

# СТРУКТУРА

## ВВЕДЕНИЕ

QC

ТЕСТИРОВАНИЕ ПО

QA

## РАЗРАБОТКА ПРОГРАММНОГО ОЕСПЧЕНИЯ

WATERFALL

V-MODEL

## SCRUM

## AGILE

AGILE МАНИФЕСТ

ЦЕННОСТИ AGILE

## ЦИКЛ ТЕСТИРОВАНИЯ

### ТЕСТИРОВАНИЕ ПО

### КОМАНДА

### ТРЕБОВАНИЕ

### КРИТЕРИИ АНАЛИЗА ТРЕБОВАНИЙ

### МЕТОДЫ ТЕСТИРОВАНИЯ ТРЕБОВАНИЙ

### ВИДЫ ТЕСТОВ

УРОВНИ ТЕСТИРОВАНИЯ

ОШИБКИ НА УРОВНЯХ ТЕСТИРОВАНИЯ

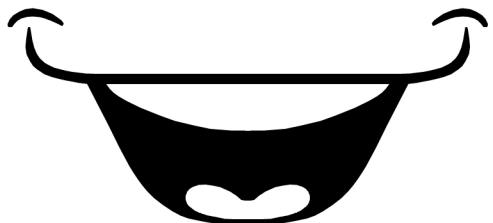
### ВИДЫ ТЕСТОВ

## ЦИКЛ ТЕСТИРОВАНИЯ

## РОЛИ В ПРОЦЕССЕ УПРАВЛЕНИЯ ДЕФЕКТОМ

# QA LIGHT

HELP  
BOOK



## **СТРУКТУРА**

**ЖИЗНЕНЫЙ ЦИКЛ ДЕФЕКТА**

**ЖИЗНЕНЫЙ ЦИКЛ ДЕФЕКТА ПО РОЛЯМ**

**ТИПЫ НЕИСПРАВНОСТЕЙ**

**УЩЕРБ ОТ ОШИБКИ**

**ПРИОРИТЕТ ОШИБКИ**

**ОПРЕДЕЛЕНИЕ ДЕФЕКТА**

**ОТЧЕТ О ДЕФЕКТЕ**

**ХОРОШИЙ ОТЧЕТ О ДЕФЕКТЕ**

**ТЕСТ ПЛАН**

**TEST CASE**

**ТЕХНИКИ ТЕСТ ДИЗАЙНА**

**ГРАНИЧНЫЕ УСЛОВИЯ**

**КЛАССЫ ЭКВИВАЛЕНТНОСТИ**

**МЕТОД ПАР**

**ТЕСТОВОЕ ПОКРЫТИЕ**

**МАТРИЦА ПОКРЫТИЯ**

**АРТЕФАКТЫ ТЕСТИРОВАНИЯ**

**ESTIMATE TIME**

**КАЧЕСТВА ТЕСТИРОВЩИКА**

**СОБЕСЕДОВАНИЕ**

**ПЕРВЫЙ ДЕНЬ НА НОВОЙ РАБОТЕ**

**BUILDS AND VERSIONS**

**ОСОБОЕ ВНИМАНИЕ**

**ПОИСК ИНФОРМАЦИИ О ПРОДУКТЕ**

## ВВЕДЕНИЕ

### Quality Control (контроль качества)

это процесс проверки качества каждой сущности и всех факторов, влияющих на процесс разработки

### Quality Assurance (обеспечение качества)

это ряд мероприятий, направленных на то, что все процессы и требования к качеству разрабатываемого продукта выполняются в полной мере

Quality Assurance гарантирует, что процесс поставлен правильно и дает предсказуемый результат, в то время как Quality Control гарантирует, что продукт удовлетворяет указанному набору требований.

**Тестирование** - процесс исследования ПО с целью получения информации о качестве продукта

**Тестирование ПО** - процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляется путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом и который осуществляется специально подготовленными QC/QA инженерами.

