

# **Основные задачи и методы проектирования программных средств**

## **Лекция 10**

### **История развития итеративной разработки программных средств**

**Овчинников П.Е.**

**МГТУ «СТАНКИН»,**

**ст.преподаватель кафедры ИС**

# История: RUP

**Rational Unified Process (RUP)** — [методология](#) разработки программного обеспечения, созданная компанией [Rational Software](#) (**1996-2003**)

В основе RUP лежат следующие **принципы**:

- Ранняя идентификация и непрерывное (до окончания [проекта](#)) устранение **основных рисков**
- Концентрация на выполнении **требований заказчиков** к исполняемой программе - анализ и построение **модели [прецедентов](#) (вариантов использования)**
- Ожидание **изменений в требованиях**, проектных решениях и реализации в процессе разработки
- [Компонентная архитектура](#), реализуемая и **тестируемая на ранних стадиях** проекта
- Постоянное **обеспечение качества** на всех этапах разработки [проекта](#) (продукта)
- Работа над проектом в сплочённой команде, **ключевая роль** в которой принадлежит **архитекторам**

# История: борьба за качество

## ГОСТ Р ИСО 9000-2015 Системы менеджмента качества. Основные положения и словарь

### качество (quality)

степень **соответствия** совокупности присущих **характеристик** **объекта** **требованиям**

### требование (requirement)

**потребность** или **ожидание**, которое установлено, обычно предполагается или является обязательным.

### объект (object), сущность (entity), элемент (item)

что-либо **воспринимаемое** или **воображаемое**.

Примечание - объекты могут быть:

- **материальными** (например, двигатель, лист бумаги, алмаз),
- **нематериальными** (например, коэффициент конверсии, план проекта) или
- **воображаемыми** (например, будущее положение организации).

# История: борьба за качество

ГОСТ Р ИСО/МЭК 25010-2015 Информационные технологии (ИТ).  
Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов

Systems and software Quality Requirements and Evaluation  
System and software quality models



# История: борьба за качество

ГОСТ Р ИСО/МЭК 25010-2015 Информационные технологии (ИТ).  
Системная и программная инженерия. Требования и оценка качества систем и программного обеспечения (SQuaRE). Модели качества систем и программных продуктов

Systems and software Quality Requirements and Evaluation  
System and software quality models



# История: управление рисками

[ГОСТ Р 51897-2011](#)/Руководство ИСО 73:2009 Менеджмент риска. Термины и определения

## риск:

следствие влияния неопределенности на достижение поставленных целей

Риск часто **характеризуют** путем описания возможного **события** и его **последствий** или их сочетания.

Риск часто **представляют** в виде **последствий** возможного **события** (включая изменения обстоятельств) и соответствующей **вероятности**.

**Неопределенность** - это **состояние** полного или частичного **отсутствия информации**, необходимой для понимания события, его последствий и их вероятностей.

[ГОСТ Р 52806-2007](#) Менеджмент рисков проектов. Общие положения

## воздействие риска:

мера последствия риска

## вторичный риск:

риск, возникающий в результате рассмотрения проблем, связанных с риском.

# История: управление рисками

## ГОСТ Р 52806-2007 Менеджмент рисков проектов. Общие положения

Анализ рисков может проводиться с использованием **качественных** и **количественных** методов, что зависит от характера и качества имеющихся данных. Пример классификационной матрицы для проведения качественного анализа приведен в таблице.

Классификация рисков зависит от оценки существующих мер и процедур менеджмента. Такая форма классификации может применяться как в отношении **последствий** (угроз), так и в отношении **возможности** их **предотвращения** или снижения.

### Матрица качественного анализа рисков

Степень вероятности риска	Степень воздействия риска (ущерб)		
	минимальная	средняя	значительная
Вероятно	Средняя	Высокая	Высокая
Возможно	Низкая	Средняя	Высокая
Маловероятно	Низкая	Низкая	Средняя

# История: управление рисками

[ГОСТ Р 52806-2007](#) Менеджмент рисков проектов. Общие положения

Соответствие между **угрозами** и **возможностями**





# RUP: итеративная модель

RUP использует **итеративную модель разработки**.

В конце каждой **итерации** (в идеале продолжающейся от 2 до 6 недель) **проектная команда** должна:

- **достичь** запланированных на данную итерацию **целей**,
- **создать** или **доработать** проектные **артефакты** и
- **получить** промежуточную, но **функциональную версию** конечного продукта

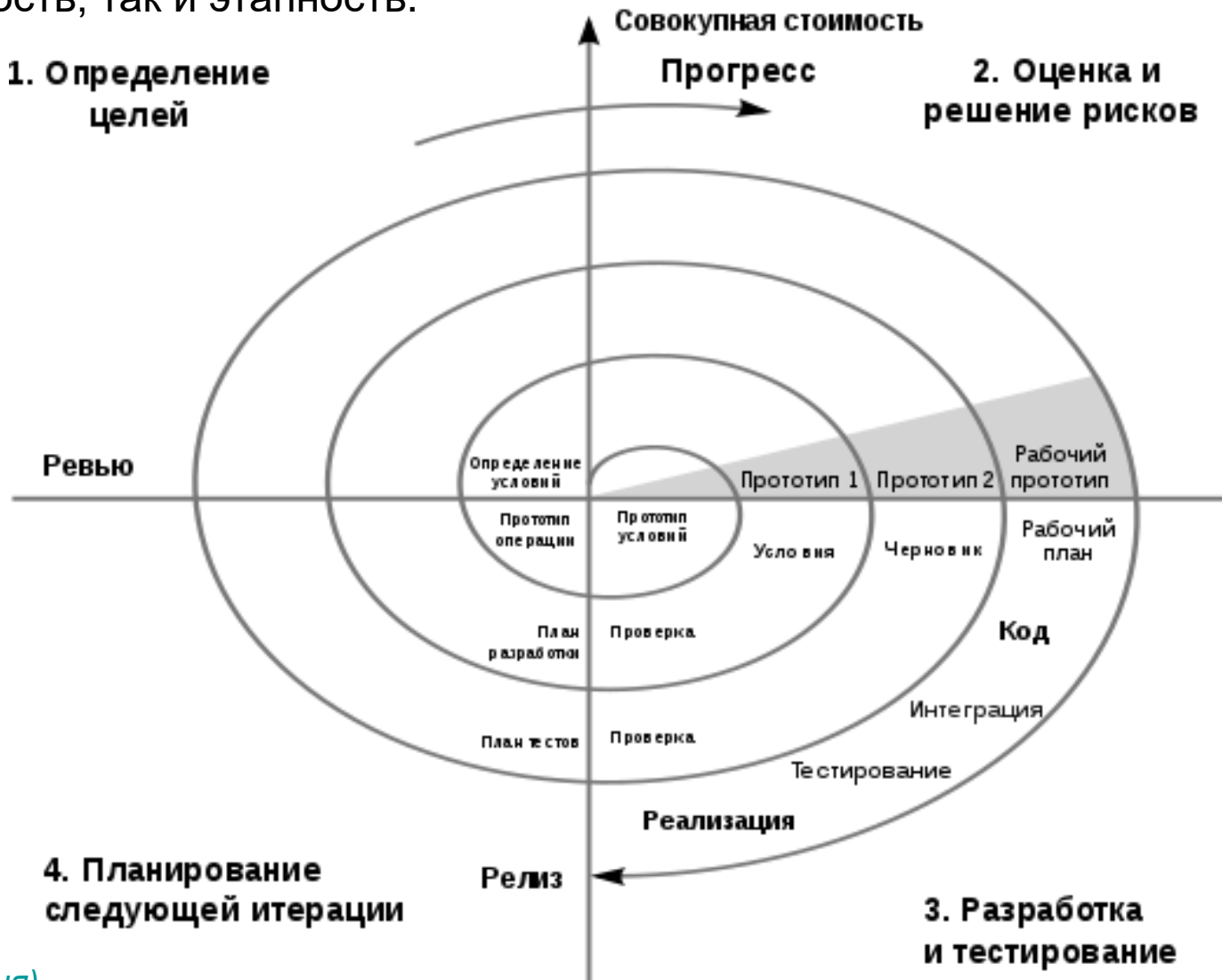
Итеративная разработка позволяет:

- **быстро реагировать** на меняющиеся требования,
- **обнаруживать и устранять риски** на ранних стадиях проекта, а также
- эффективно **контролировать качество** создаваемого продукта

Первые идеи итеративной модели разработки были заложены в ["спиральной модели"](#)

# RUP: спиральная модель

Спиральная модель, предложенная Барри Боэмом в **1986 году**, представляет собой процесс разработки программного обеспечения, сочетающий в себе как итеративность, так и этапность.



