Какое качество нужно вашему проекту и как организовать разделение ответственности за него

Максим Цепков

IT-архитектор и бизнес-аналитик

Навигатор и эксперт по миру Agile, бирюзовых организаций и Спиральной динамике

http://mtsepkov.org

Quality Conf 2019

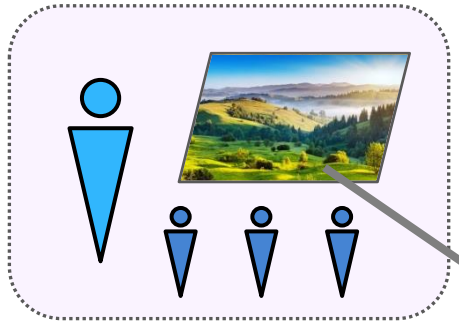
Конференция про качественную разработку IT-продуктов

Качество: какой смысл вложен в понятие?

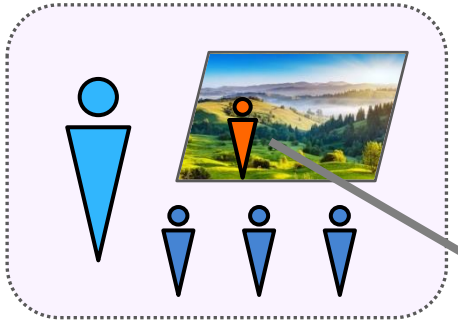
- Многие полагают, что понятия однозначны и фразу «Проект надо сделать качественно» все **должны** понимать одинаково, а для ответственности тоже есть правильное распределение
- Реально мир меняется, и **смысл** понятий **меняется** вместе с ним
- Но в головах многих смыслы сохраняются такими, как были в момент обучения или первого узнавания
- Мы поговорим о том, **какое** бывает качество проектов, и как организовывать разделение ответственности за него

Цель – представить спектр представлений, на основе которых вы сможете сделать то, что нужно вашему проекту

Доклад – карта местности, надо увидеть себя



Рассказ дает карту местности и возможные варианты движения



Необходимо увидеть на этой карте будущий путь своего проекта и себя на этом пути

Путешествие не является необходимым – это зависит от ситуации в проекте. Но пока ты не представишь себя идущим, материалы доклада будут мертвой теорией

По мотивам схемы из моего доклада
«Как строить свой профессиональный путь - схемы самоопределения»

План доклада

- Какое качество нужно разным проектам?
  - Как менялись представления о «качественном проекте»
  - Какое качество нужно для разной ИТ-разработки
- Качество: кто и за что отвечает?
  - Между кем разделяется ответственность
  - Как работать с границами ответственности
    и какова ответственность тестировщика
- А как все-таки меняются смыслы?
  - Team: как нам понимать друг друга и эффективно сотрудничать

Как менялись представления о «качественном проекте»

Энтони Лаудер «Культуры программных проектов» (2008)

Оригинал, перевод (pdf), рецензия Стаса Фомина

- История ИТ-отрасли делится на этапы
- Для каждого этапа характерен свой подход к ведению ИТ-проектов: представления об успехе, критерии качества и организации работ
- Выделяются четыре культуры:
  - Научная
  - Заводская
  - Дизайнерская
  - Сервисная
- Каждая культура породила свои учебники, они основаны на представлениях того времени и согласованы между собой

Смена культур ИТ-проектов

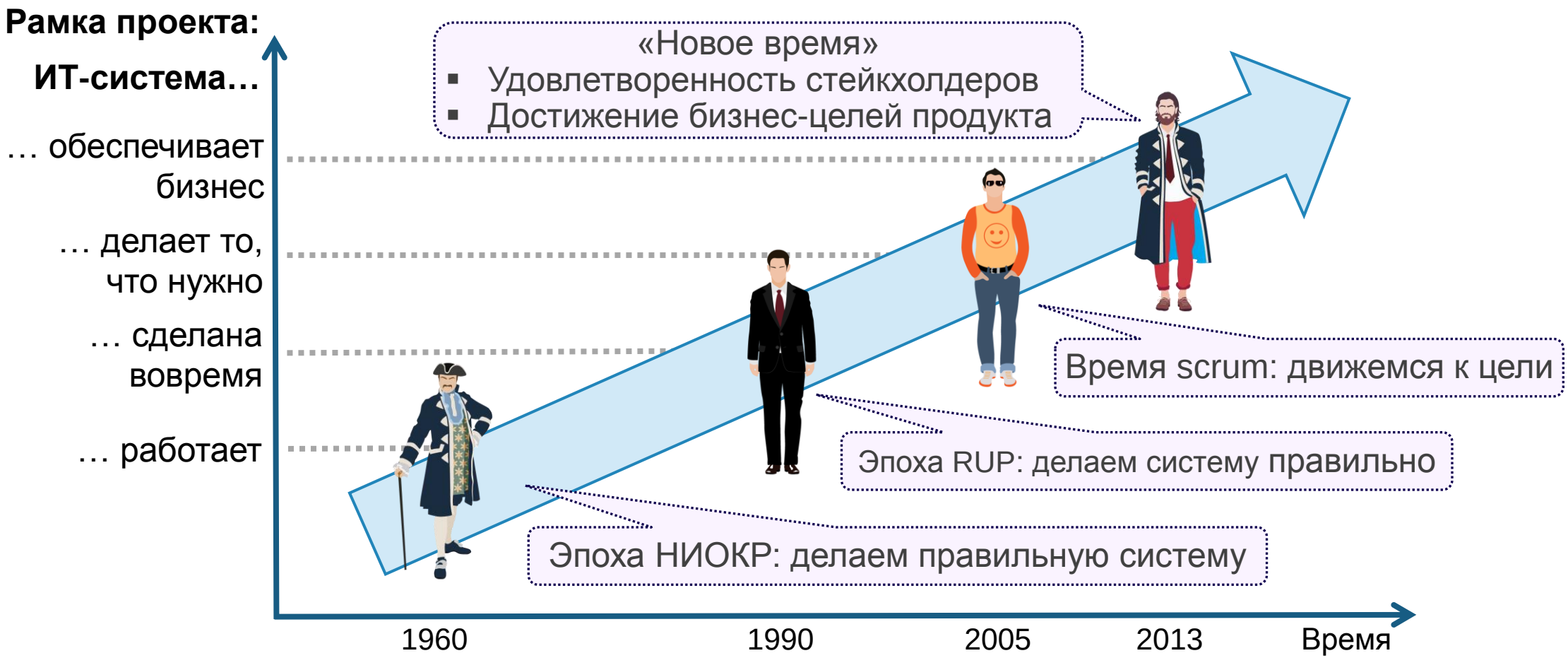
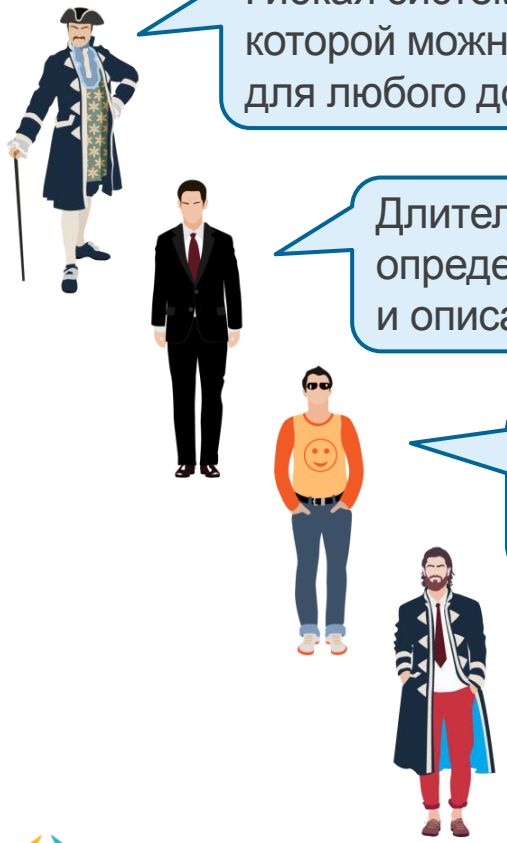


Схема из моего доклада «Мыслить проектно: история и современность» на SECR-2018, впервые рассказана на AgileDays-2015

Пример: шаблоны документов в разных культурах

Задача: сделать механизм шаблонов для ввода типовых документов



Гибкая система шаблонов, с помощью которой можно настроить шаблон для любого документа

Настройка шаблонов окажется сложной даже для опытных сотрудников. Про удобный поиск и права доступа забудут

Длительная проработка задачи, определение критериев эффективности и описание «идеальной системы»

Шаблоны окажутся жесткими и пригодными только для узкого класса ситуаций, а расширить функционал будет сложно

Быстрая серия прототипов и решений с последовательным усложнением шаблонов

Первые версии окажутся совсем не адекватными, на демо нужны конечные пользователи и фасилитация. Со сложными и редкими кейсами могут быть проблемы

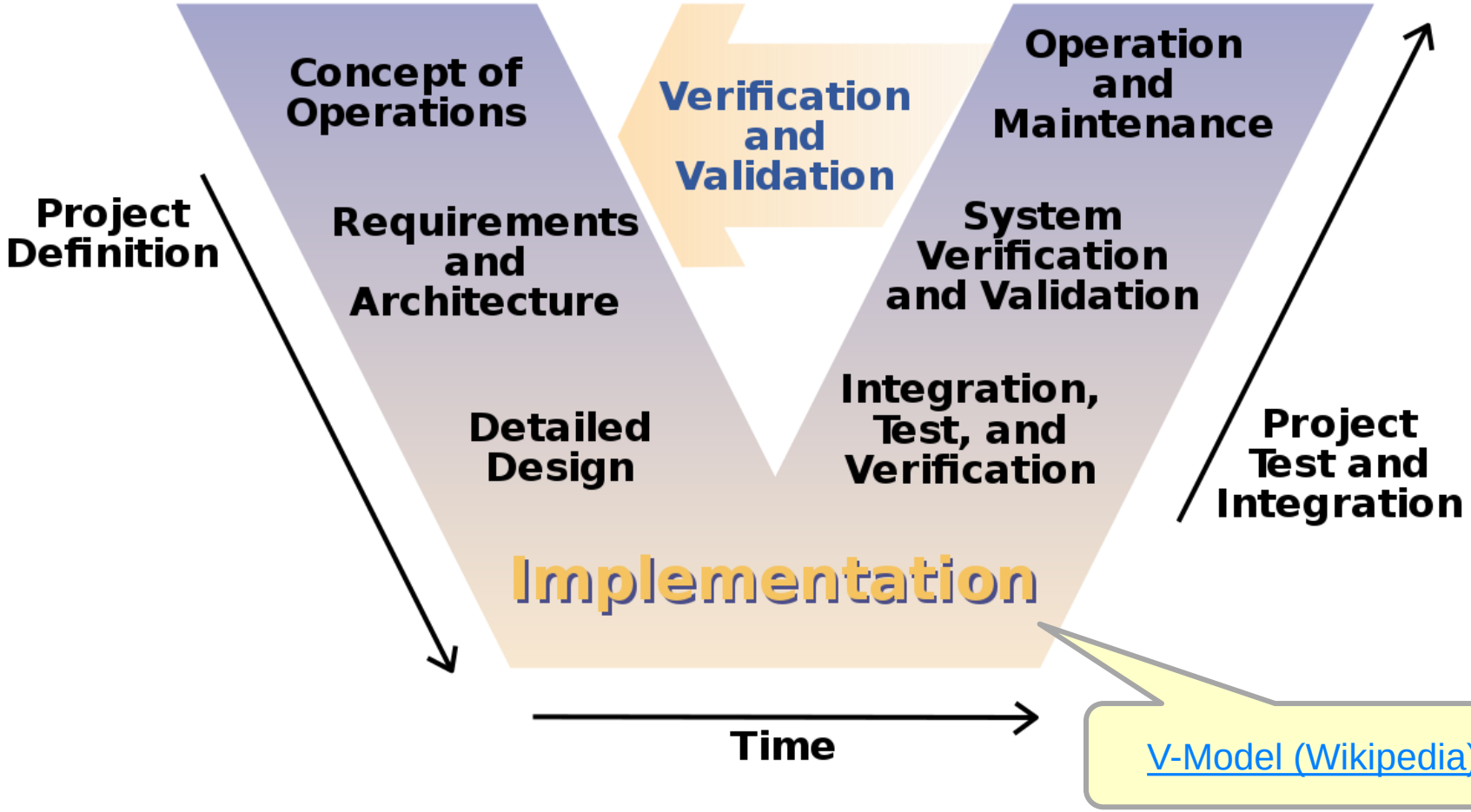
Создание целевых групп для первых версий и работа именно с их кейсами. Может появиться несколько альтернативных механизмов, например, образцы документов вместо шаблонов

Эпоха НИОКР: когда компьютеры были большими

- Создавались большие и сложные системы
- Требования к системе редко менялись
- Проекты делал квалифицированный персонал
- Упор был на качество ИТ-системы

Цель проекта – создать **совершенную ИТ-систему** в **одном** экземпляре

Представление о проекте – V-модель



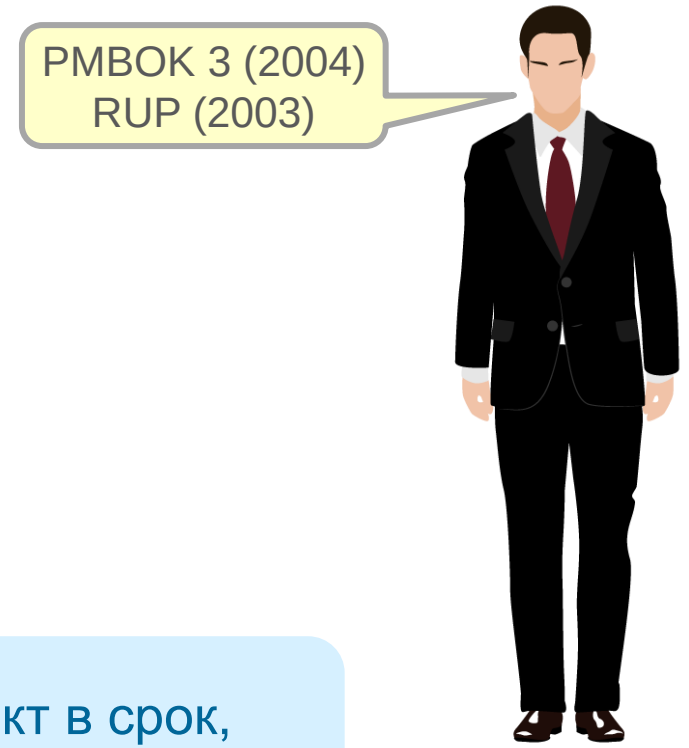
V-Model (Wikipedia)