# АРТЕФАКТЫ ТЕСТИРОВАНИЯ

## План тестирования (Test Plan)

- это документ описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения

## Тест кейс (Test Case )

- это последовательность действий, по которой можно проверить соответствует ли тестируемая функция установленным требованиям.

## Набор тест кейсов и тестов (Test set & Test suite)

- Это набор тест кейсов(тест кейсов) объединённых в категорию по определённым признакам

## Дефекты / Баг Репорты (Bug Reports / Defects)

- это документы, описывающие ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

## Матрица покрытия (traceability matrix)

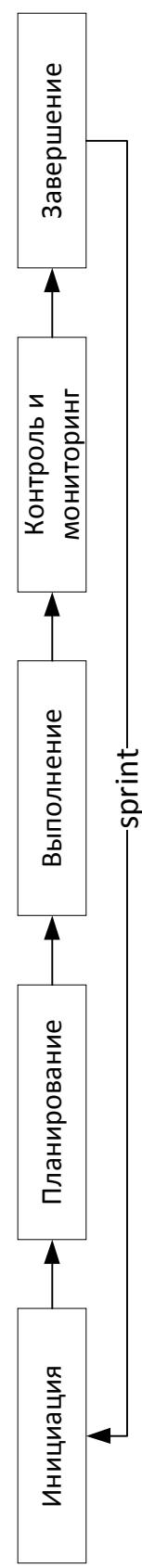
Таблица, содержащая отношение требований к подготовленным тестовым сценариям (Test Cases)

## Чеклист (Check-list)

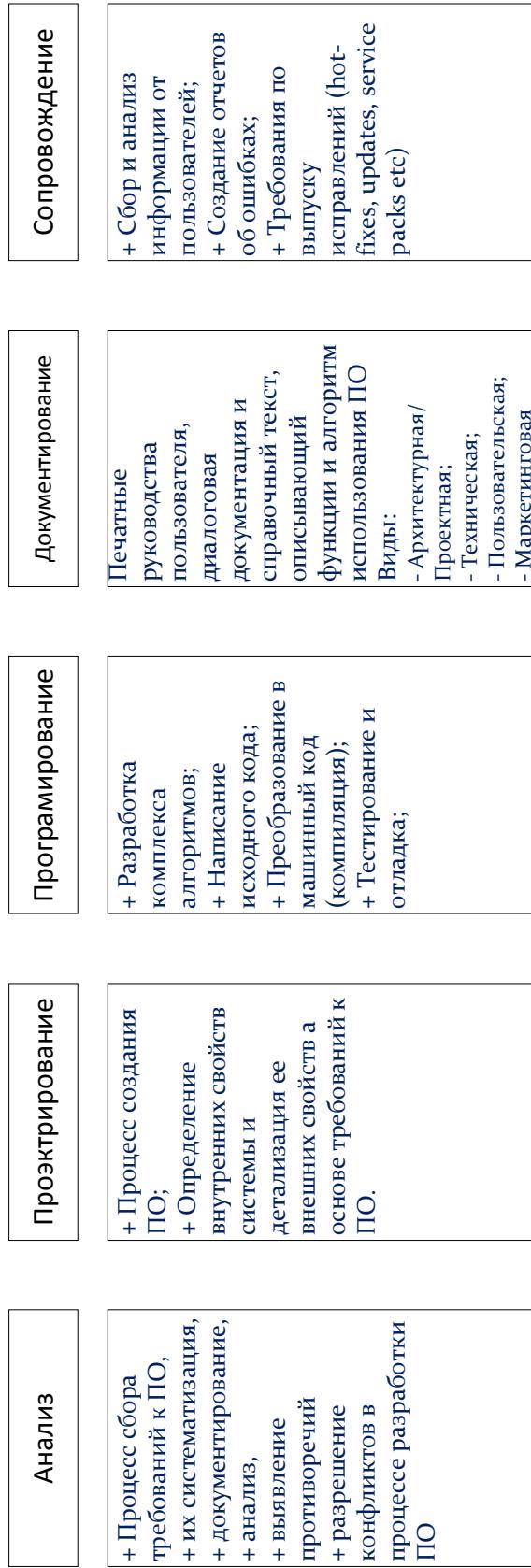
Таблица создана для ускорения процесса тестирования. Имеющая расширенное краткое описание теста и место для отметки о его выполнении.