# RUP: управление рисками

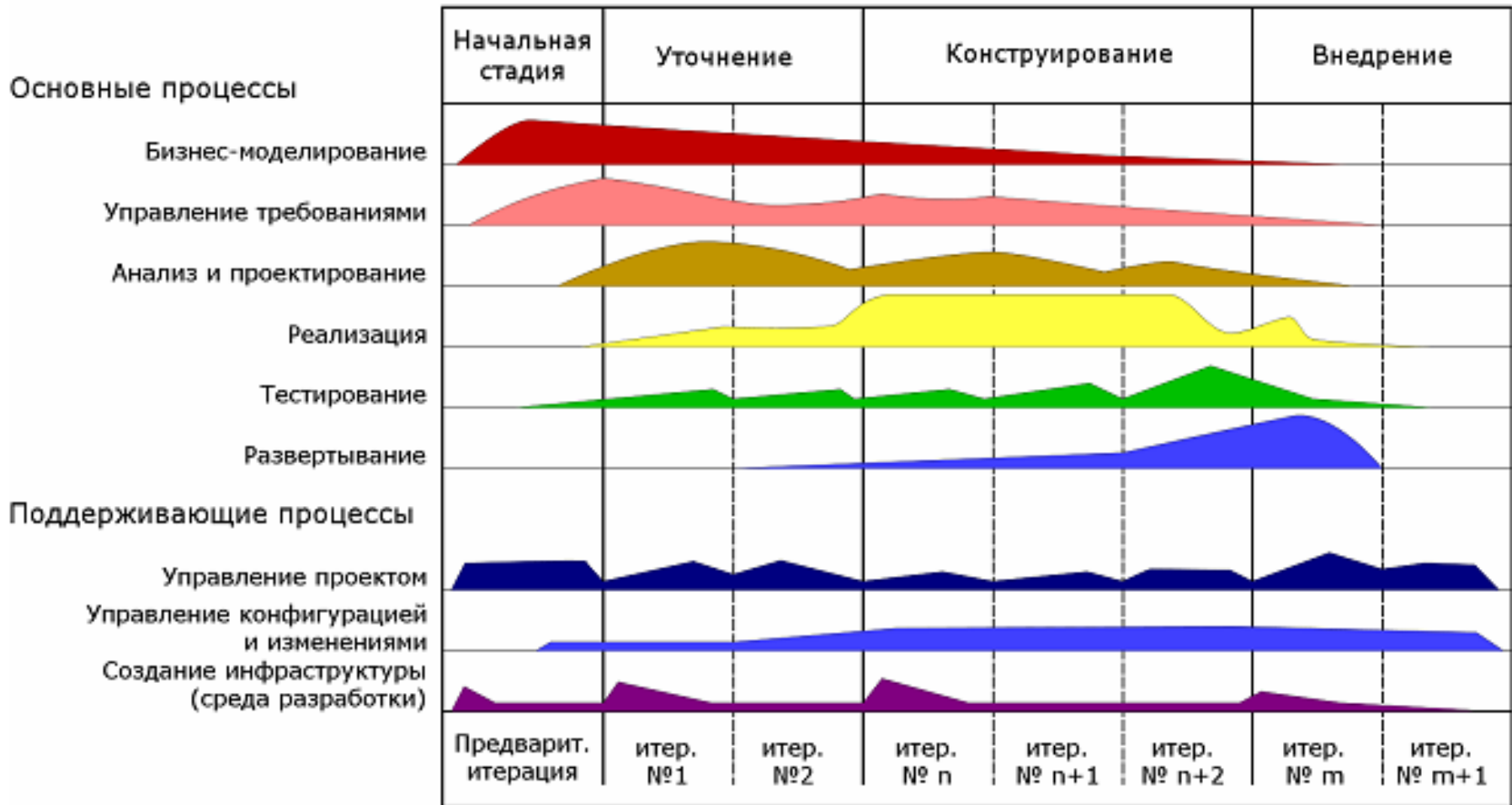
Боэм формулирует десять **наиболее распространённых (по приоритетам) рисков**:

1. Дефицит специалистов
2. Нереалистичные сроки и бюджет
3. Реализация несоответствующей функциональности
4. Разработка неправильного пользовательского интерфейса
5. «Золотая сервировка», перфекционизм, ненужная оптимизация и оттачивание деталей
6. Непрерывающийся поток изменений
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлечённых в интеграцию
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами
9. Недостаточная производительность получаемой системы
10. Разрыв между квалификацией специалистов и требованиями проекта

# RUP: процессы и стадии

Рабочие процессы

Стадии



Итерации

# RUP: стадии (фазы)

## 1. Начальная стадия (Inception):

- Формируются видение и границы проекта
- Создается экономическое обоснование (business case)
- Определяются основные требования, ограничения и ключевая функциональность продукта
- Создается базовая версия [модели прецедентов](#)
- Оцениваются риски

После завершения этих шагов проект проверяется по следующим критериям:

- [согласие заинтересованных сторон](#) по определению объема работ и оценке стоимости / графика.
- понимание требований, подтвержденное верностью основных вариантов использования
- достоверность оценок затрат / графика, приоритетов, рисков и процесса разработки
- глубина и широта каждого разработанного архитектурного прототипа.
- установление базового уровня для сравнения фактических расходов с запланированными

Если проект **не проходит** эту веху, называемую целевой вехой жизненного цикла, ее можно либо **отменить**, либо **повторить** после перепроектирования.

# RUP: стадии (фазы)

**2. Уточнение (Elaboration)** - производится анализ предметной области и построение исполняемой архитектуры. Результатом этапа уточнения является:

- модель **вариантов использования**, в которой были определены варианты использования и субъекты, а также разработана большая часть описаний вариантов использования; модель вариантов использования должна быть **завершена на 80%**
- описание **архитектуры** программных средств в процессе разработки программной системы
- исполняемая архитектура , которая реализует архитектурно значимые прецеденты
- пересмотренное **экономическое обоснование** и **список рисков**
- **план развития** всего проекта
- **прототипы**, которые наглядно снижают каждый выявленный технический риск
- предварительное руководство пользователя (необязательно)

Если проект не может пройти этот рубеж, еще есть время его **отменить** или **изменить**. Однако после выхода из этого этапа проект переходит в операцию с высоким риском, когда изменения намного сложнее и вреднее.

# RUP: стадии (фазы)

**3. Построение (Construction)** - происходит реализация большей части функциональности продукта.

Фаза завершается **первым внешним релизом** системы и вехой **начальной функциональной готовности** (Initial Operational Capability)

**4. Внедрение (Transition)** - создается финальная версия продукта и передается от разработчика к заказчику.

Это включает в себя:

- программу бета-тестирования,
- обучение пользователей, а также
- определение качества продукта

В случае, если качество не соответствует ожиданиям пользователей или критериям, установленным в фазе Начало, **фаза** Внедрение **повторяется снова**

Выполнение всех целей означает достижение вехи готового продукта (Product Release) и завершение полного цикла разработки

# RUP: строительные блоки

RUP основан на наборе **строительных блоков** и **элементов контента**, описывающих:

- **что** должно быть произведено,
- какие **навыки** для этого необходимы и
- пошагового объяснения, описывающего, **как** конкретные цели разработки должны быть достигнуты.

Основные строительные блоки или элементы содержимого:

- **роли (кто)**  
роль определяет набор связанных **навыков, компетенций** и **обязанностей**
- **рабочие продукты (что)**  
рабочий продукт представляет собой нечто, являющееся **результатом задачи**, включая все **документы** и **модели**, созданные в процессе разработки
- **задачи (как)**  
задача описывает **единицу работы**, назначенную роли, которая обеспечивает значимый результат



# RUP: дисциплины

В рамках **каждой итерации** задачи разделены на девять дисциплин:

Шесть «инженерных дисциплин»

- бизнес-моделирование
- требования
- **анализ и проектирование**
- реализация
- испытания
- развертывание

Три вспомогательных дисциплины

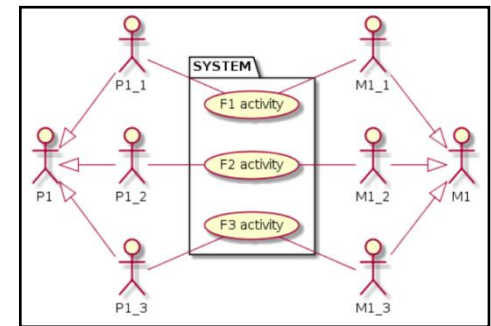
- [конфигурация и управление изменениями](#)
- [управление проектом](#)
- [окружающая среда](#)

# RUP: прецеденты

Для отражения модели прецедентов на диаграмме используются:

- **рамки** системы ([англ. system boundary](#)) — **прямоугольник** с названием в верхней части и эллипсами (прецедентами) внутри. Часто может быть опущен без потери полезной информации
- **актёр** (англ. *actor*) — стилизованный **человечек**, обозначающий набор **ролей** пользователя (понимается в широком смысле:

- **человек**
- **внешняя сущность**
- **класс**
- **другая система**



взаимодействующего с некоторой сущностью (системой, подсистемой, классом)

Актёры не могут быть связаны друг с другом (за исключением отношений обобщения/наследования)

- **прецедент** — **эллипс** с надписью, обозначающий выполняемые системой **действия** (могут включать возможные варианты), приводящие к наблюдаемым актёрами результатам

# RUP: прецеденты

## ИТЕРАТИВНАЯ РАЗРАБОТКА

Сценарий использования, вариант использования,

### **прецедент использования**

(англ. *use case*) — в разработке программного обеспечения и системном проектировании это описание поведения системы, когда она взаимодействует с кем-то (или чем-то) из внешней среды

Методика сценариев использования применяется для выявления требований к поведению системы, известных также как **пользовательские и функциональные требования**

## ГИБКАЯ РАЗРАБОТКА

**Пользовательские истории** (англ. *User Story*) — способ **описания требований** к разрабатываемой системе, сформулированных **как одно или более предложений** на повседневном или деловом языке пользователя

Пользовательские истории используются гибкими методологиями разработки программного обеспечения для спецификации требований (вместе с приёмочными испытаниями)

# RUP: прецеденты

## Полный формат описания прецедента:

### 1. Идентификатор прецедента

Как можно идентифицировать прецедент в его контексте?

### 2. Название прецедента

Как называется процесс деятельности (activity)?

### 3. Контекст

Что является внешней средой для описываемой деятельности?

### 4. Участники (actors) и цели (goals)

Кто или что взаимодействует с системой для достижения определенных целей?