Инженерная культура сейчас

- Построение ядра системы и совершенных фреймворков
- Фокус на стройности и архитектурном совершенстве
- Стремление поддержать все сложные кейсы
- Неприятие особых случаев, исключений
  и временных решений
- Слабая забота о тех, кто не будет работать
  со сложными решениями, даже когда они в большинстве

Эпоха RUP: массовая потребность в проектах потребовала много разработчиков

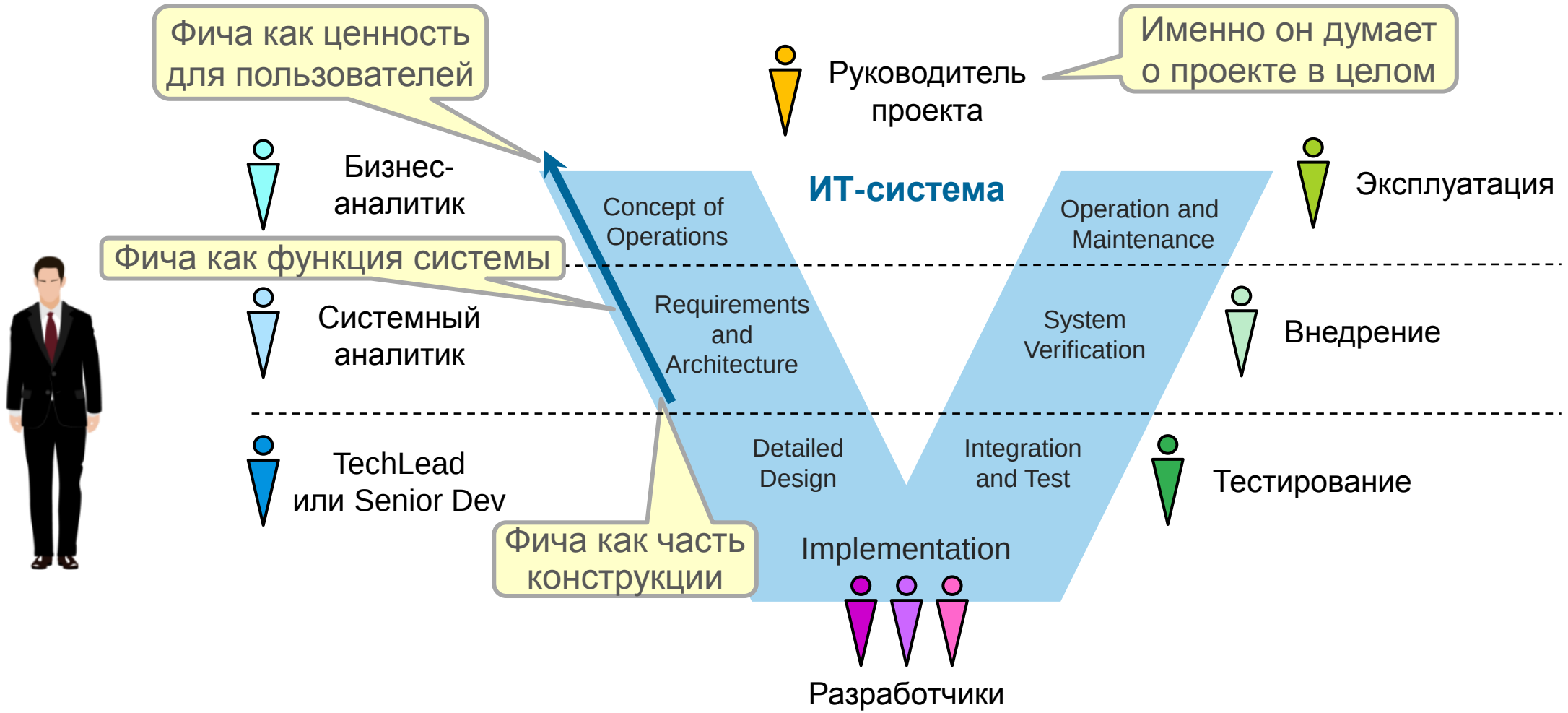
- Применим к ИТ-разработке принципы промышленного производства
- Разделим задачу на этапы: проектирование, разработка, внедрение
- Наладим процессы и разделим зоны ответственности



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

Получалось не очень

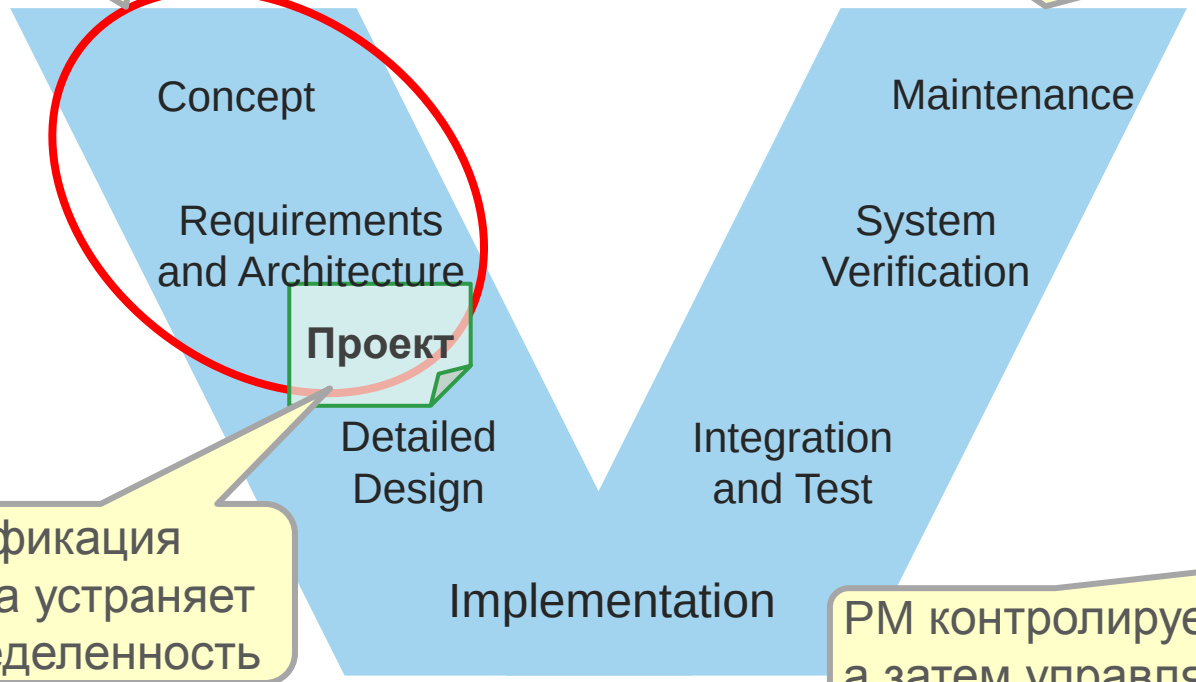
Специализации в проекте



Неопределенность – в проектировании

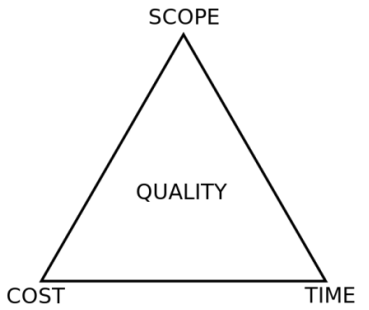
Проектирование – **НИОКР**, результат **не гарантирован**

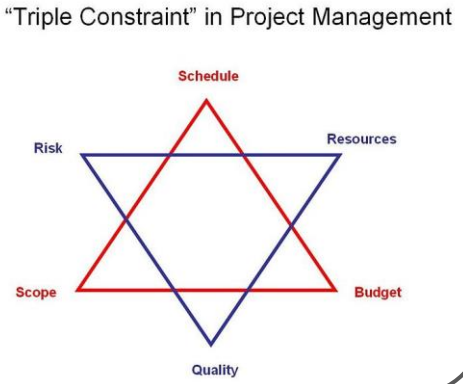
Исполнение – **производство**, результат обеспечивается регламентами, включая контроль рисков



Спецификация проекта устраняет неопределенность

Проектный треугольник



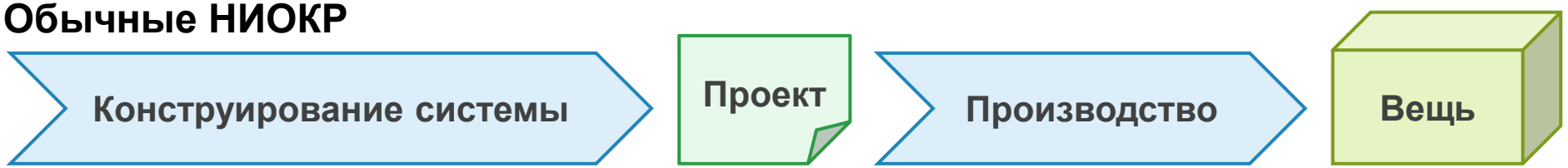


PM контролирует результат проектирования, а затем управляет исполнением в проектном треугольнике

Разработка кода – часть проектирования

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

Обычные НИОКР



ИТ-разработка

А еще для уверенного успеха НИОКР в проектировании нельзя использовать технологии с уровнем зрелости (TRL) ниже 8. Если бы так действовали в IT, смартфоны выходили бы без софта ☺

Почему не работают регламенты?

- Успех проекта определяют люди – это было осознано еще в 1980-х и обосновано Томом ДеМарко в книге «Человеческий фактор» (1987)
  - Критика регулярного управления (книга Ф. Брукса «Мифический человеко-месяц», 1975)
  - Разработка софта – НИОКР, а не производство (статья Jack W. Reeves, 1992)
  - Решения рядового разработчика влияют на успех всего проекта
  - Производительность разработчика в разных условиях отличается на порядок
- Этому не поверили, и в конце 1990-х был поставлен эксперимент по нормированию процессов – RUP. Он окончился неудачей
  - Стоимость выросла многократно без гарантий успеха
  - Обеспечить реакцию на изменения требований не получилось

Но эксперименты продолжаются, культура живет

Вызовы, на которые не ответили в RUP

- **Стоимость**: процедура увеличивает ее кратно, не сильно повышая вероятность успеха
- **Изменчивость:** потребности меняются быстрее, чем проходит цикл разработки, и нужно учесть эти изменения
- **Управленческие кадры:** где их брать, особенно руководителей групп?
- **Нормирование аналитической работы:** попробовали в PMBOK 4 – не получилось

В стандарте признано

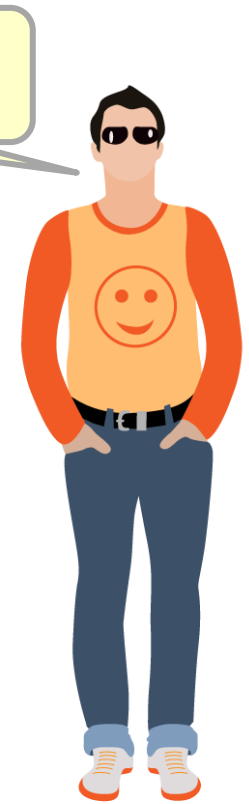
Итерации в RUP – тяжелые

Появление персоналок кратно усилило вызовы

- Возникла возможность автоматизировать бизнес компаний – но бизнес-процессы за время проекта успевают измениться
- Резкий дефицит квалифицированных кадров
- Конкуренция компаний за специалистов, а не разработчиков – за рабочие места
- Профессиональная самореализация – одна из главных ценностей разработчика, как совместить ее с коллективным результатом?

Agile и scrum: ответ на вызовы

- Планирование не работает – наблюдаем за траекторией движения проекта и приближением к цели
- Коллективное преодоление неопределенностей: все члены команды думают о движении проекта
- Концепция SMART-целей, измеримость достижения
- Требования изменяются вместе с целью



Качественный проект – это частые инкрементальные поставки **нужного** софта

User story – мышление за пользователя

Как <*роль*> я хочу <*сделать что-то*> для того, чтобы <*достичь целей*>

- Часть «для того, чтобы»
  - Позволяет разработчику занять позицию пользователя при реализации фичи
  - Говорит о бизнес-целях использования
  - Позволяет принимать решения в процессе реализации
- Эта часть появилась не сразу, а из опыта использования
- О фиксации позиции пользователя заговорили и в других форматах требований

UI-design и usability

Интерфейс должен быть функциональным и понятным

Но дешевым в изготовлении. А пользователей – научим

Дизайн – хорошо. Но важнее эффективная работа пользователя – **usability**

Enterprise

И нам это тоже важно

Public Web

Сайты должны быть привлекательны для пользователей. Нужен **UI-design**

На интерактивных сайтах usability тоже важна. Особенно в интернет-магазинах

И пользователи не должны учиться – интерфейс должен быть интуитивно понятен. Опираемся на **User eXperience**

У пользователя **много устройств**: смартфоны, планшеты, ноутбук, компьютер. Он работает одновременно во многих приложениях

Современные векторы развития

- От проектной деятельности – к непрерывному развитию продукта
  - Канбан в ИТ (2010)
  - DevOps (2012)
- От качества ИТ-системы – к удовлетворенности стейкхолдеров
  - PMBOK 5 (2013) (частично)
- От создания системы – к достижению возможностей для бизнеса и пользователя
- Каждой ИТ-разработке – свой метод
  - OMG Essence (2012)

Качественная ИТ-разработка удовлетворяет стейкхолдеров и обеспечивает возможности для бизнеса

Какое качество нужно для разной **ИТ-разработки?**

Слово «проект» исчезло не случайно.
Меняем не только смысл, но и понятие!
Вместо «проекта» – **предприНятие**, **endeavor**

Что такое успешный проект?

