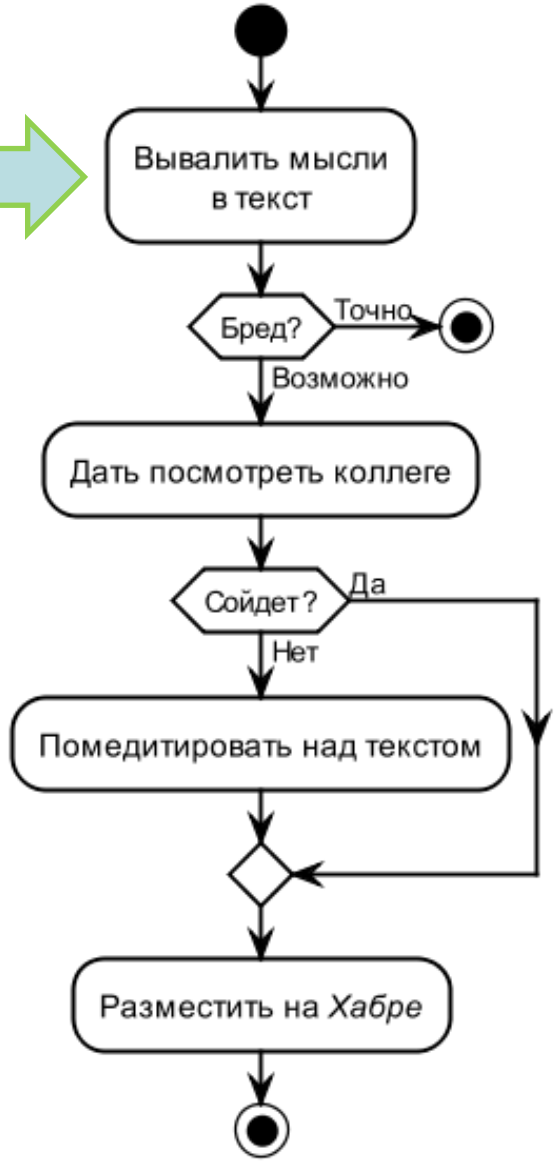
```
/*
    The main Math class
    Contains all methods for performing basic math functions
*/
/// <summary>
/// The main Math class.
/// Contains all methods for performing basic math functions.
/// </summary>
public class Math
{
    // Adds two integers and returns the result
    /// <summary>
    /// Adds two integers and returns the result.
    /// </summary>
    public static int Add(int a, int b)
    {
        // If any parameter is equal to the max value of an integer
        // and the other is greater than zero
        if ((a == int.MaxValue && b > 0) || (b == int.MaxValue && a > 0))
            throw new System.OverflowException();

        return a + b;
    }
}
```

Документирование в комментариях (Пример)

Документирование программного кода

```
start
:Вывалить мысли
в текст;
if (Бред?) then (Возможно)
    :Дать посмотреть коллеге;
    if (Сойдет?) then (Да)
    else (Нет)
        :Помедитировать над текстом;
    endif
    :Разместить на //Хабре//;
else (Точно)
    stop
endif
stop
```

PlantUML




```
if (a==true) { // если а верно
  b = b + 1;   // увеличить b
}
```

```
if (a) { // если найден элемент
  b++;   // указатель вперед на 1 слово
}
```

```
if (item_found) {
  next_item_pointer++;
}
```

PlantUML (Пример)