### 5. Предусловия (pre-conditions)

Что должно произойти, прежде чем прецедент может стартовать?

### 6. Постусловия (post-conditions)

Что является успешным результатом?

### 7. Основной поток (main flow)

Что нужно сделать для перехода от предусловий к постусловиям (участники, действие (activity), результаты)?

### 8. Исключения (exceptions)

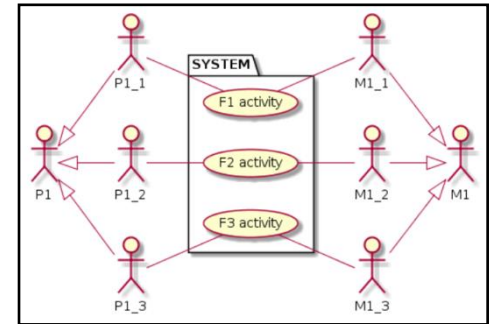
Что может пойти не так (условия (риск), последствия, реакция)?

### 9. Альтернативы (alternates)

Что может повлиять на путь перехода от предусловий к постусловиям?

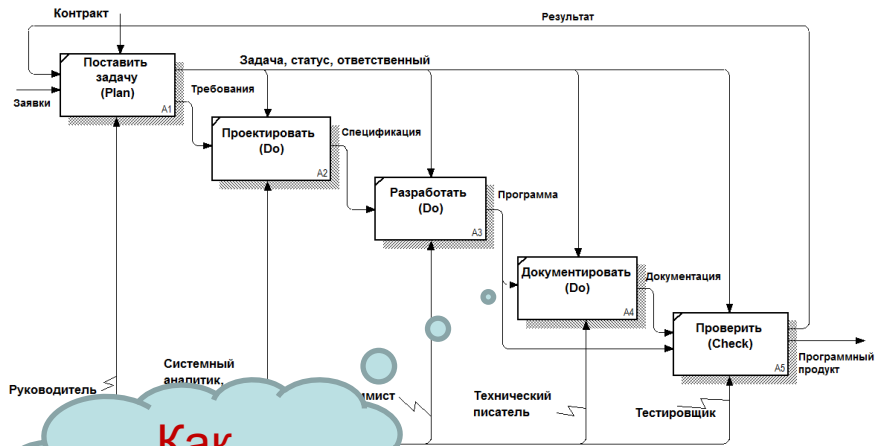
### 10. Временные параметры

Каков триггер (событие, стартующее прецедент), каковы частота повторения и продолжительность?

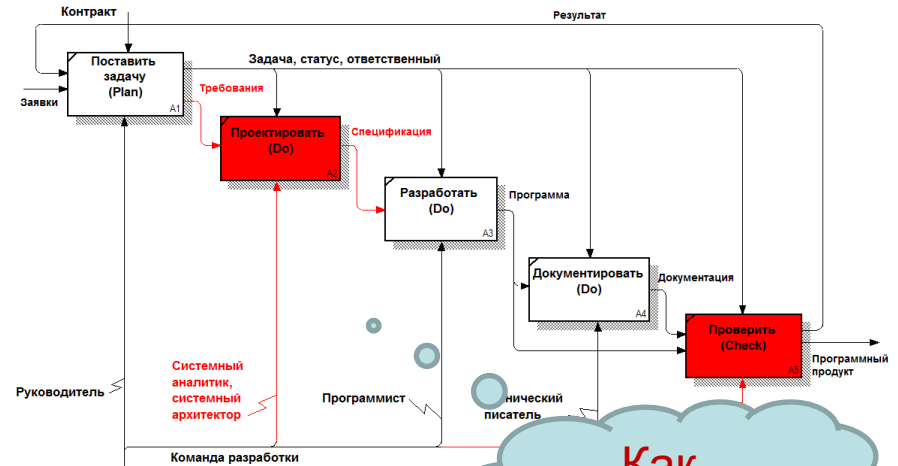


# Проблематика цикла PDC(A): через A (act) могут изменяться процессы

AsIs (как есть)



ToBe (как будет)

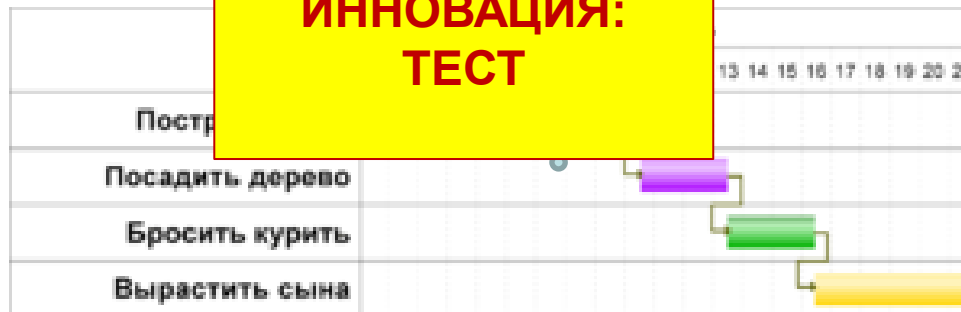


Как узнать?

Как выбрать?

ToDo (что сделать?) =

**ИННОВАЦИЯ:  
ТЕСТ**



## Инновации: история термина

В своей работе «Теория экономического развития» (1911) **Йозеф Шумпетер** впервые рассмотрел вопросы «новых комбинаций» изменений в развитии и дал полное описание инновационного процесса

Термин «инновация» Йозеф Шумпетер стал использовать в 30-е гг. XX века, понимая при этом под **инновацией** изменение с целью внедрения и использования новых видов потребительских товаров, новых производственных, транспортных средств, рынков и форм организации в промышленности

Согласно Йозефу Шумпетеру, **инновация** является главным **источником прибыли**: «прибыль, по существу, является результатом выполнения новых комбинаций», «без развития нет прибыли, без прибыли нет развития»

**Инновации** – это **экономический** термин!

# Планирование организационных инноваций

Федеральный закон "О науке и государственной научно-технической политике" от 23.08.1996 N 127-ФЗ

## Инновации

**введенный в употребление** новый или значительно улучшенный **продукт** (товар, услуга) или **процесс**, новый **метод продаж** или новый **организационный метод** в деловой практике, организации рабочих мест или во внешних связях.

«Рекомендации по сбору и анализу данных по инновациям»  
(«[Руководство Осло](#)»)

## Инновация

**введение в употребление** какого-либо нового или значительно улучшенного **продукта** (товара или услуги) или **процесса**, нового **метода маркетинга** или нового **организационного метода** в деловой практике, организации рабочих мест или внешних связях

Путем сравнения моделей «как есть» (As Is) и «как должно быть» (To Be) составляется модель «что сделать» (To Do).

# Инновации: какие они бывают

В соответствии с определением:

- **продуктовые** (пример: электронные пропуска)
- **процессные** (пример: цифровое проектирование)
- **маркетинговые** (пример: аренда лицензий)
- **организационные** (пример: социальные сети в учебном процессе)

Принято рассматривать инновации с позиции «**новое для предприятия**», поскольку именно деятельность конкретных предприятий способна сформировать **макроинновации** в отраслях, регионах и целых государствах

Инновации же, в свою очередь, могут состоять из ряда **микроинноваций**, которые проще всего объяснить как изменения в деятельности отдельного сотрудника предприятия или отдельного потребителя

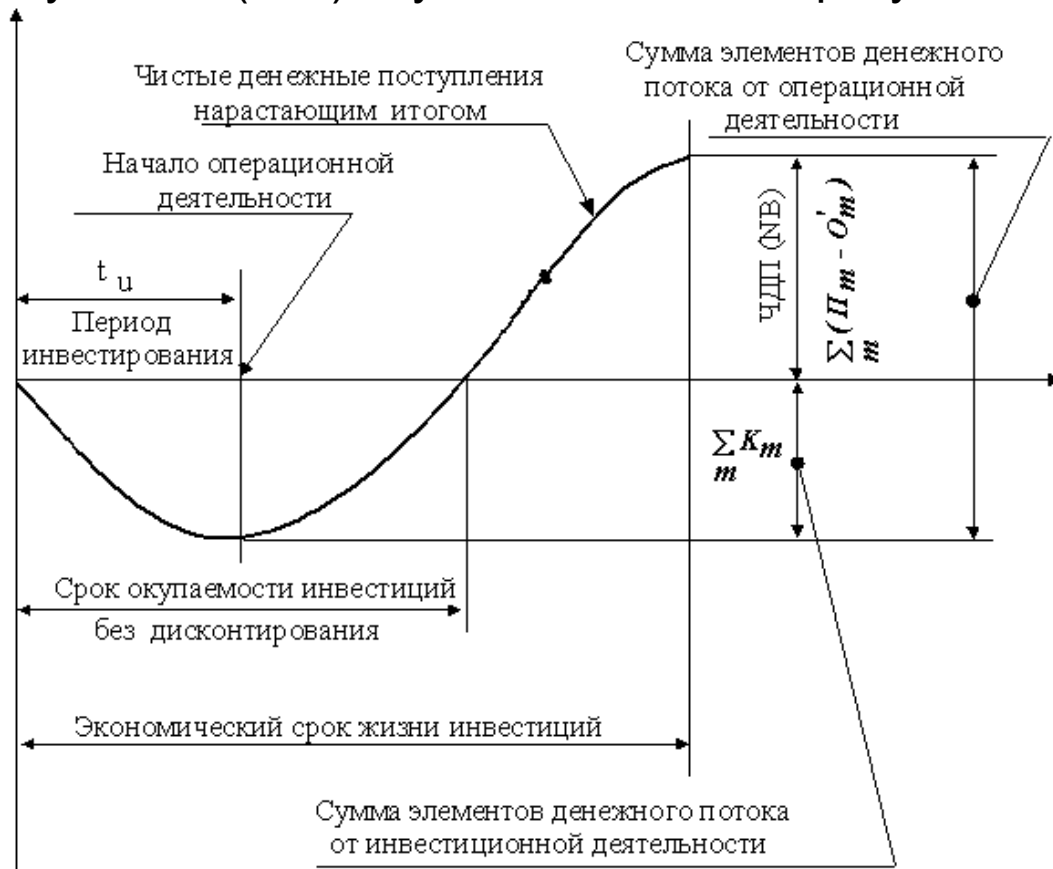


# Планирование организационных инноваций

Федеральный закон "О науке и государственной научно-технической политике" от 23.08.1996 N 127-ФЗ

Инновационный проект - комплекс направленных на достижение экономического эффекта мероприятий по осуществлению инноваций, в том числе по коммерциализации научных и (или) научно-технических результатов.