Мы создали **качественную систему**
в соответствии со спецификацией требований

… и при этом **уложились в сроки**
и бюджет проекта – заказчик доволен

Мы создали тот **софт, который нужен заказчику**,
опираясь на обратную связь и сотрудничая с ним

Стейкхолдеры проекта могут достигать своих бизнес-целей
в соответствии с ожиданиями от проекта

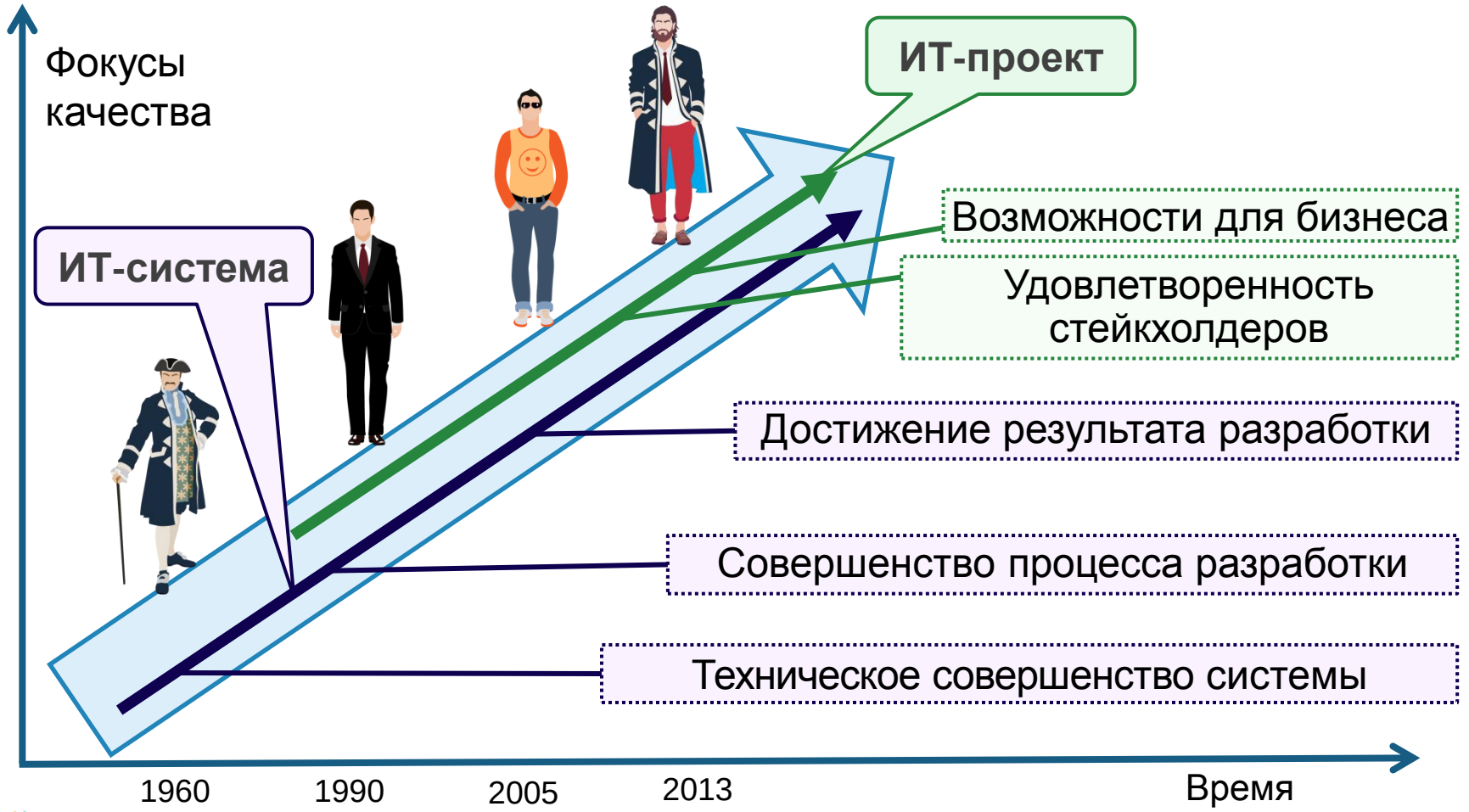
Если требования неверны и нужна другая система?

Жаль, что все это выяснилось так близко к сдаче проекта. Все сделано по требованиям – **вы должны это принять**. А потом мы готовы сделать новый проект за новые деньги

Частые демонстрации работающего софта позволяют проверить его адекватность задачам заказчика и **скорректировать движение проекта**

Если при очередной демонстрации выясняется, что софт не позволяет решить задачу бизнеса, **команда вместе со стейкхолдерами заказчика ищет решение**. Успех проекта – реализация такого решения. Деньги и сроки – предмет переговоров

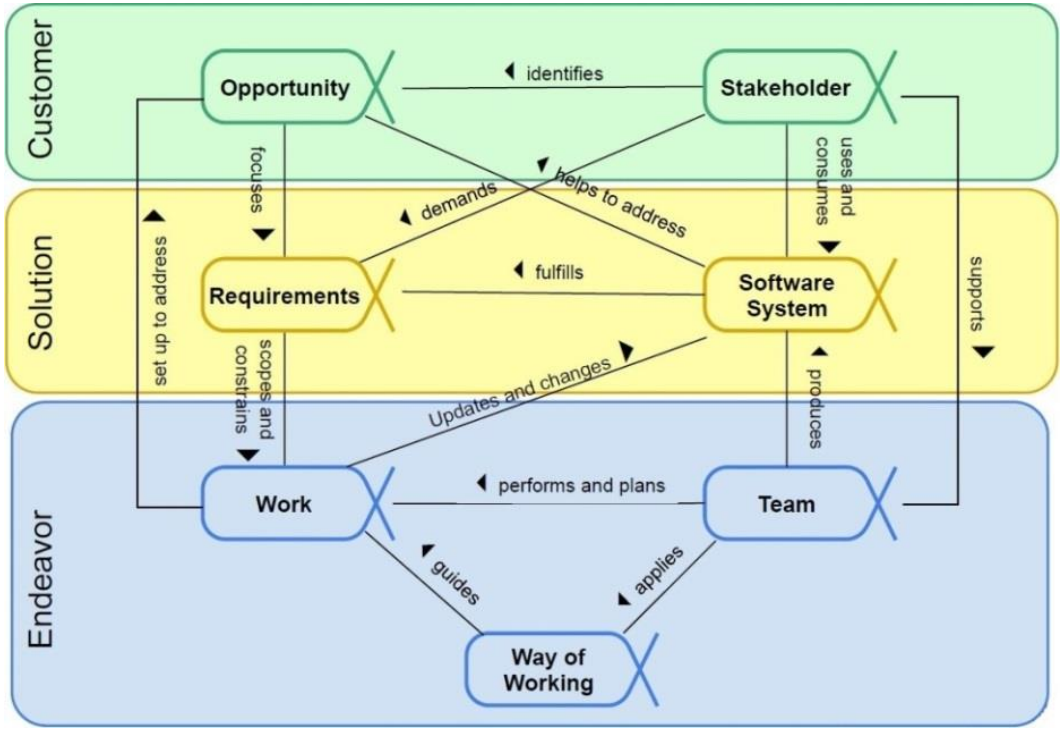
Каждый этап добавлял новые фокусы качества, а не отменял старые

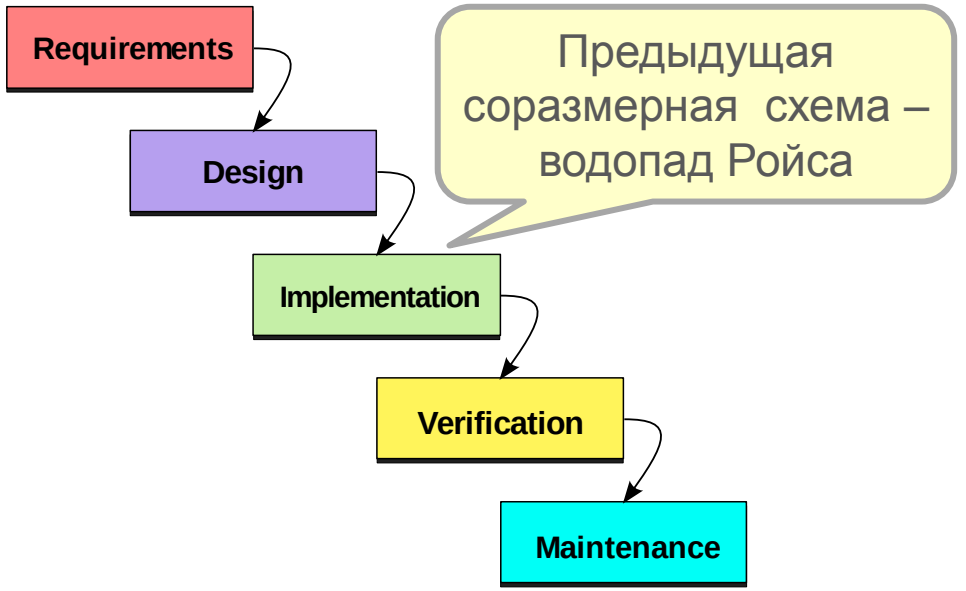


OMG Essence – пространство привязки фокусов качества

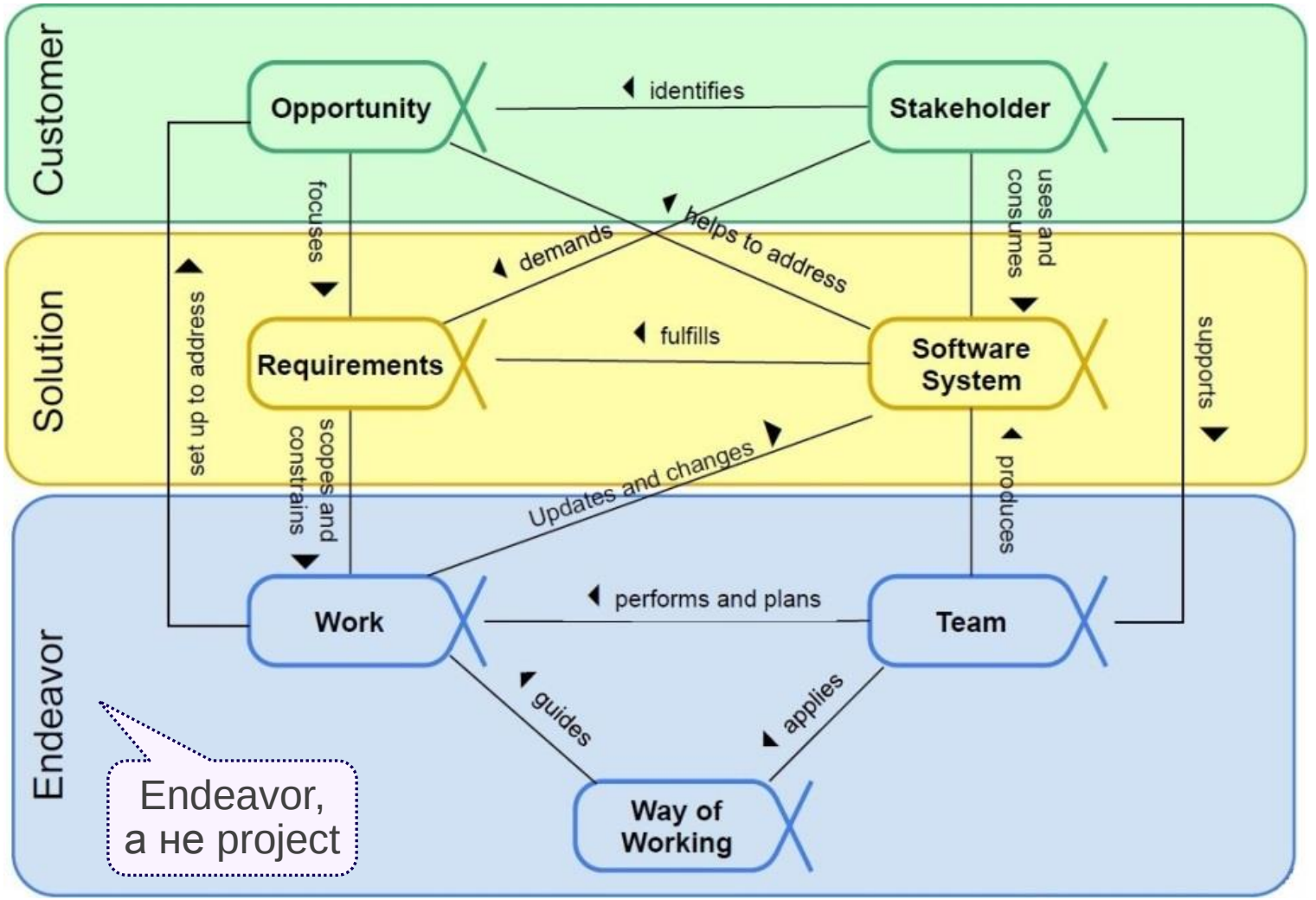
Создан группой SEMAT под руководством Ивара Якобсона

- Стандарт на сайте OMG
- Конкретные описания на сайте Ивара Якобсона
- Книга Ивара Якобсона The Essence of Software Engineering
- Курс системного мышления Анатолия Левенчука: coursera, учебник