# РАЗРАБОТКА ПРОГРАММНОГО ОСПРЕЧЕНИЯ

ЖИЗНЕННЫЙ ЦИКЛ  
ПРОЕКТА



Стадии процесса

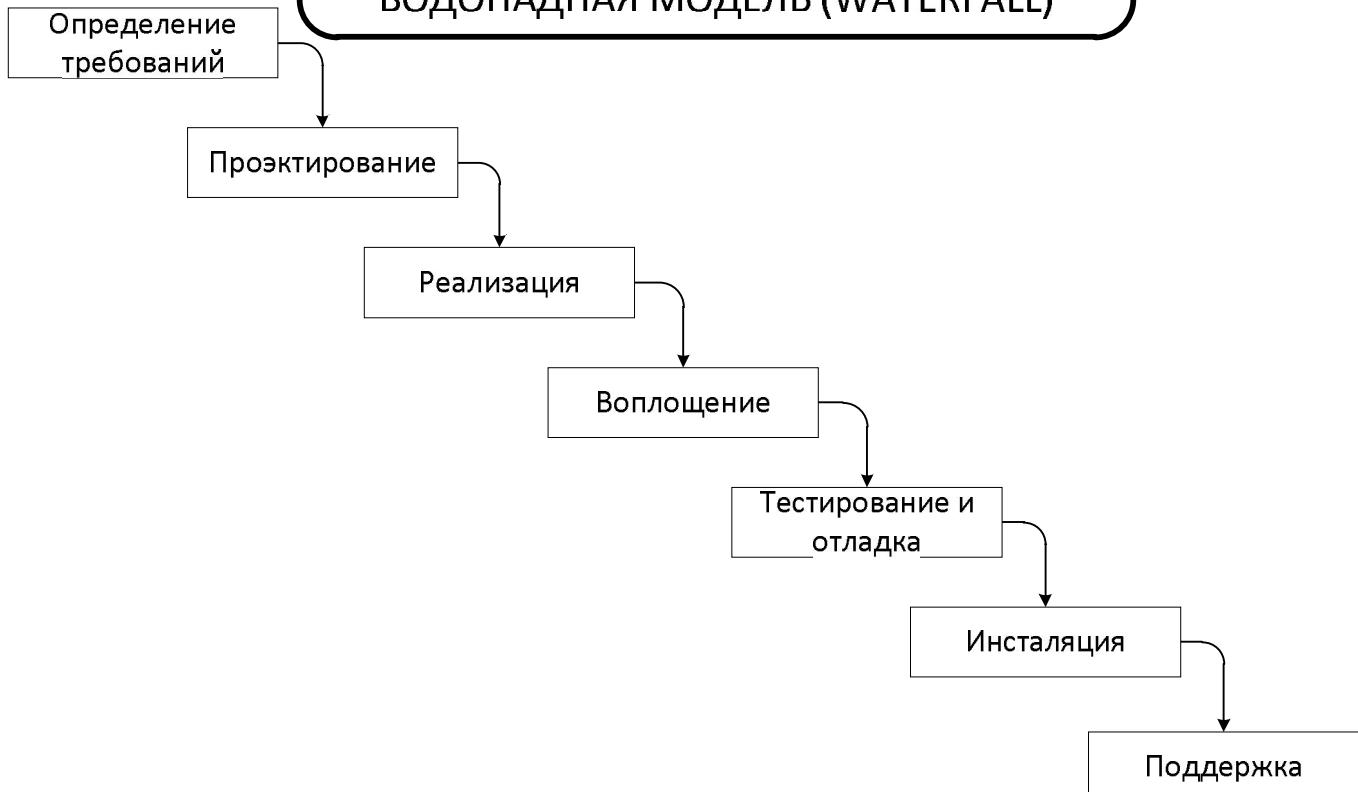


Нотации –  
схематическое  
выражение  
характеристик:  
- Блок-схемы;  
- ER-диаграммы;  
- UML-диаграммы;  
- Макеты.