Инвестиционный проект — экономический или социальный проект, основывающийся на инвестициях; обоснование экономической целесообразности, объёма и сроков осуществления прямых инвестиций в определённый объект, включающее проектно-сметную документацию, разработанную в соответствии с действующими стандартами.



127-ФЗ

Инвестиционное проектирование

# Agile:история

**Гибкая методология разработки** ([англ. Agile software development](#)), **agile-методы** — обобщающий термин для целого ряда подходов и практик, основанных на ценностях [Манифеста гибкой разработки программного обеспечения](#) и 12 принципах, лежащих в его основе

К гибким методологиям, в частности, относят [экстремальное программирование](#), [DSDM](#), [Scrum](#), [FDD](#), [BDD](#) и др.

В течение **1990-х** годов ряд легких методов разработки программного обеспечения развивался в ответ на преобладающие тяжелые методы, которые критики называли чрезмерно регулируемыми, планируемыми и микроуправляемыми.

К ним относятся:

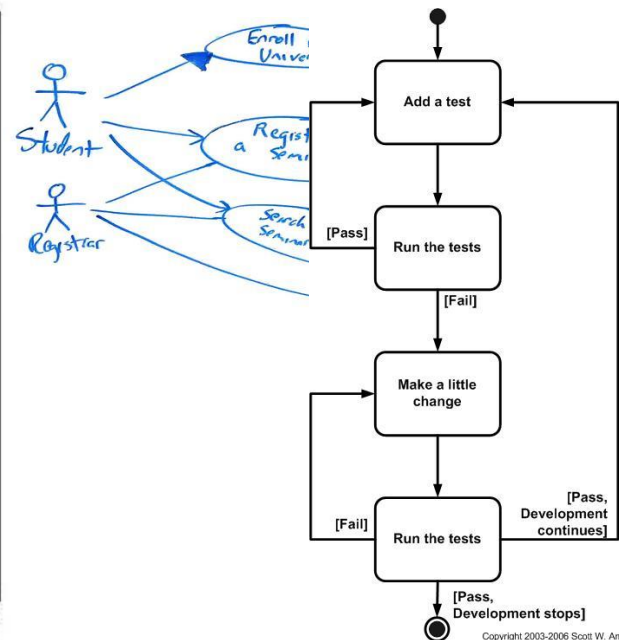
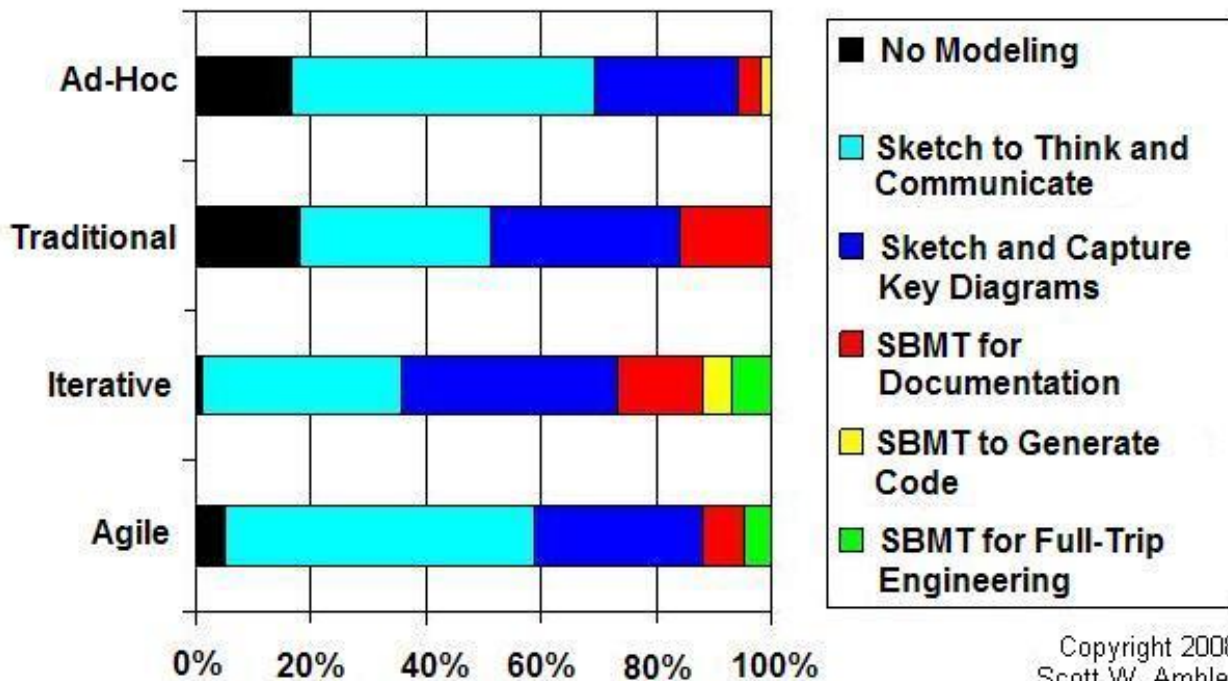
- быстрая разработка приложений (RAD) с **1991** года;
- унифицированный процесс и метод разработки динамических систем с **1994** года
- Scrum с **1995** года
- Crystal Clear и экстремальное программирование (XP), с **1996** года и функционально-ориентированная разработка с **1997** года

# Agile: история

**Гибкий унифицированный процесс (AUP, [англ. Agile Unified Process](#))** - упрощенная версия унифицированного процесса [Unified Process \(UP\)](#). Данная методология разработки программного обеспечения разработана в **2008 году** и соединяет в себе элементы [гибких методологий](#) и унифицированного процесса.

В частности, AUP предполагает разработку через тестирование ([TDD](#)), применение гибкого моделирования ([англ. Agile modeling](#)) и [рефакторинга баз данных](#), гибкое управление изменениями

## Primary Strategy For Modeling



# Agile: история

В отличие от [RUP](#), AUP содержит всего семь дисциплин:

## 1. Моделирование

Выработка понимания прикладной области проекта, устройства бизнеса организации, а также выработка приемлемых решений прикладных проблем, которые требуется разрешить в рамках проекта.

## 2. Реализация

Трансформация моделей в исполнимый код, его тестирование с использованием [модульных тестов](#).

## 3. Тестирование

Объективная оценка качества продукта. Поиск дефектов, проверка корректности спроектированной системы, а также её соответствия требованиям.

## 4. Развертывание

Планирование процедуры развертывания системы, а также исполнение плана развертывания.

## 5. [Управление конфигурациями](#)

Разграничение доступа к артефактам проекта. Контроль всех изменений и версий артефактов проекта.

## 6. [Управление проектом](#)

Направление действий всех участников проекта. Управление рисками, руководство персоналом, координация заинтересованных лиц и внешних систем с целью поставки продукта с соблюдением временных и бюджетных ограничений.

## 7. Организация среды

Обеспечение доступности для членов команды проекта всех необходимых ресурсов, инструкций, стандартов, документов, аппаратных и программных инструментов.

# Agile:история

**Профили жизненного цикла** систем и программного обеспечения и рекомендации для очень **малых предприятий (VSE)**. Очень малая организация (VSE) - это предприятие, организация, отдел или проект, в котором работает до 25 человек. ISO / IEC 29110 - это серия международных стандартов и руководств под названием «Системная и программная инженерия - Профили жизненного цикла для очень малых предприятий (VSE)»

Промышленные и общественные организации (например, правительственные учреждения, некоммерческие организации) признают, что VSE производят ценные продукты и услуги. VSE также разрабатывают и поддерживают системы и программное обеспечение, используемое в более крупных системах, поэтому необходимо признать VSE как поставщиков высококачественных систем и программного обеспечения.

**ISO/IEC TR 29110-1:2016 Systems and software engineering — Lifecycle profiles for Very Small Entities (VSEs) — Part 1: Overview**

**ISO / МЭК 29110-2-1: 2015 Программная инженерия - Профили жизненного цикла для очень малых предприятий (VSE) - Часть 2-1: Структура и таксономия**

# Agile: критика

Начнем с **худшего** в agile подходах – **идей**, которые **вредят процессу** разработки

## **Неприятие предваряющего анализа**

Речь, бесспорно, пойдет о неприятии такой деятельности, как предваряющий анализ, прежде всего предваряющий анализ требований и предваряющее проектирование.

## **Пользовательские истории как основа требований**

В предыдущих главах по разным поводам отмечалась полезность пользовательских историй, в первую очередь, как способ проверки полноты требований.