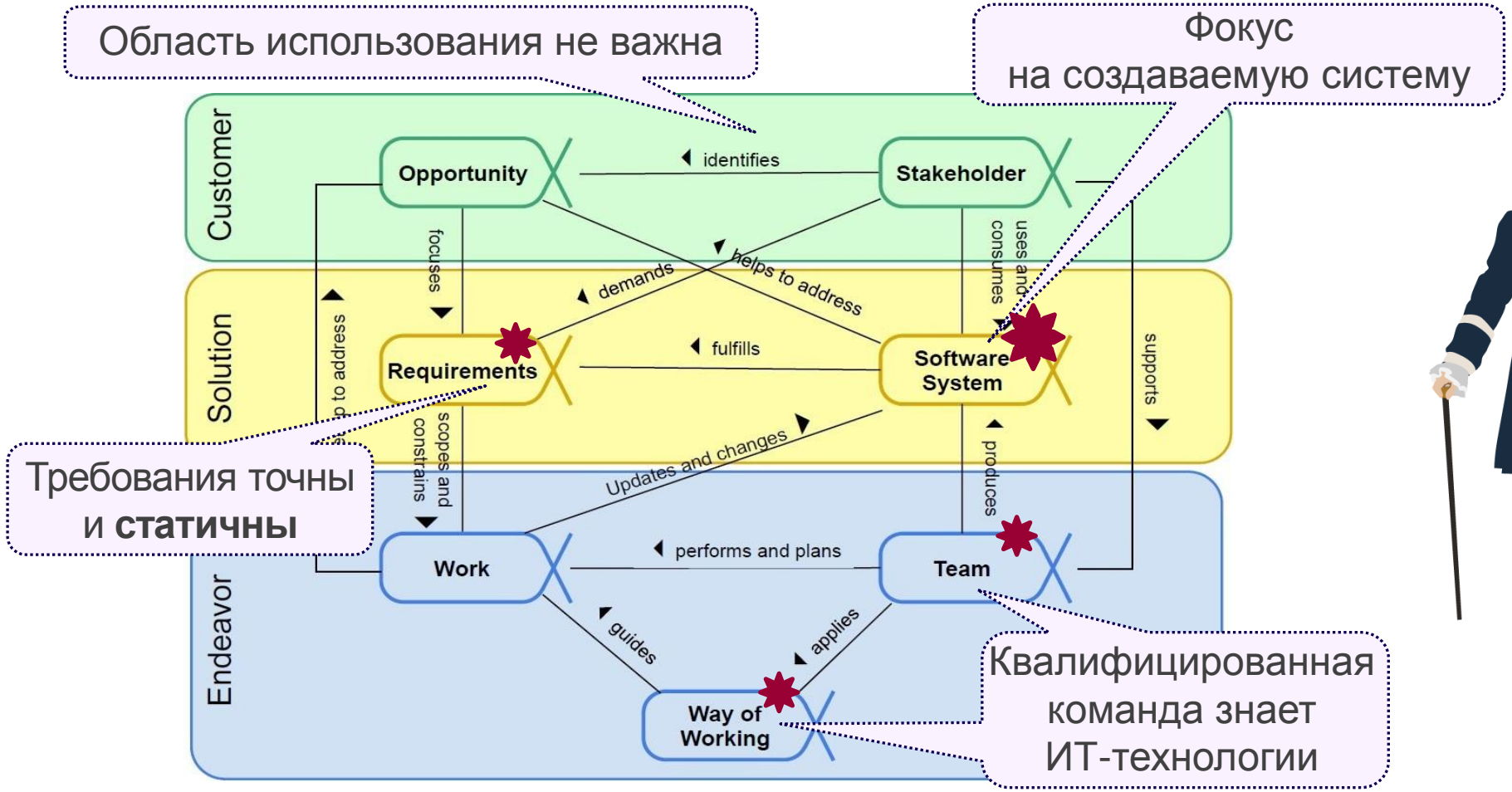




Альфы проекта – множество фокусов



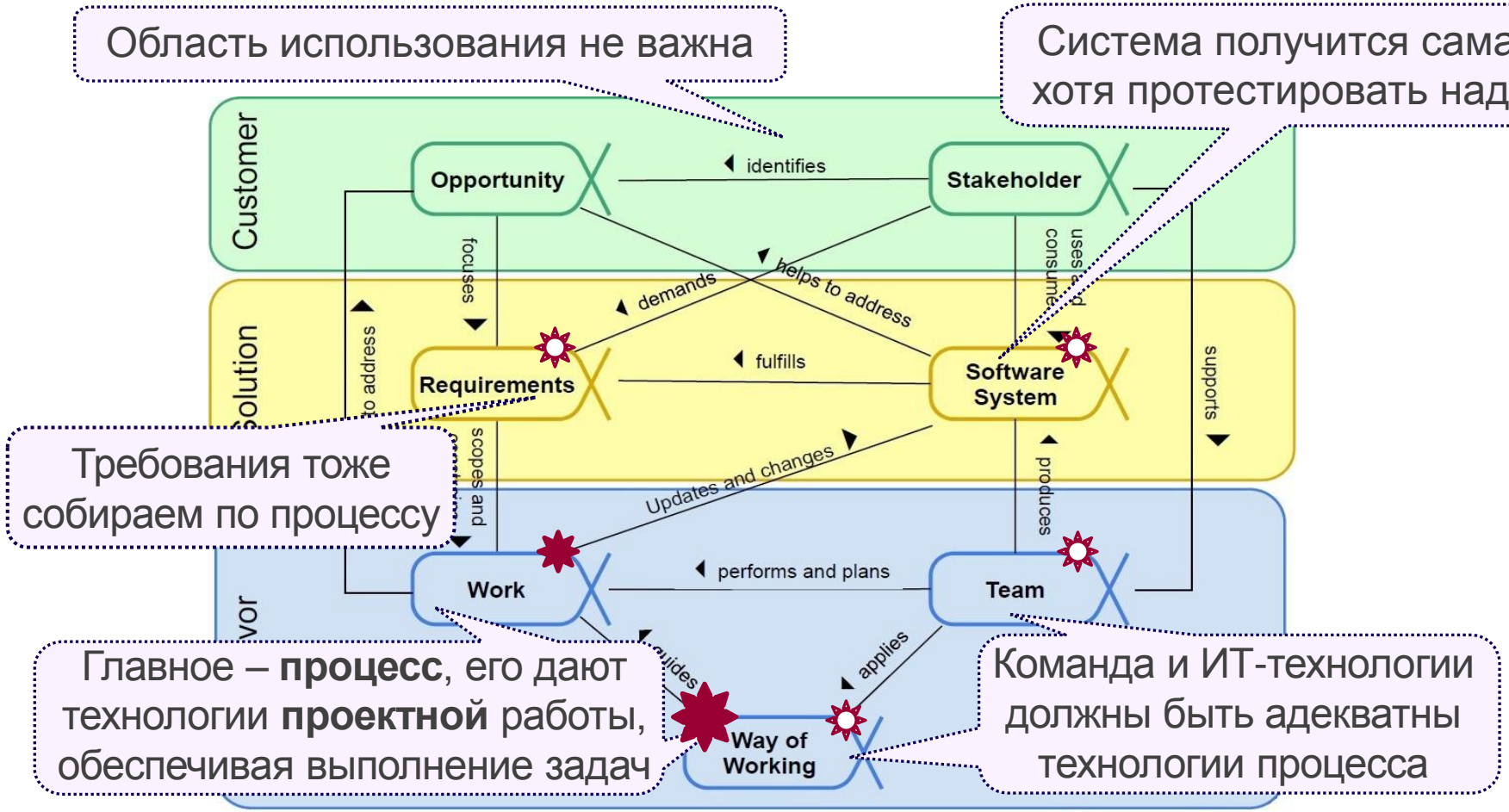
Система понятна как **черный** ящик, но **сложно** устроена – НИОКР



Система понятна как **белый** ящик – организуем **процесс** разработки

Область использования не важна

Система получится сама, хотя протестировать надо



Требования тоже собираем по процессу

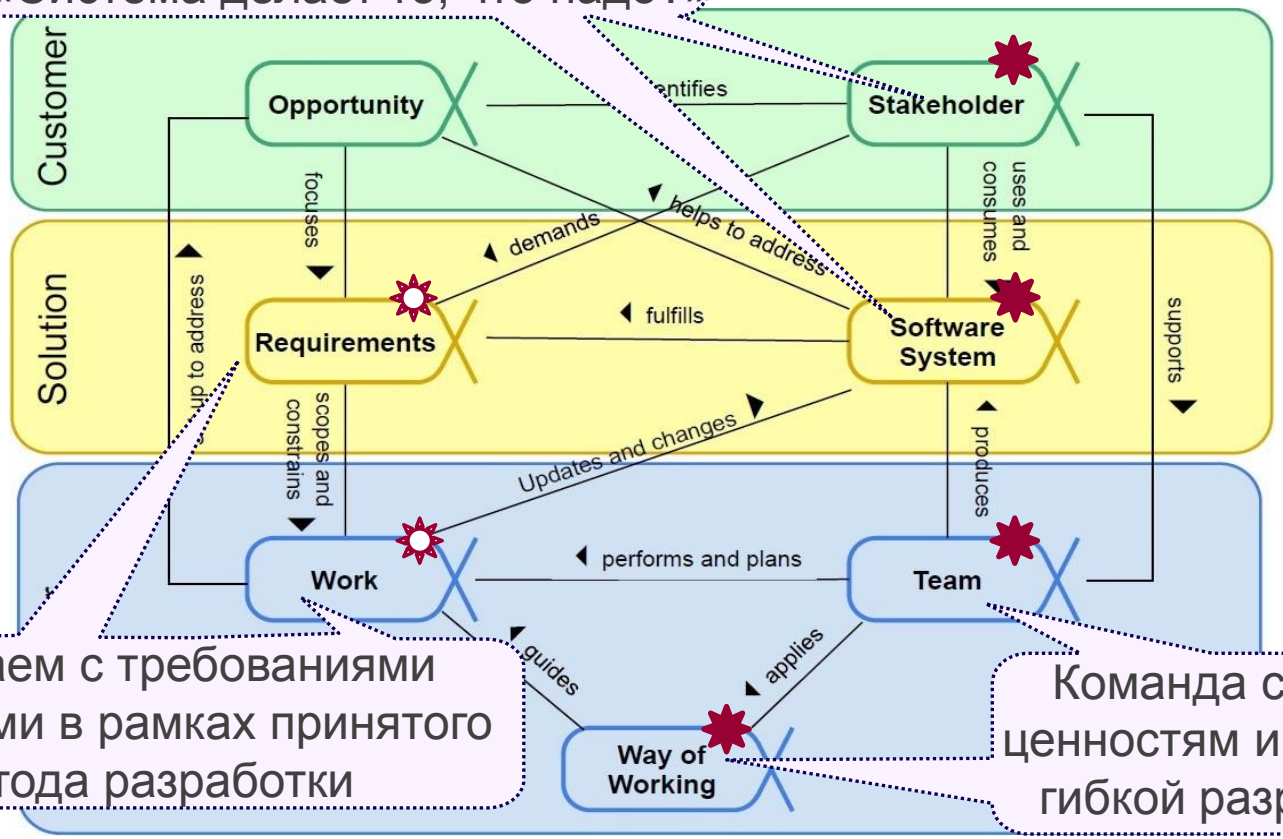
Главное – **процесс**, его дают технологии **проектной** работы, обеспечивая выполнение задач

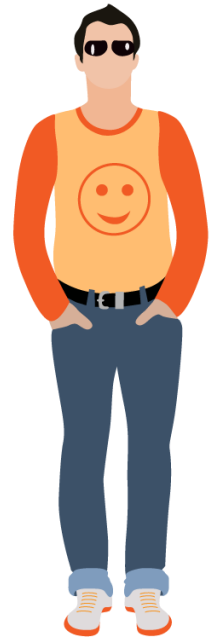
Команда и ИТ-технологии должны быть адекватны технологии процесса

Образ системы неясен – приближаемся к нему итерациями

Объективно или из-за ограниченной квалификации команды

Постоянная проверка
у стейкхолдеров:
«Система делает то, что надо?»





Работаем с требованиями
и задачами в рамках принятого
метода разработки

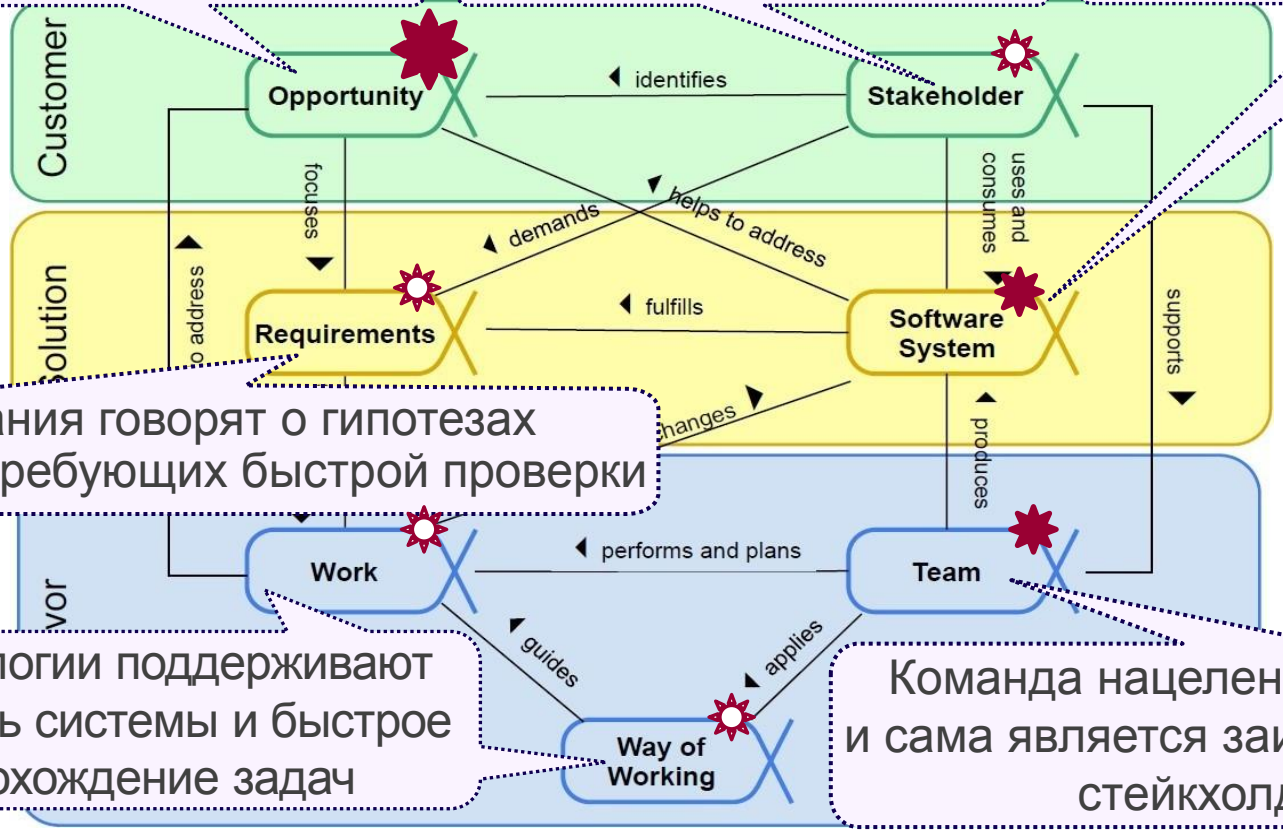
Команда следует
ценностям и методам
гибкой разработки