Печатные  
руководства  
пользователя,  
диалоговая  
документация и  
справочный текст,  
описывающий  
функции и алгоритм  
использования ПО  
Виды:  
- Архитектурная/  
Проектная;  
- Техническая;  
- Пользовательская;  
- Маркетинговая

+ Сбор и анализ  
информации от  
пользователей;  
+ Создание отчетов  
об ошибках;  
+ Требования по  
выпуску  
исправлений (hot-  
fixes, updates, service  
packs etc)

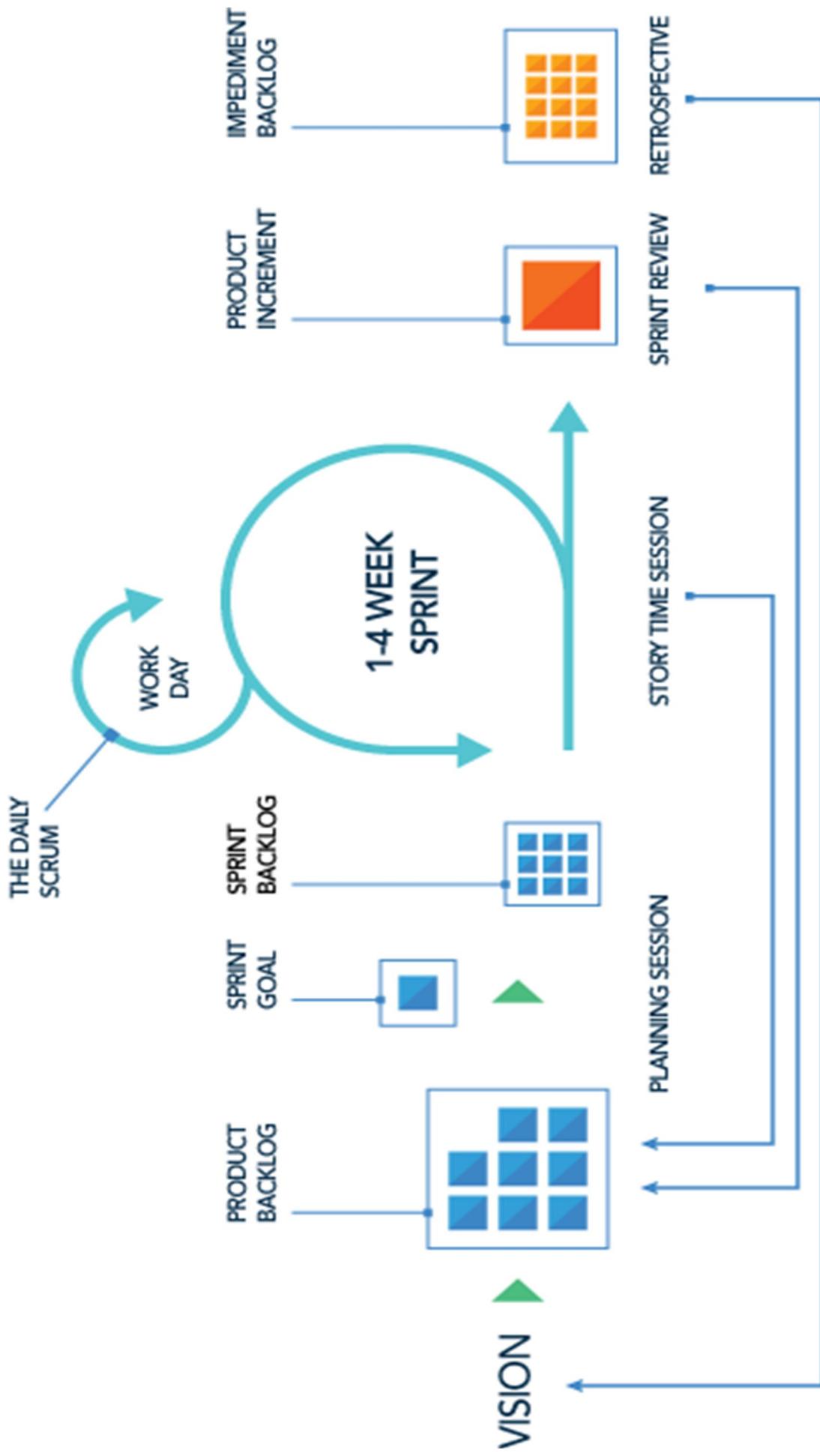
## ВОДОПАДНАЯ МОДЕЛЬ (WATERFALL)