## **Разработка, базирующаяся на функциях, и игнорирование зависимостей**

Согласно основной идее agile методов, разработка проекта представляет последовательность реализаций индивидуальных функций, выбираемых на каждом шаге на основе их бизнес-стоимости

## **Отказ от средств анализа зависимостей**

Потенциальная сложность взаимодействия функций требует тщательного анализа зависимостей, возникающих между задачами. Проект может опустить этот анализ только на свой собственный риск

## **Отказ от традиционных менеджерских задач**

Самоорганизуемые команды, продвигаемые agile методами, не имеющие менеджера с традиционной обязанностью распределения задач, представляют лучшее решение для немногих команд и не подходят для многих других.

# Agile: критика

Начнем с **худшего** в agile подходах – идей, которые **вредят процессу** разработки

## **Встроенный потребитель**

Идея XP встраивать в команду потребителя не очень хорошо работает на практике по причинам, объясненным в предыдущих обсуждениях. Однако введенное в Scrum понятие владельца продукта фигурирует ниже в списке блестящих идей.

## **Тренер (консультант) как отдельная роль**

Идея Scrum назначения Scrum-мастера хороша для Scrum, но не подходит для большинства проектов. Хорошая разработка требует не тех, кто только говорит, но тех, кто делает.

## **Разработка, управляемая тестами**

Разработка, управляемая тестами, и требование связывания теста с каждым участком кода, реализующим некоторую функциональность, перечисляется ниже в списке хороших и блестящих идей. Это помогает реализовать рефакторинг.

## **Отказ от документов**

Критика agile тяжелого процесса создания документов, мало что дающих конечному потребителю, справедлива для отдельных сегментов индустрии. В некоторых случаях, таких как критически важные системы, немного можно сделать по исправлению ситуации, поскольку для сертификации требуются документы.

# Agile: критика

**Технический долг** (также известный как **долг кодинга**) — это метафора [программной инженерии](#), обозначающая накопленные в [программном коде](#) или [архитектуре](#) проблемы, связанные с пренебрежением к качеству при [разработке программного обеспечения](#) и вызывающие дополнительные затраты труда в будущем.

Технический долг обычно незаметен для конечных пользователей продукта, а связан с недостатками в [сопровождаемости](#), тестируемости, понятности, модифицируемости, [переносимости](#).

По аналогии с финансовым [долгом](#), **технический долг** может обрести «[процентами](#)» — **усложнением** (или даже невозможностью) **продолжения разработки**, дополнительным временем, которое разработчики потратят на изменение программного продукта, исправление ошибок, сопровождение и т. п. Хотя увеличение технического долга как правило негативно влияет на будущее проекта, оно может быть и **сознательным, компромиссным решением**, продиктованным сложившимися обстоятельствами.

Сам по себе [плохой код](#) не всегда является техническим долгом, так как ущерб ("проценты по долгу") появляются из-за необходимости изменения кода со временем.



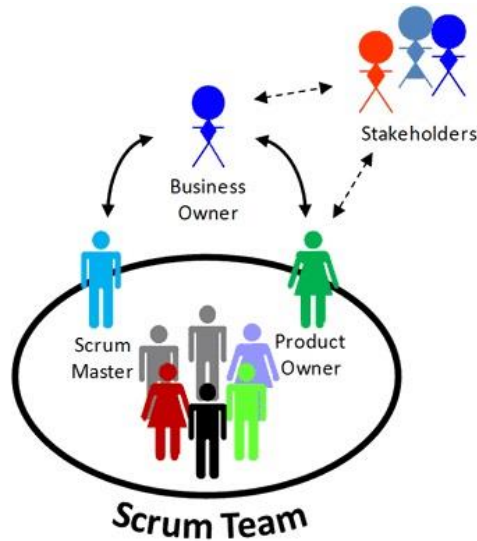
# Модный Agile: Scrum

## Задачи истории спринта (Sprint Story Tasks)

Добавляются к историям спринта. Выполнение каждой задачи оценивается в часах. Каждая задача не должна превышать 12 часов (зачастую команда настаивает, чтобы максимальная продолжительность задачи равнялась одному рабочему дню)

## Ежедневное стоячее SCRUM-совещание (Daily SCRUM)

- начинается в одно и то же время в одном месте
- все могут наблюдать, но только «свиньи» говорят
- в митинге участвуют SCRUM Master, SCRUM Product Owner и SCRUM Team
- **длится ровно 15 минут**
- все участники во время Daily SCRUM стоят (митинг в формате Daily Standup)

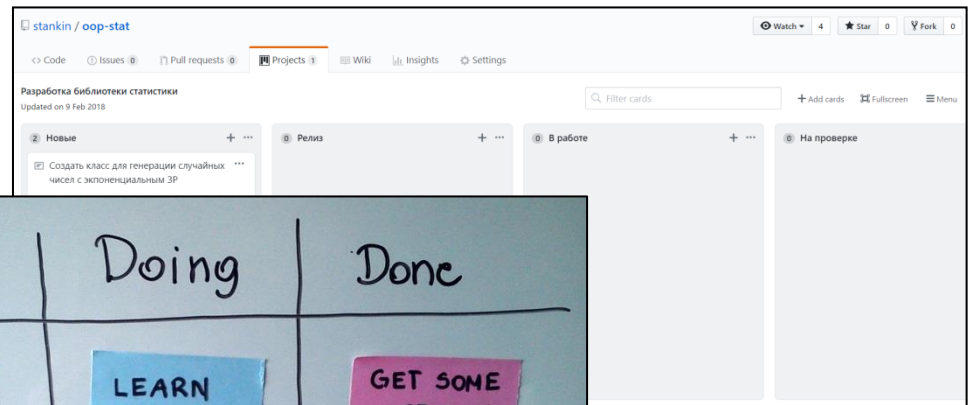
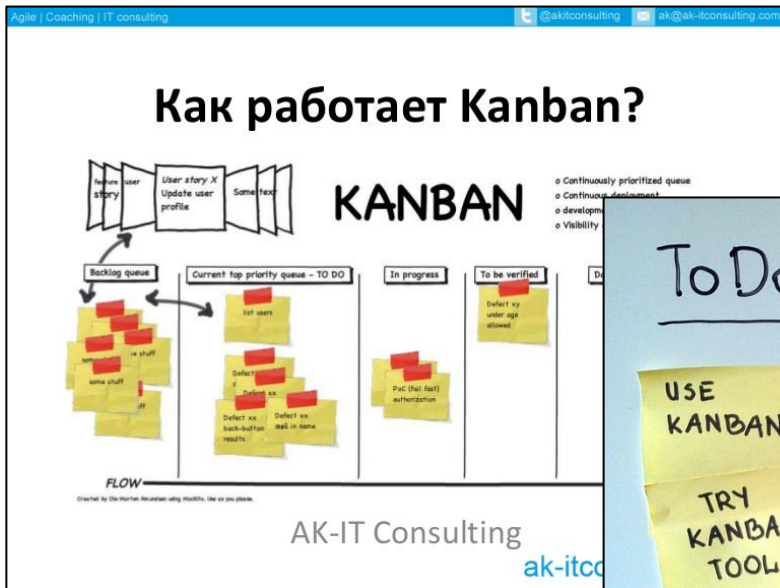


# Модный Agile: Канбан

**Канбан** (от [яп.](#) 看板 «рекламный щит, вывеска») — метод управления [разработкой](#), реализующий принцип «[точно в срок](#)» и способствующий **равномерному распределению нагрузки** между работниками.

При данном подходе весь процесс разработки прозрачен для всех членов команды. Задачи по мере поступления заносятся в отдельный список, откуда каждый разработчик может извлечь требуемую задачу.

Канбан — наглядная система разработки, показывающая, что необходимо производить, когда и сколько. Метод основан на [одноименном методе](#) в [производственной системе «Тойоты»](#) и [бережливом производстве](#).



[Канбан \(2005-2007\)](#)  
[Что такое канбан](#)