Система обеспечивает ожидаемые возможности развития бизнеса

Возможности в мире – драйвер изменений системы

Стейкхолдеры мониторят изменения мира

Система быстро изменяется вслед за миром



Требования говорят о гипотезах изменений, требующих быстрой проверки

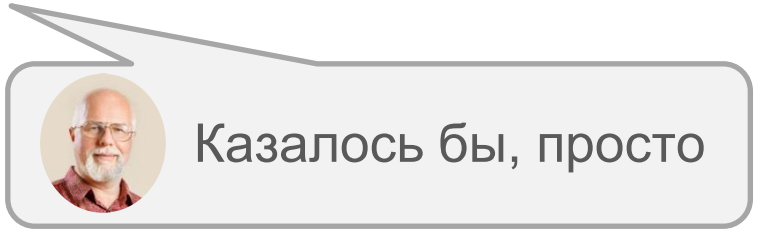
Технологии поддерживают гибкость системы и быстрое прохождение задач

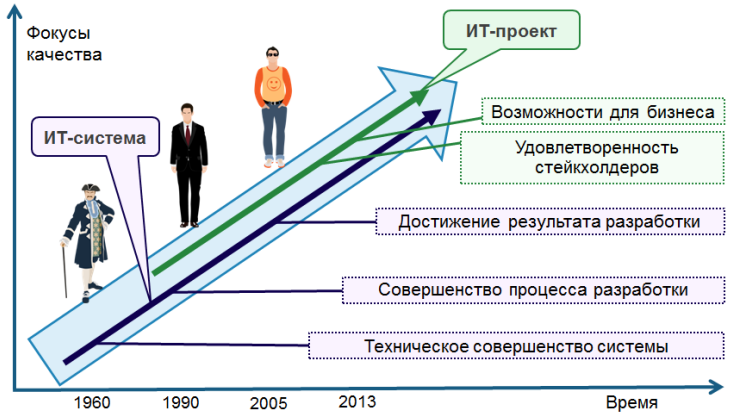
Команда нацелена на результат и сама является заинтересованным стейкхолдером

Между кем разделяется ответственность

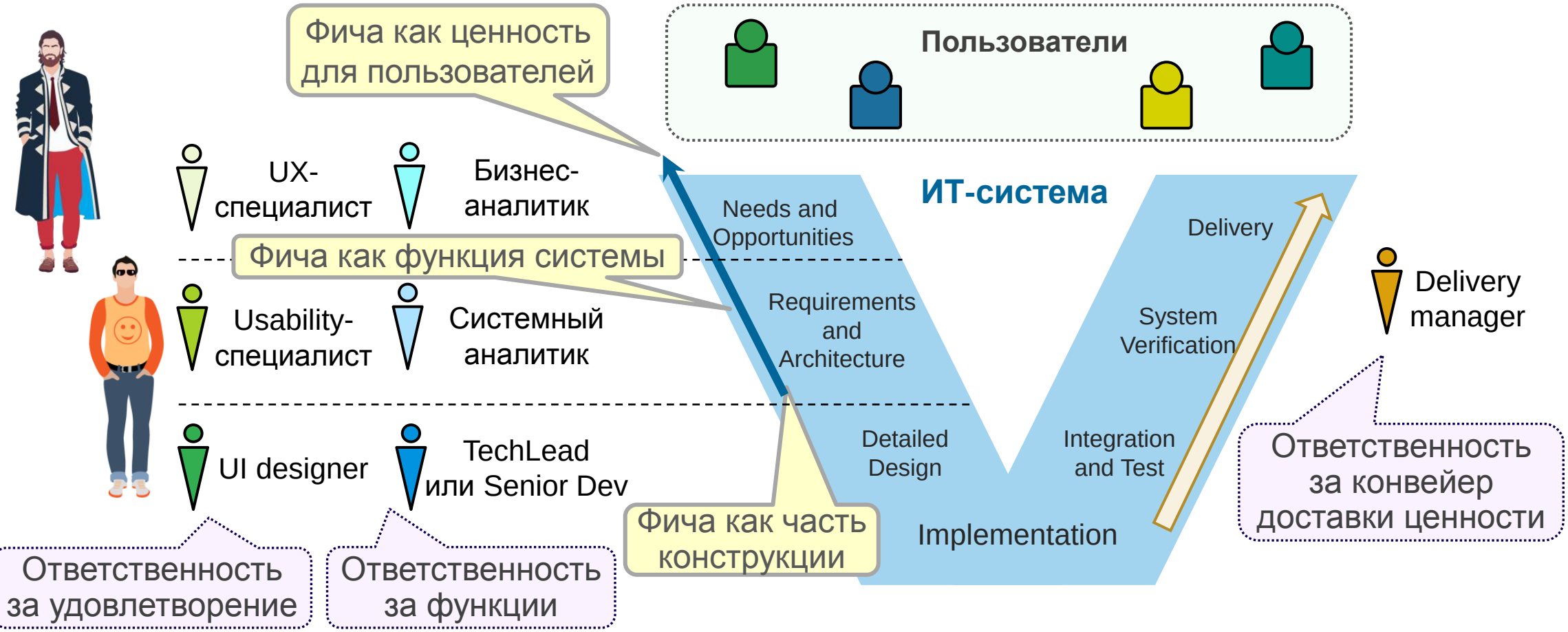
QE и QA – послание от RUP

- Quality Engineer отвечает за **качество ИТ-системы** и проверяет ее тестированием
- Quality Assurance отвечает за **качество процесса**, которое в замысле должно привести к качеству системы
- Это лишь **два фокуса** ответственности из многих
- Остальные могут быть возложены на тех же людей или на отдельных лиц

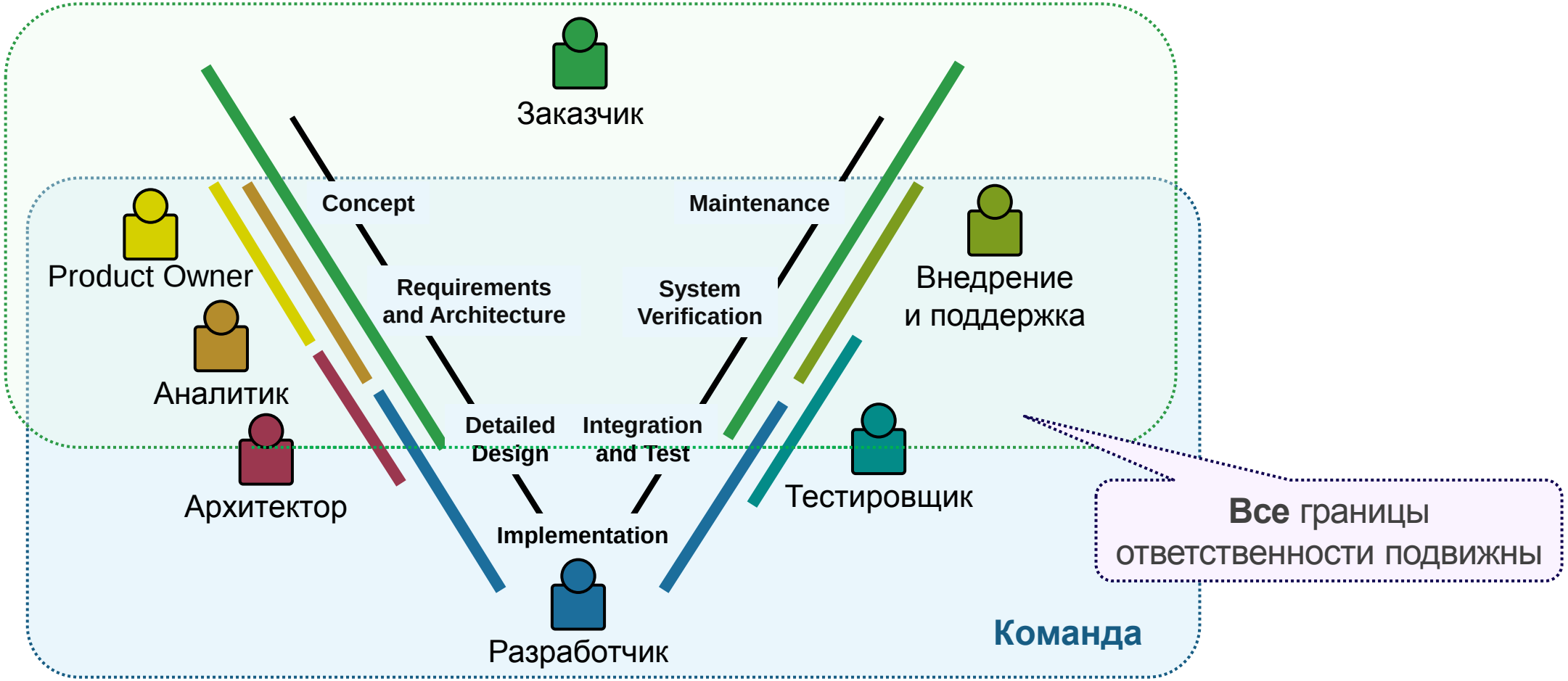




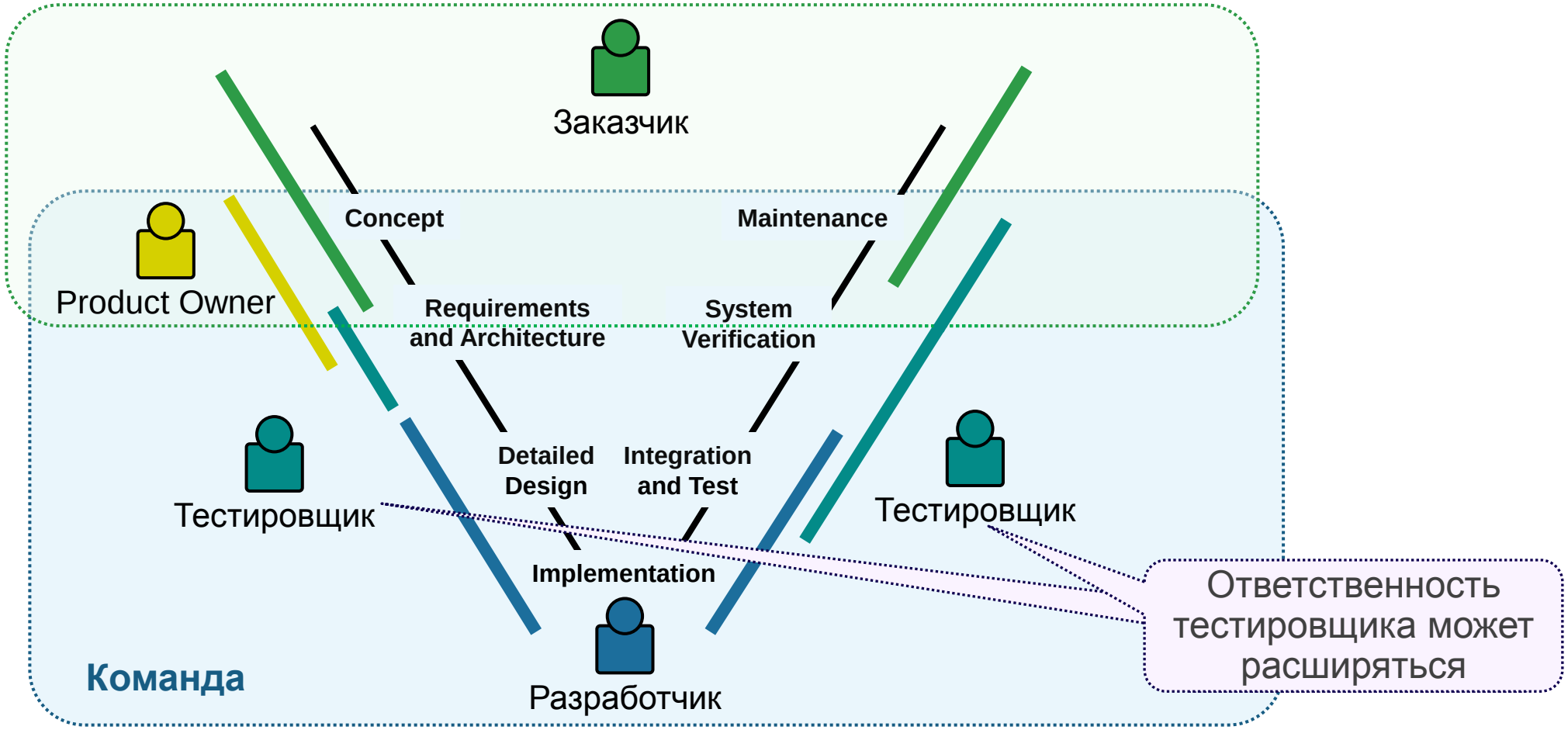
Удовлетворить пользователей → новые специализации