## V-образная модель (V-model)



## SCRUM



# AGILE

Scrum

Kanban

XP

TDD

## Agile манифест

Личности и их взаимодействие важнее, чем процессы и средства

Работающее ПО важнее, чем исчерпывающая документация

Сотрудничество с заказчиком важнее, чем обязательства по контракту

Реагирование на изменения важнее, чем строгое следование плану

## Ценности Agile

### Гибкость и простота

Agile-процессы готовы к изменениям требований даже на поздних этапах разработки.  
Важна простота - искусство увеличения объема работ, которых удалось избежать.

### Частые релизы

Наивысший приоритет - удовлетворенность заказчика:  
- Ранние и периодические поставки ПО  
- ПО работающее и ценное для заказчика  
Продолжительность каждой итерации - от пары недель до пары месяцев.  
Предпочтение - коротким интервалам.

### Самоорганизующаяся команда

Над проектом работают мотивированные люди. Создаются все условия, поддержка и полное доверие.  
Самые лучшие архитектуры, требования и дизайны систем создаются самоорганизующимися командами.  
Команда сама организует оптимальный процесс.

### Больше общения

Потенциальные пользователи системы и разработчики должны работать **вместе** на протяжении всего проекта.  
Самый действенный и эффективный способ обмена информацией как внутри команды разработчиков, так и с внешним миром - **непосредственное общение**.

## СОСТАВЛЯЮЩИЕ

Product backlog

Sprint

Daily scrum

Taskboard

User-story

Product owner

Sprint-backlog:  
tasks

Scrum master

Burndown  
chart

Sprint planning

Demo

Ретроспектива спринта

Abnormal  
Termination

## AGILE

Product backlog	Product backlog один на весь релиз Им владеет менеджер продукта (“ <b>product owner</b> ”) Он не статичен – записи можно добавлять, удалять, менять им приоритет Общедоступен, но поддерживается одним человеком
User-story	Содержит список функциональных единиц системы (“ <b>user stories</b> ”), запланированных на след релиз
Sprint	Фаза разработки состоит из нескольких итераций – спринтов. Обычно спринт длится 2-4 недели. Этапы: Планирование Разработка Демонстрация Ретроспектива
Sprint-backlog: tasks	Описывает задачи, запланированные командой на спринт Задачи – действия, необходимые для реализации запланированной на спринт функциональности В описание задачи входит ее оценка
Daily scrum	Проводится каждый день в фиксированное время Рекомендуется проводить стоя в течение 10-15 минут 1 минута на каждого 3 Вопроса: Что сделал; Что буду делать; Возникшие проблемы Если что-то нужно обсудить, назначается время после скрама
Sprint planning	Проводится в начале спринта Участвует вся команда User stories разбиваются на задачи и оцениваются членами команды В результате команда подписывается на ту функциональность, на которую хватает времени спринта

# AGILE

## Taskboard

## Burndown chart

Диаграмма сгорания задач показывающая количество сделанной и оставшейся работы  
Для спринта;  
Для проекта.  
Обновляется ежедневно для визуализации работы команды  
График должен быть общедоступен

## Abnormal Termination

Остановка спринта раньше срока в исключительных ситуациях  
Команда понимает, что не может достичь цели проекта в срок  
Владелец проекта понимает, что исчезла необходимость в реализации цели спринта  
Обсуждение причины остановки  
Запуск нового спринта