# Реинжиниринг

**Обратная разработка** (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



# 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**





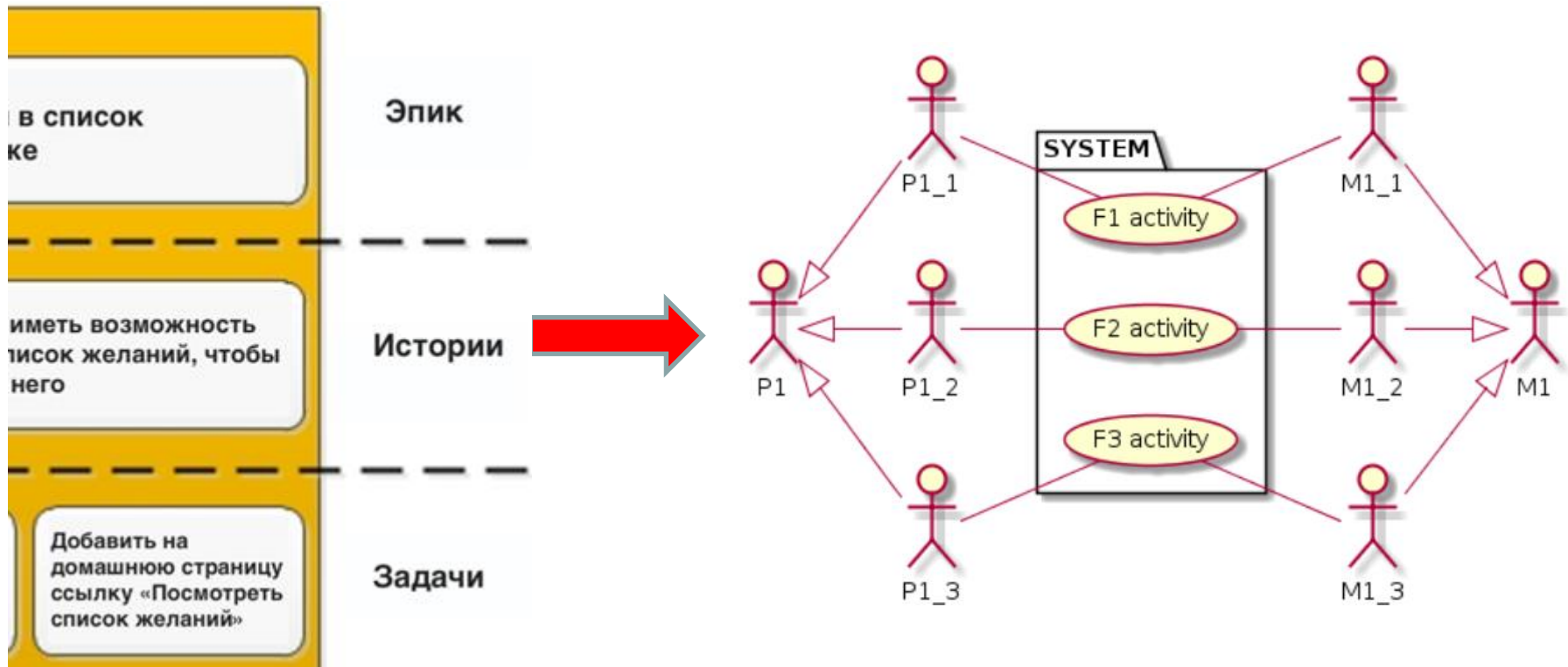
# Реинжиниринг в TFD/TDD/BDD

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

## Шаг 2. Восстановление прецедентов

Вход: истории

Выход: прецеденты

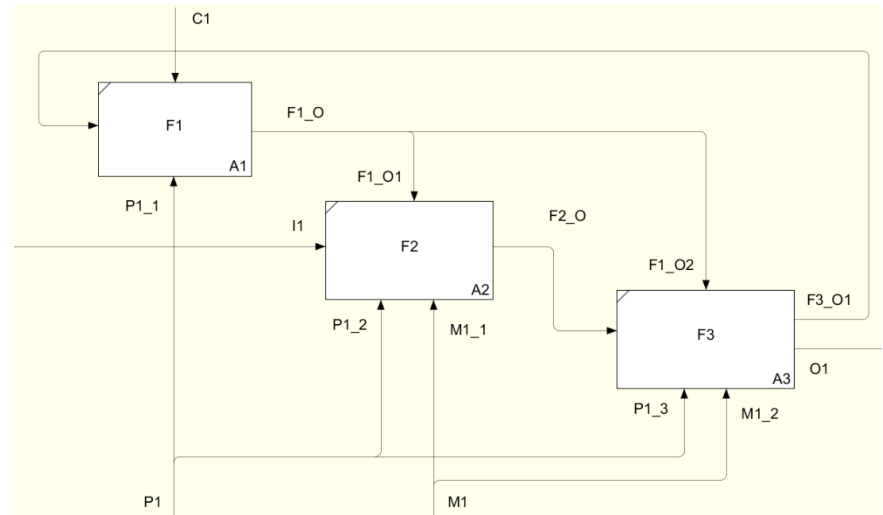
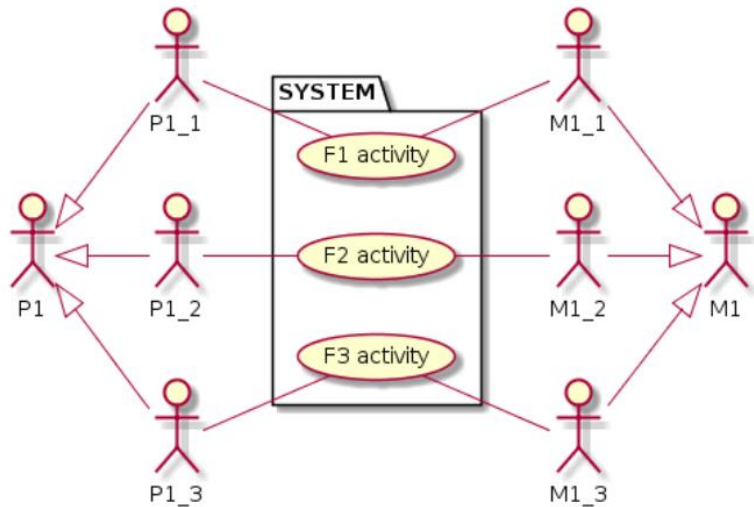


# Реинжиниринг в FDD/TFD/TDD/BDD

## Шаг 3. Восстановление функциональной структуры

Вход: прецеденты

Выход: модель процессов



**При выполнении преобразования необходимо дополнить (обогащить) 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!**



# Прецеденты: IDEF0 ↔ Use Case

## Описание решения

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

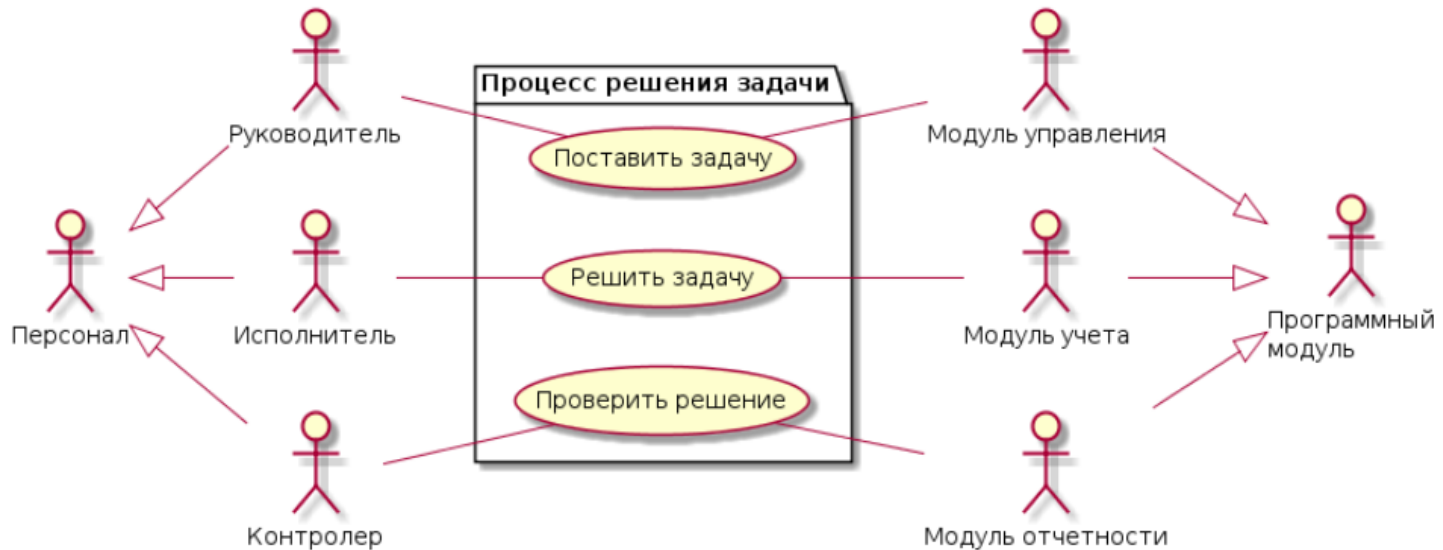
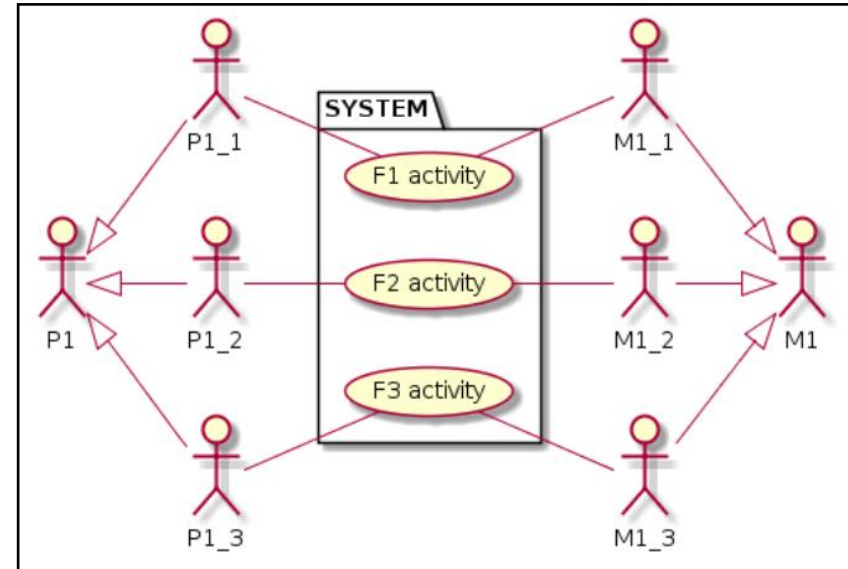
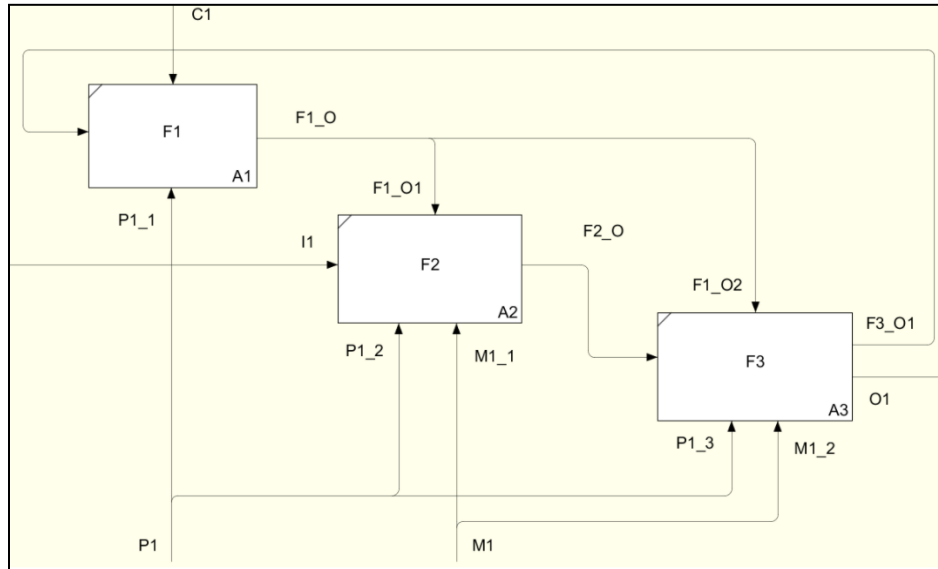
- стрелки **механизмов** преобразуются в "**actor**";
- декомпозируемые механизмы становятся родительскими "actor";
- имена **блоков** становятся именами **прецедентов**;
- все блоки дочерней диаграммы объединяются в один пакет с именем родительской.