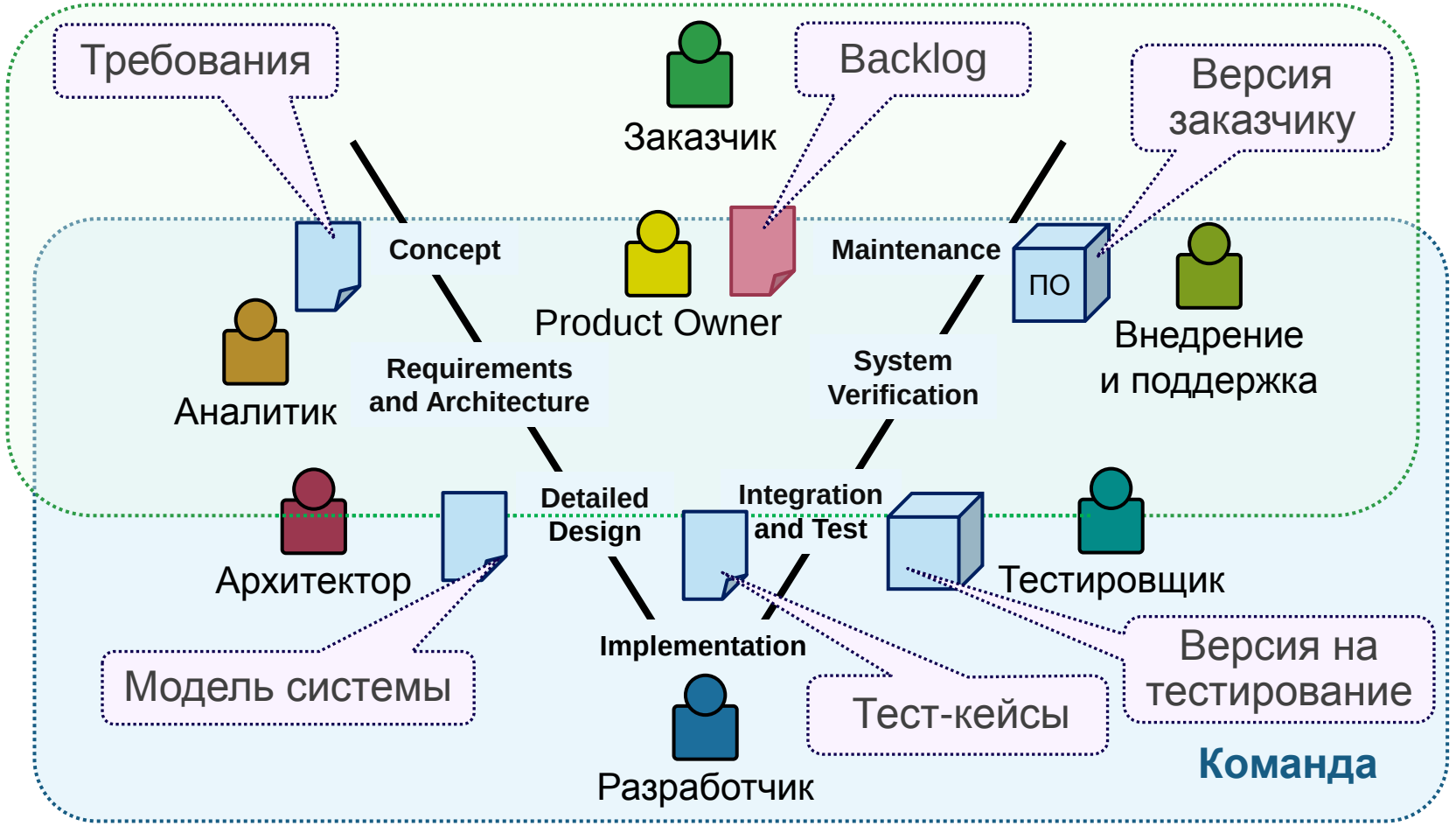
Команда включает много ролей



Не все роли могут присутствовать, а область заказчика может быть меньше



А еще нужно делить ответственность за артефакты



Как работать с границами ответственности

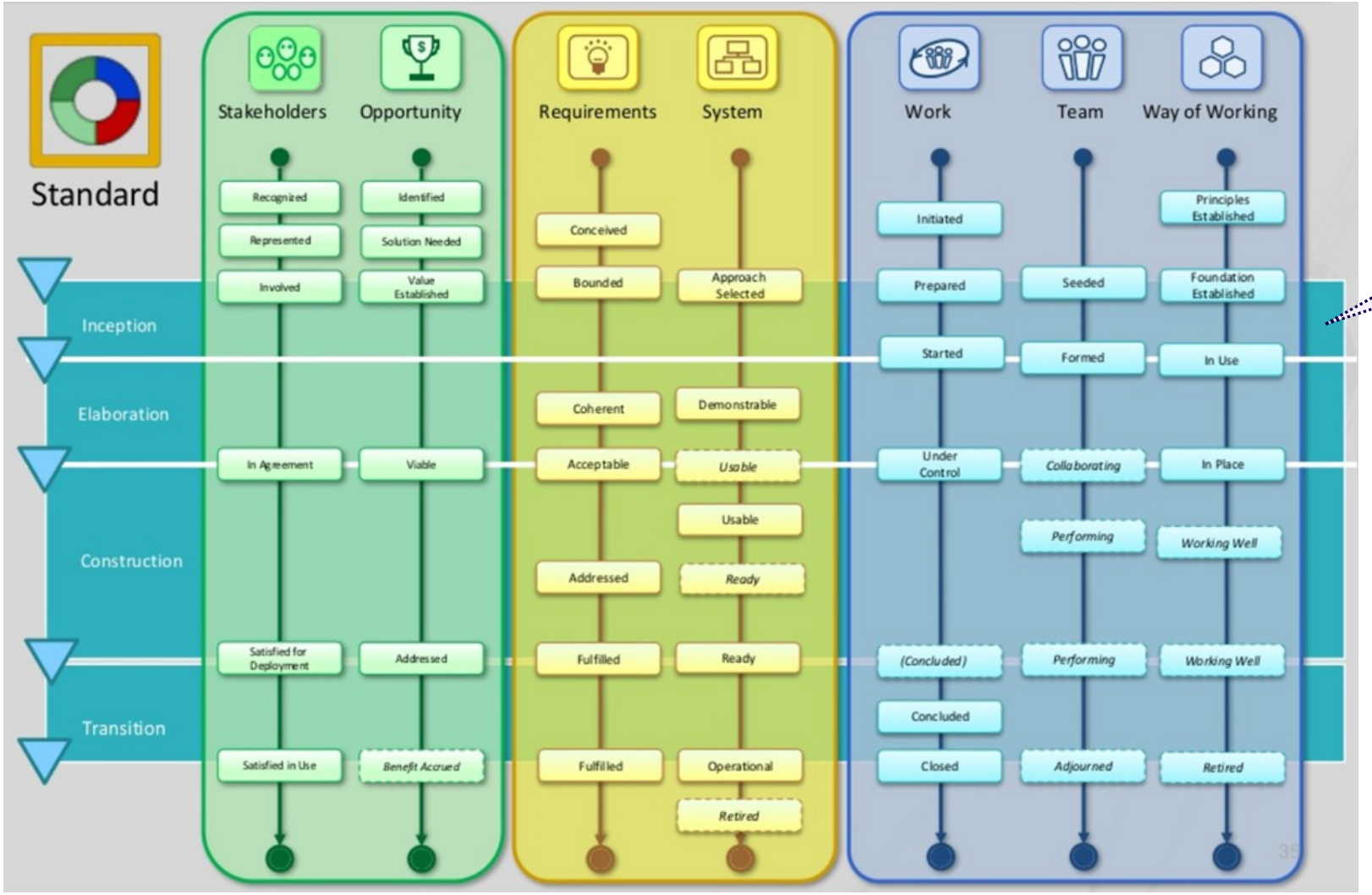
Используем Lifecycle OMG Essence

- Альфы задают объекты, состояния которых отражают движение проекта

Артефакт – частный вид объекта

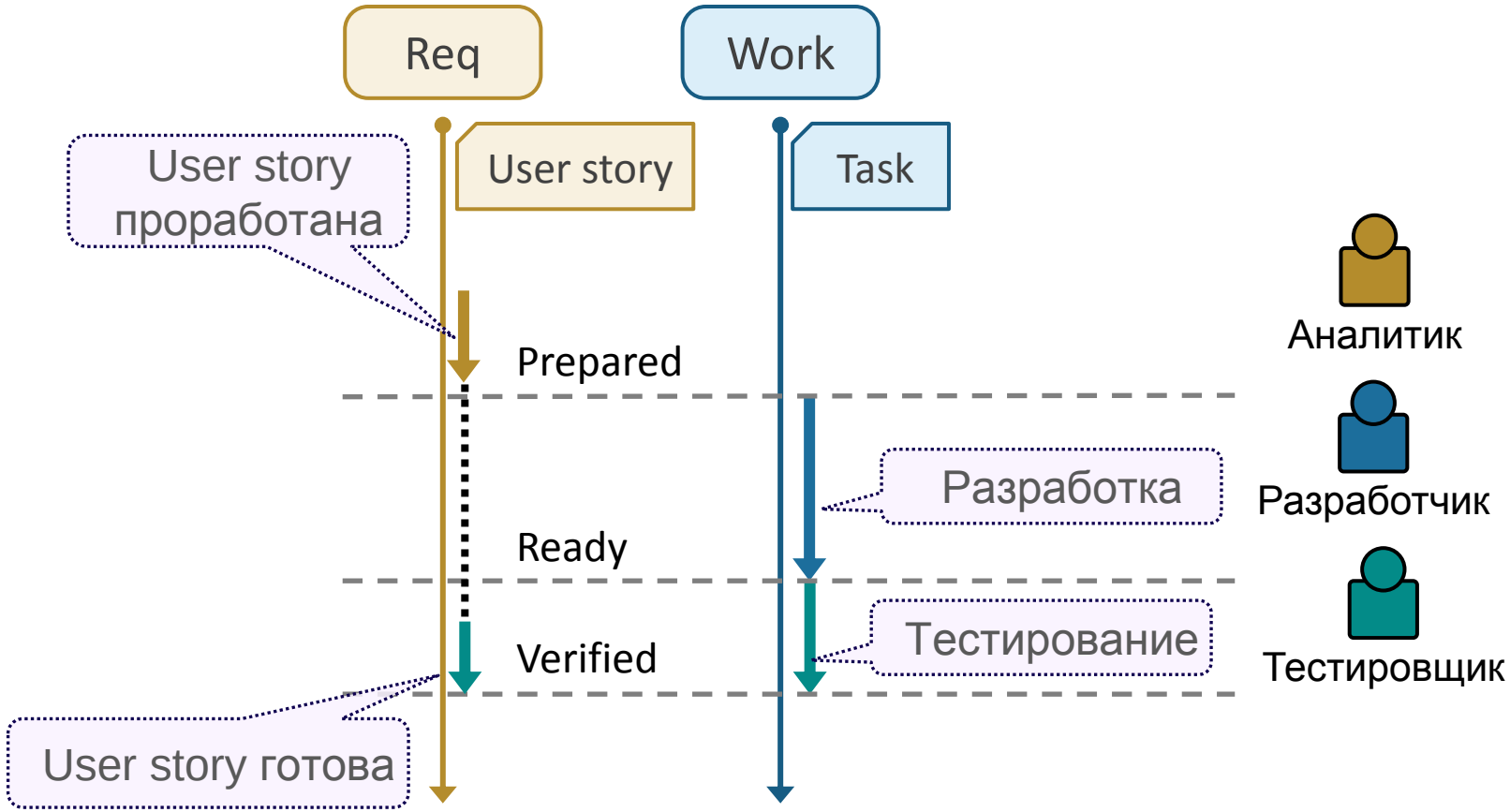
- Ответственность делится по этим объектам
- **Check lists** состояний определяют, в чем ответственность
- Lifecycle-диаграмма иерархична: можно представить весь проект, его релиз, спринт или разработку фичи