## 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**.

# Прецеденты: IDEF0 → Use Case



# 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.

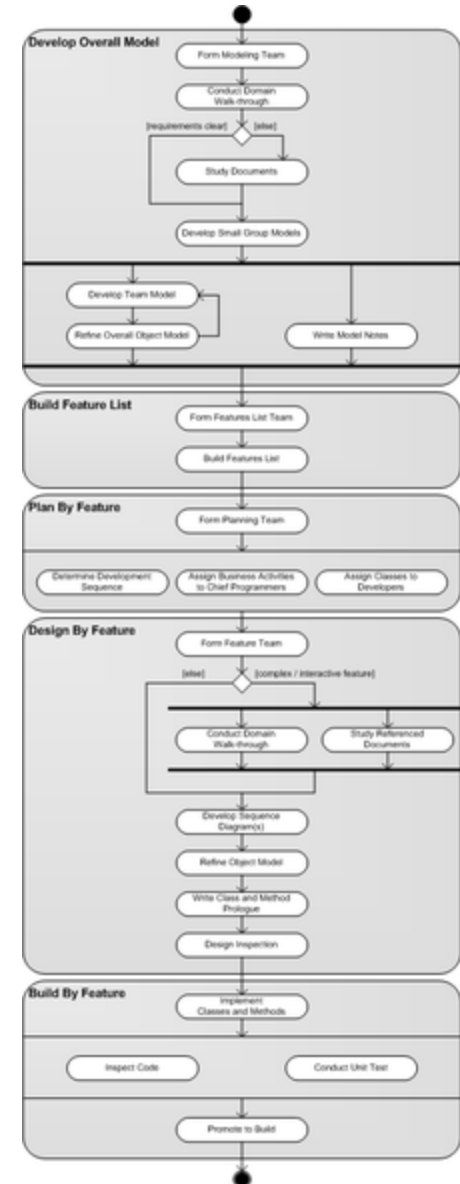
FDD включает в себя пять базовых видов деятельности:

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

[FDD \(Википедия\)](#)

[Практики Agile](#)

[Классификация прототипов](#)



# Гибкая разработка: модель FDD

## Разработка общей модели

Разработка начинается с высокоуровневого сквозного анализа широты решаемого круга задач и **контекста системы**. Далее для каждой моделируемой области делается более детальный сквозной анализ. Сквозные описания состояются в небольших группах и выносятся на дальнейшее обсуждение и экспертную оценку. Одна из предлагаемых моделей или их объединение становится моделью для конкретной области. Модели каждой области задач объединяются в общую итоговую модель, которая изменяется в ходе работы.

## Составление списка возможностей (функций)

Информация, собранная при построении общей модели, используется для составления списка функций. Это осуществляется разбиением областей ([англ. domain](#)) на подобласти (предметные области, [англ. subject areas](#)) с точки зрения функциональности. Каждая отдельная подобласть соответствует какому-либо [бизнес-процессу](#), **шаги** которого становятся **списком функций** (свойств). В данном случае функции — это маленькие части понимаемых пользователем функций, представленных в виде **«<действие> <результат> <объект>»**, например, «проверка пароля пользователя».

## План по свойствам (функциям)

После составления списка основных функций, наступает черёд составления плана разработки программного обеспечения. Владение классами распределяется среди ведущих программистов путём упорядочивания и организации **свойств** (или наборов свойств) в **классы**.

# Гибкая разработка: модель FDD

## Проектирование функций

Для каждого свойства создается проектировочный пакет. Ведущий программист выделяет небольшую группу свойств для разработки в течение двух недель. Вместе с разработчиками соответствующего класса ведущий программист составляет подробные [диаграммы последовательности](#) для каждого свойства, уточняя общую модель. Далее пишутся «болванки» классов и методов, и происходит критическое рассмотрение дизайна.

## Реализация функции

После успешного рассмотрения дизайна данная видимая клиенту функциональность реализуется до состояния готовности. Для каждого класса пишется программный код. После [модульного тестирования](#) каждого блока и проверки кода завершенная функция включается в основной проект ([англ.](#) *build*).

FDD выделяет шесть последовательных этапов для каждой функции (свойства). Первые три полностью завершаются в процессе проектирования, последние три — в процессе реализации.

Анализ области	Дизайн	Проверка дизайна	Код	Проверка кода	Включение в сборку
1 %	40 %	3 %	45 %	10 %	1 %

# Гибкая разработка: модели TDD и 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)

# Гибкая разработка: модель BDD

**BDD** (сокр. от [англ. Behavior-driven development](#), дословно «разработка через поведение») — это методология разработки программного обеспечения, являющаяся ответвлением от [методологии разработки через тестирование](#) (TDD).

Основной идеей данной методологии является совмещение в процессе разработки чисто технических интересов и интересов бизнеса, позволяя тем самым управляющему персоналу и программистам говорить на одном языке.

Для общения между этими группами персонала используется [предметно-ориентированный язык](#), основу которого представляют конструкции из естественного языка, понятные неспециалисту, обычно выражающие **поведение программного продукта** и **ожидаемые результаты**.

В **BDD** сначала пишут **спецификацию**, а потом **реализацию**. В конце у нас есть и то, и другое.

Спецификацию можно использовать тремя способами:

- Как **Тесты** – они гарантируют, что функция работает правильно.
- Как **Документацию** – заголовки блоков describe и it описывают поведение функции.
- Как **Примеры** – тесты, по сути, являются готовыми примерами использования функции.

Имея спецификацию, мы можем улучшить, изменить и даже переписать функцию с нуля, и при этом мы будем уверены, что она продолжает работать правильно.

# Гибкая разработка: модульные тесты

Автоматическое тестирование с использованием [фреймворка Mocha](#)

Автоматическое тестирование означает, что тесты пишутся отдельно, в дополнение к коду. Они по-разному запускают наши функции и сравнивают результат с ожидаемым.

```
1 describe("pow", function() {
2
3   it("возводит в степень n", function() {
4     assert.equal(pow(2, 3), 8);
5   });
6
7 });
```

[еще: QUnit](#)

```
1 describe("pow", function() {
2
3   function makeTest(x) {
4     let expected = x * x * x;
5     it(`${x} в степени 3 будет ${expected}`, function() {
6       assert.equal(pow(x, 3), expected);
7     });
8   }
9
10  for (let x = 1; x <= 5; x++) {
11    makeTest(x);
12  }
13
14 });
```