Каждая альфа живет собственной жизнью

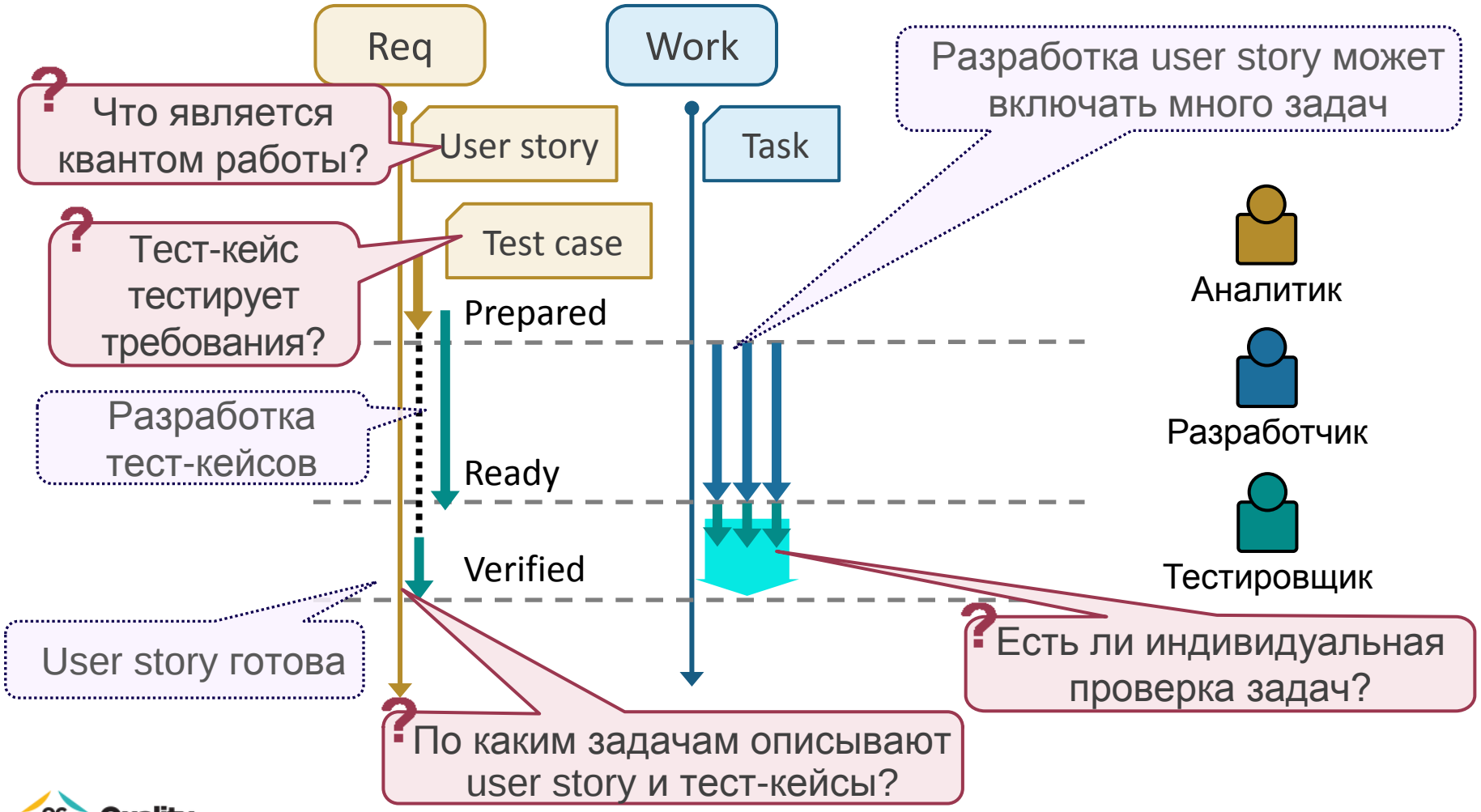


Пример: жизненный цикл «стандартного» проекта

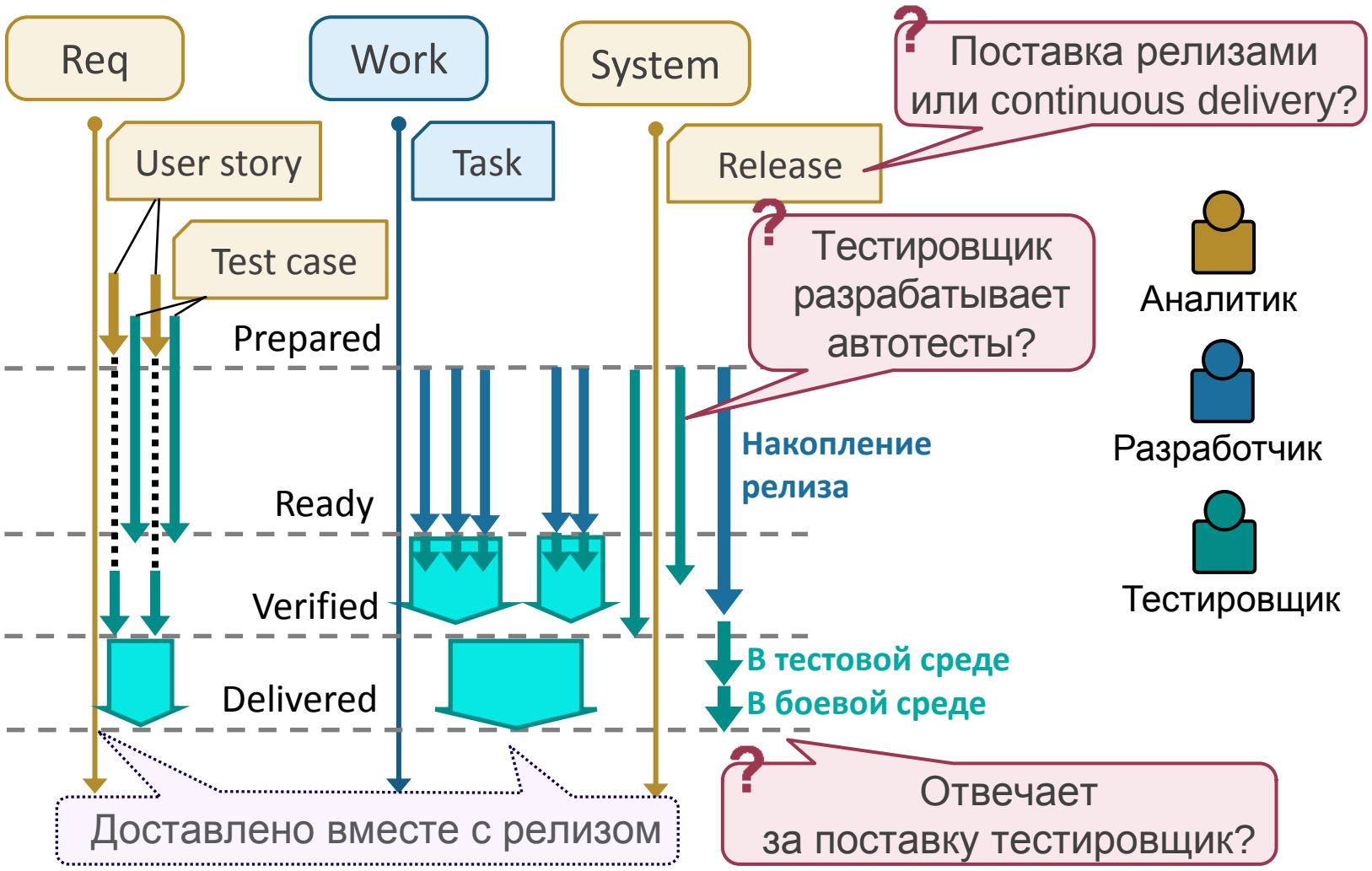
Простейший вариант – ответственность за проверку задачи



Ответственность за готовность user story

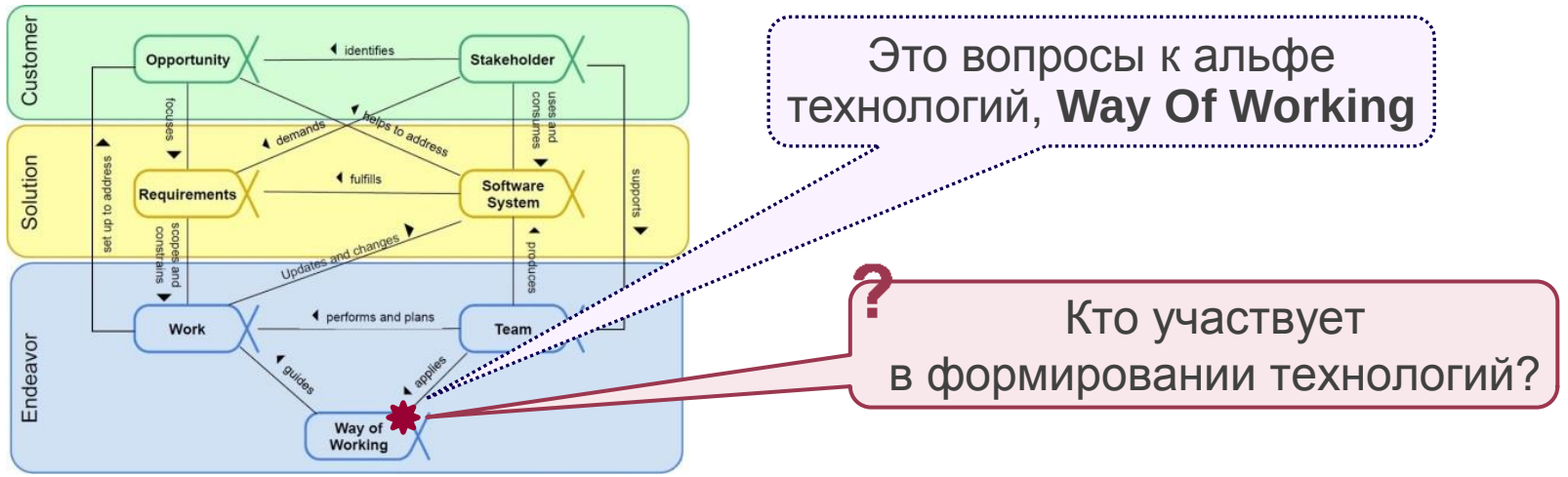


Ответственность за поставку системы



Что значит «может быть так»? Нужно ли иначе?

- Этот способ работы в вашей разработке сложился исторически или был спроектирован?
- Способ был адекватен особенностям разработки? Какие цели он обеспечивает?
- Способ продолжает оставаться адекватным сейчас или его пора заменять? Чем его заменять?

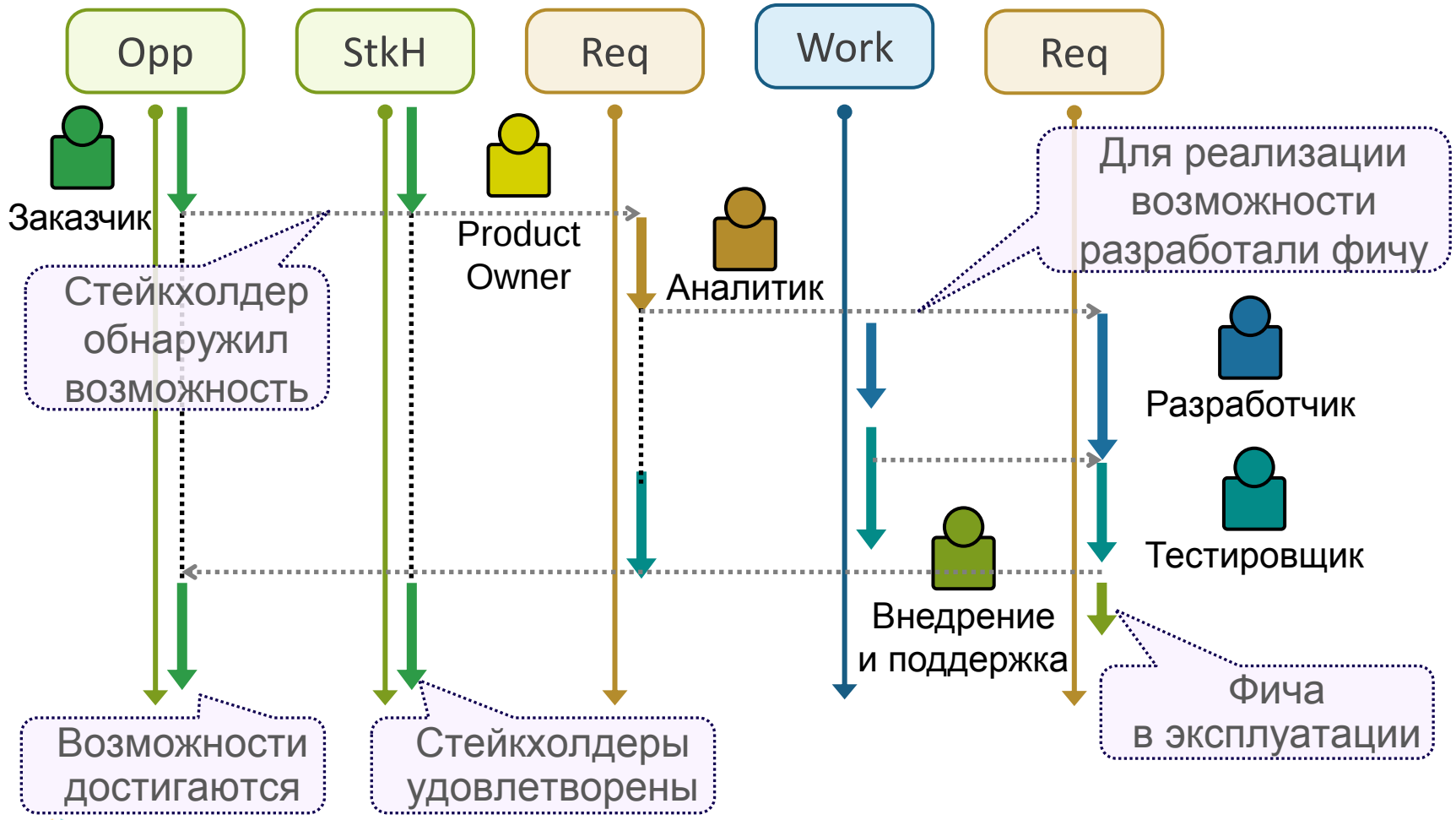


Как определять технологии?

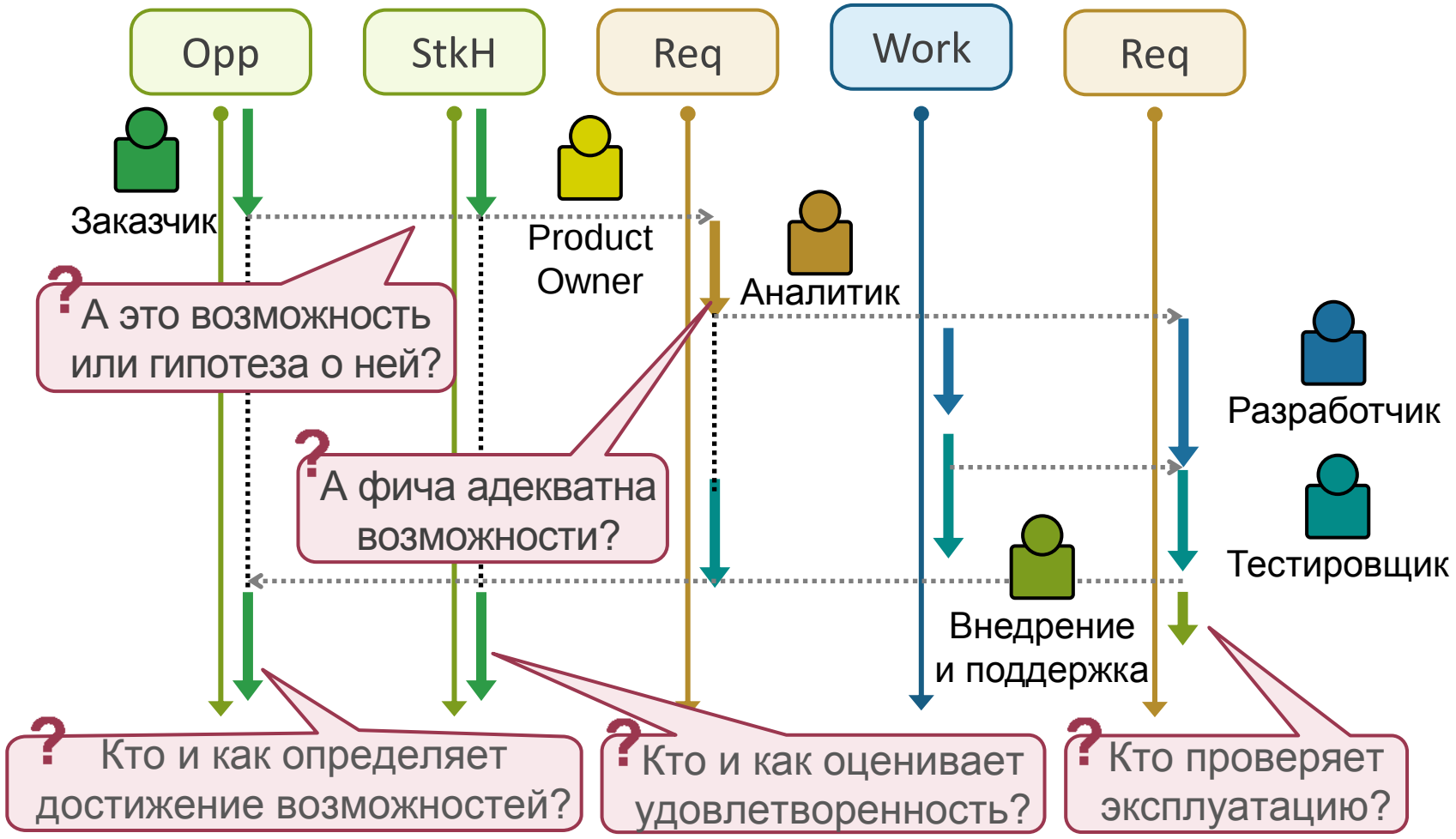
- Нужна ли разработке continuous delivery
  или уместна поставка релизов?
- Нужны ли разработке автотесты и, если нужны,
  то в каком объеме?