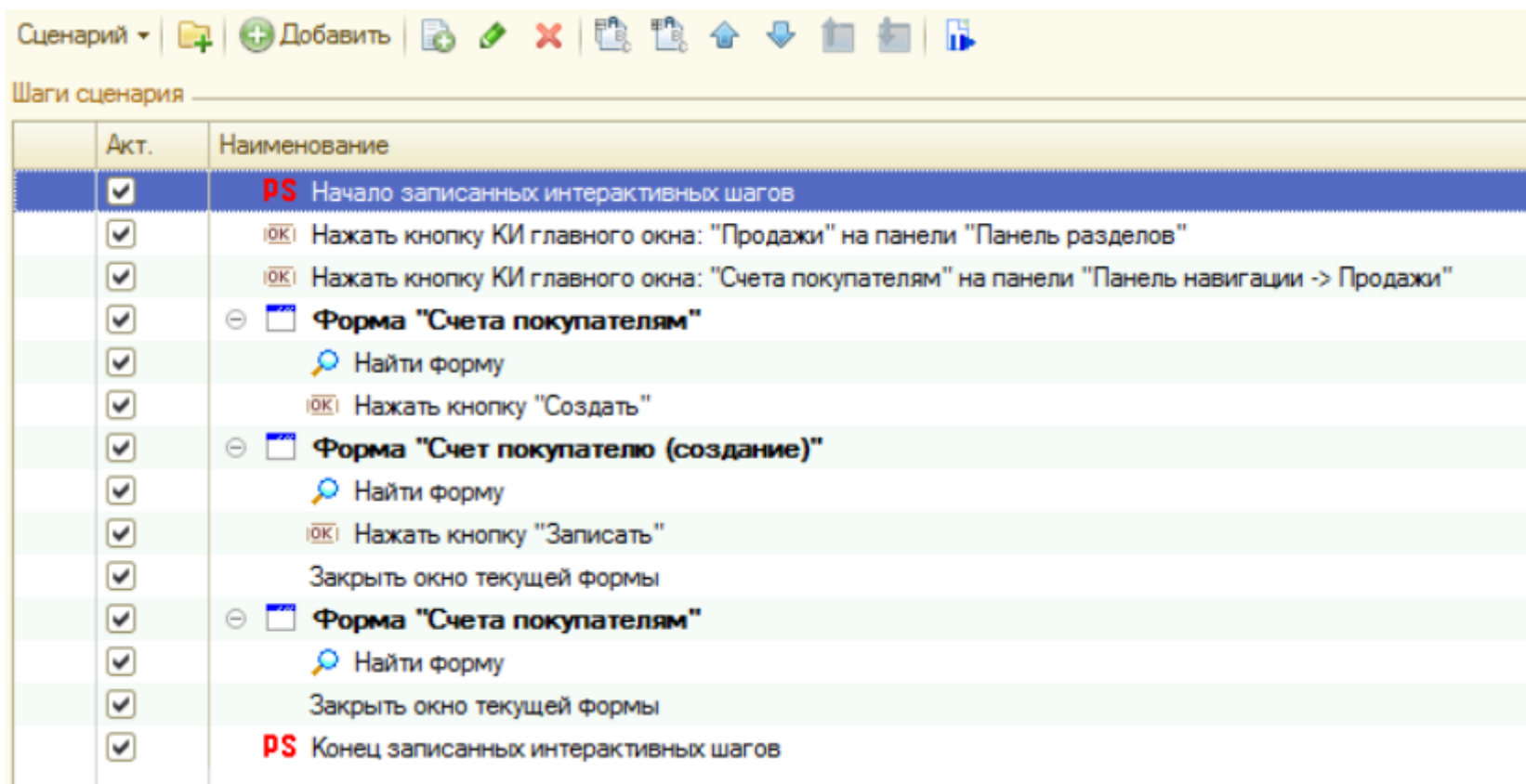


# Гибкая разработка: сценарные тесты

## Сценарное тестирование в 1С

Сценарий теста описывает порядок **действий пользователя** с **данными** в программе. Цель такого тестирования — проверить правильно ли работает программа при вводе в нее пользователем разных хозяйственных операций.

Инструмент можно использовать и для **функционального тестирования**, то есть проверки отдельных функций программы.



Сценарий

Добавить

Шаги сценария

Акт.	Наименование
<input checked="" type="checkbox"/>	<b>PS</b> Начало записанных интерактивных шагов
<input checked="" type="checkbox"/>	<b>ЮК</b> Нажать кнопку КИ главного окна: "Продажи" на панели "Панель разделов"
<input checked="" type="checkbox"/>	<b>ЮК</b> Нажать кнопку КИ главного окна: "Счета покупателям" на панели "Панель навигации -> Продажи"
<input checked="" type="checkbox"/>	<b>Форма "Счета покупателям"</b>
<input checked="" type="checkbox"/>	Найти форму
<input checked="" type="checkbox"/>	<b>ЮК</b> Нажать кнопку "Создать"
<input checked="" type="checkbox"/>	<b>Форма "Счет покупателю (создание)"</b>
<input checked="" type="checkbox"/>	Найти форму
<input checked="" type="checkbox"/>	<b>ЮК</b> Нажать кнопку "Записать"
<input checked="" type="checkbox"/>	Закрыть окно текущей формы
<input checked="" type="checkbox"/>	<b>Форма "Счета покупателям"</b>
<input checked="" type="checkbox"/>	Найти форму
<input checked="" type="checkbox"/>	Закрыть окно текущей формы
<input checked="" type="checkbox"/>	<b>PS</b> Конец записанных интерактивных шагов

# Гибкая разработка: сценарные тесты

## Стратегия автоматизации тестирования для Agile-проектов

Автоматизированное тестирование должно быть не изолированной задачей, а непрерывным процессом, неотъемлемо вписанным в жизненный цикл ПО.

### Регрессионное тестирование

Автоматические регрессионные тесты — основа стратегии автоматизации тестирования.

### Автоматическая интеграция / API-тесты или сервис-тесты

При необходимости тестирования взаимодействия с внешними сервисами, в случае, если внешние сервисы не доступны либо не могут гарантировать предоставление данных, отвечающих условиям тестирования, можно использовать эмуляторы внешних сервисов, например [WireMock](#). API-тесты и/или сервис-тесты могут запускаться на компьютере разработчика или быть частью сборки, но если они начинают занимать длительное время, лучше запускать их в среде непрерывной интеграции. Для сервис-тестов можно использовать такие инструменты, как [SoapUI](#).

### Тесты приложения

Для проведения таких тестов в браузере можно использовать [Selenium](#) [WebDriver](#). Этот инструмент является наиболее популярным для проведения автоматизированного тестирования в браузерах и предоставляет богатые возможности API для проведения сложных проверок.

# Гибкая разработка: модель MDD (MDE)

Разработка, управляемая моделями (*model-driven development*, **MDD**, *Model-driven engineering*, **MDE**) — это стиль разработки программного обеспечения, когда **модели** становятся **основными артефактами** разработки, из которых **генерируется код** и другие артефакты.

**Модель** — это **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. Модель определяет **нотацию** и **метамодель**. Нотация представляет собой совокупность графических элементов, которые применяются в модели и могут быть интерпретированы людьми. Метамоделю описывает используемые в модели понятия и фиксирует информацию в виде метаданных, которые могут быть обработаны инструментальными средствами.

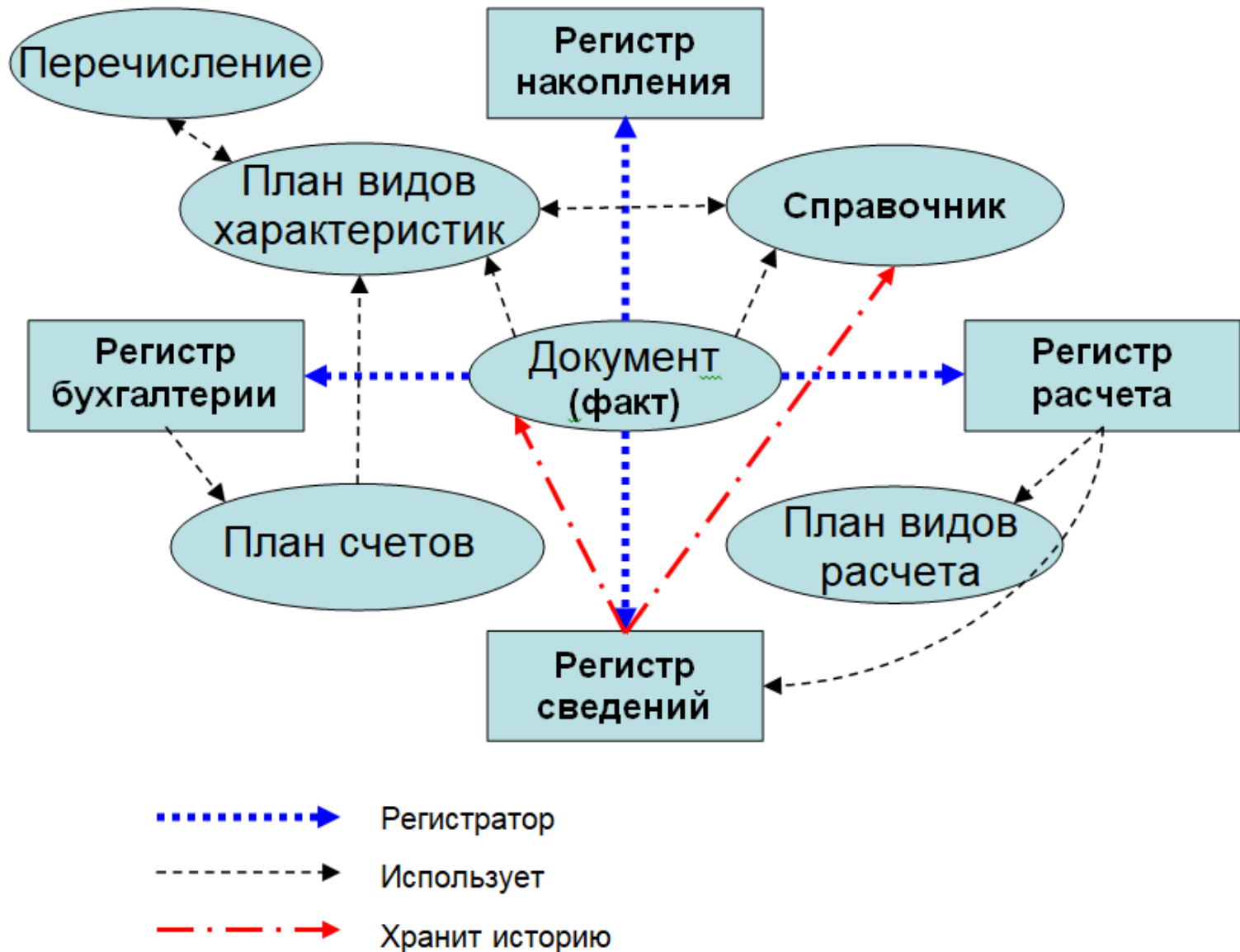
Наиболее известными современными MDE-инициативами являются:

1. разработка Object Management Group (OMG) под названием model-driven architecture (MDA)
2. экосистема Eclipse для инструментов моделирования и программирования (Eclipse Modeling Framework)

[MDD/MDE \(Википедия\)](#)

[1С приложение за 15 минут](#)

# Полная реализация MDD: метамодель



# Частичная реализация принципов MDD: MVC

## Паттерны MVC