Это зависит от назначения разработки.
Альфы **Opportunity** и **Stakeholder**

Удовлетворенность стейкхолдеров и обеспечение возможностей бизнеса



А кто и как проверяет цели и их достижение?



Ответственность за возможности

- Заказчик отвечает за опознание возможности или выдает гипотезы?
- Могут ли стейкхолдеры заказчика оценить проект фичи на соответствие возможности?
- Проявляют ли стейкхолдеры заказчика возможности для бизнеса в своих запросах?

Если возможности – лишь гипотезы или нет гарантии их достижения

- Необходимо определить ответственность и способ оценки гипотез
- Предпочтительно continuous delivery для быстрого цикла реализации
- Полезно A/B-тестирование
- Полезно применять тестирование гипотез до реализации или с помощью макетов

A/B-тестирование и проверка на макетах может входить в обязанности тестировщика, а может выполняться аналитиком или маркетологом

Как релиз приходит к пользователям?

- Можно ли автоматически проверить критичный функционал?
- Что тестируем автоматически?
- Отгружает релиз человек или автомат?
- Можно ли отменить отгрузку?

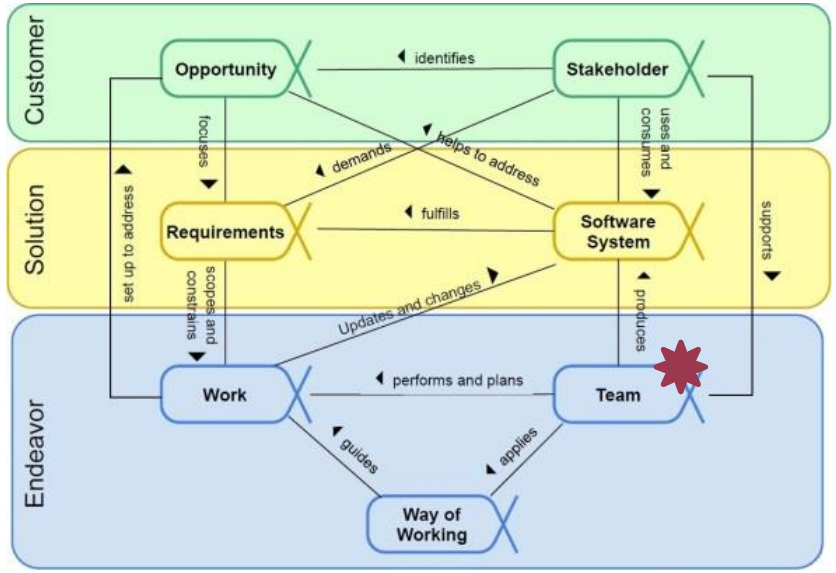
Continuous delivery

- Кто пишет новости версии?
- Эксплуатация – отдельная команда?
- Кто обучает пользователей?

DevOps

Область ответственности тестировщика и способ его работы сильно зависят от ответов на эти вопросы

Team: как нам понимать друг друга и эффективно сотрудничать

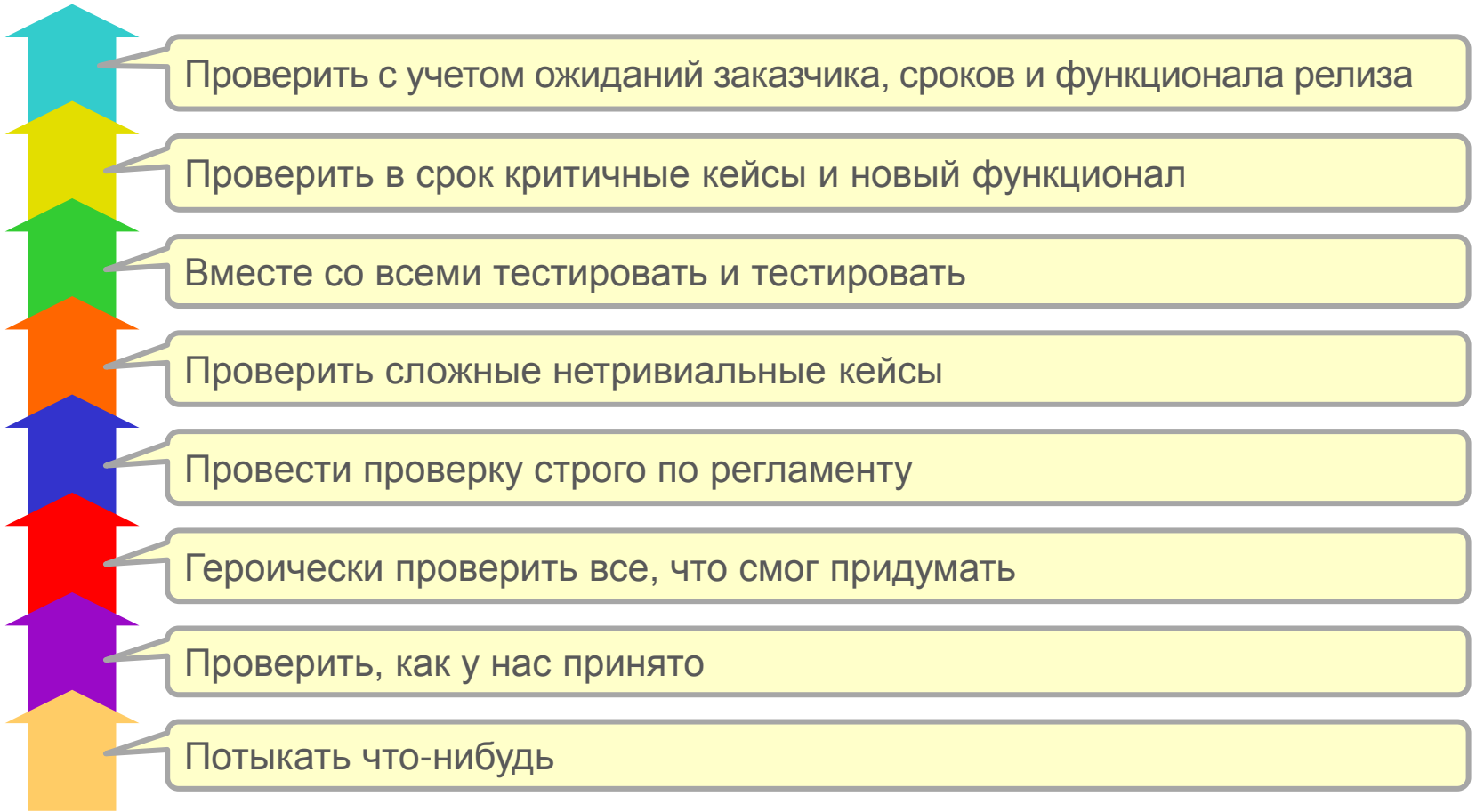


Team

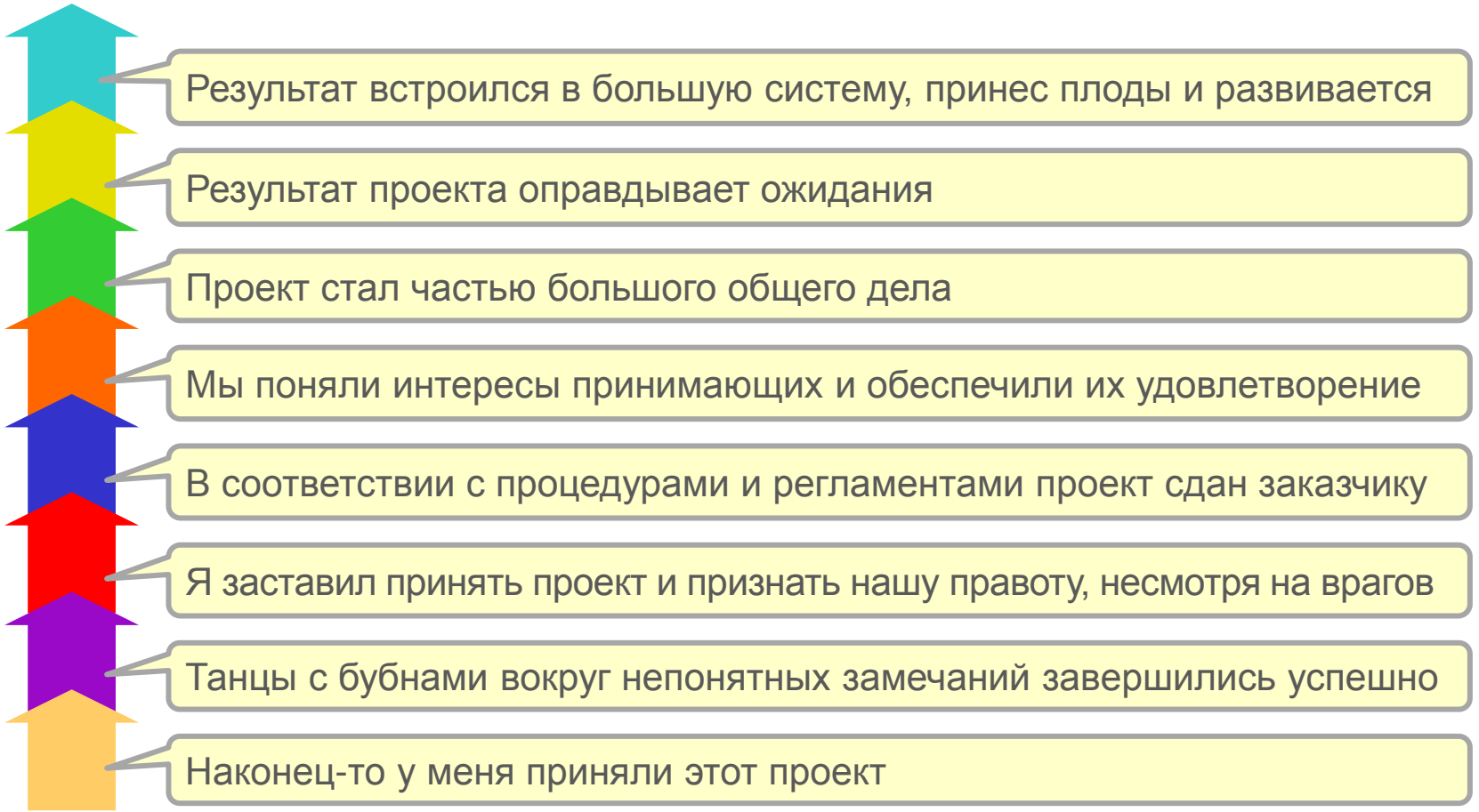
- Каковы ценности и нормы?
- Как организована команда?
- Как организовано взаимодействие?
- Как решаются конфликты?
- Как идет передача ответственности?

Представления о ценностях и нормах образуют устойчивые фреймы.
Их модель дает **Спиральная динамика**, где выделено **8 уровней**

Что значит «протестировать релиз»?



Что значит «успешно завершить проект»?



Подводя итоги

- Каждой ИТ-разработке нужно **свое** качество
- Ответственность тоже делится **по-своему**
- Надо договариваться об идеальной картине, учитывая:
  - представления стейкхолдеров и команды
  - объективные особенности проекта
- А затем – работать над воплощением идеала

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

На сайте много материалов по Agile, бирюзовым организациям и спиральной динамике, мои доклады, статьи и конспекты книг

Приложение. Ссылки из презентации

Мои доклады и материалы

- Как строить свой профессиональный путь - схемы самоопределения
- Мыслить проектно: история и современность
- Краткое описание уровней Спиральной динамики (другие мои материалы)
- Спиральная динамика для аналитика - работа на стыке культур

Другие материалы

- Стандарт OMG Essence – Kernel and Language for Software Engineering Methods
- Конкретные описания процессов по OMG на сайте Ивара Якобсона
- Курс системного мышления Анатолия Левенчука: coursera, учебник

Книги

- Энтони Лаудер «Культуры программных проектов» Оригинал, перевод (pdf)
- Jack W. Reeves. What is software design (перевод)
- Том ДеМарко «Человеческий фактор»
- Фредерик Брукс «Мифический человеко-месяц»
- Ивар Якобсон «The Essence of Software Engineering»
- Дон Бек и Крис Кован «Спиральная динамика»