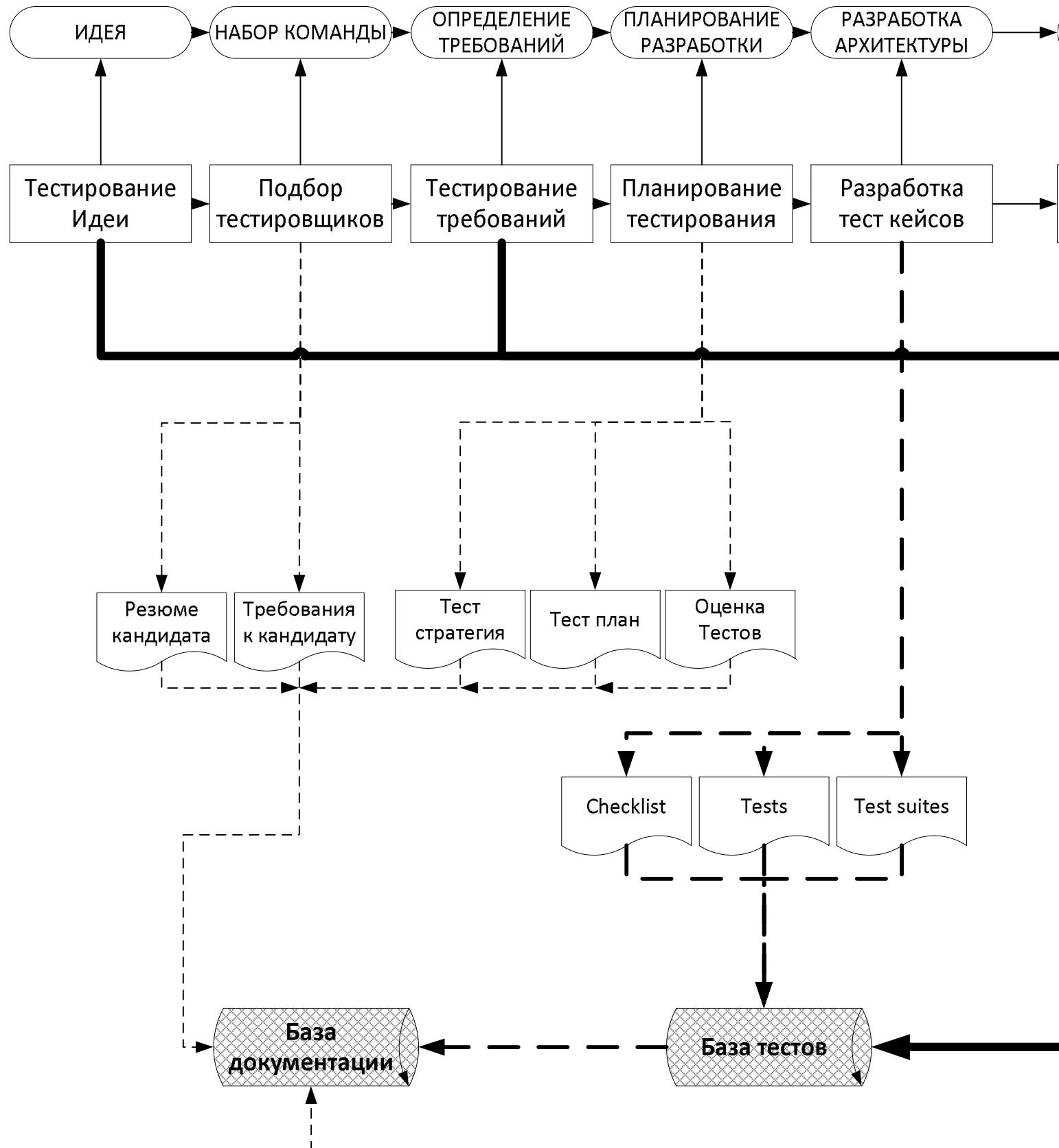
## Demo

В конце каждого спринта проводится ревью  
Это демонстрация реализованной функциональности  
В ней может участвовать любой человек, задействованный в проекте  
В идеале после каждой демонстрации можно отправлять продукт заказчику

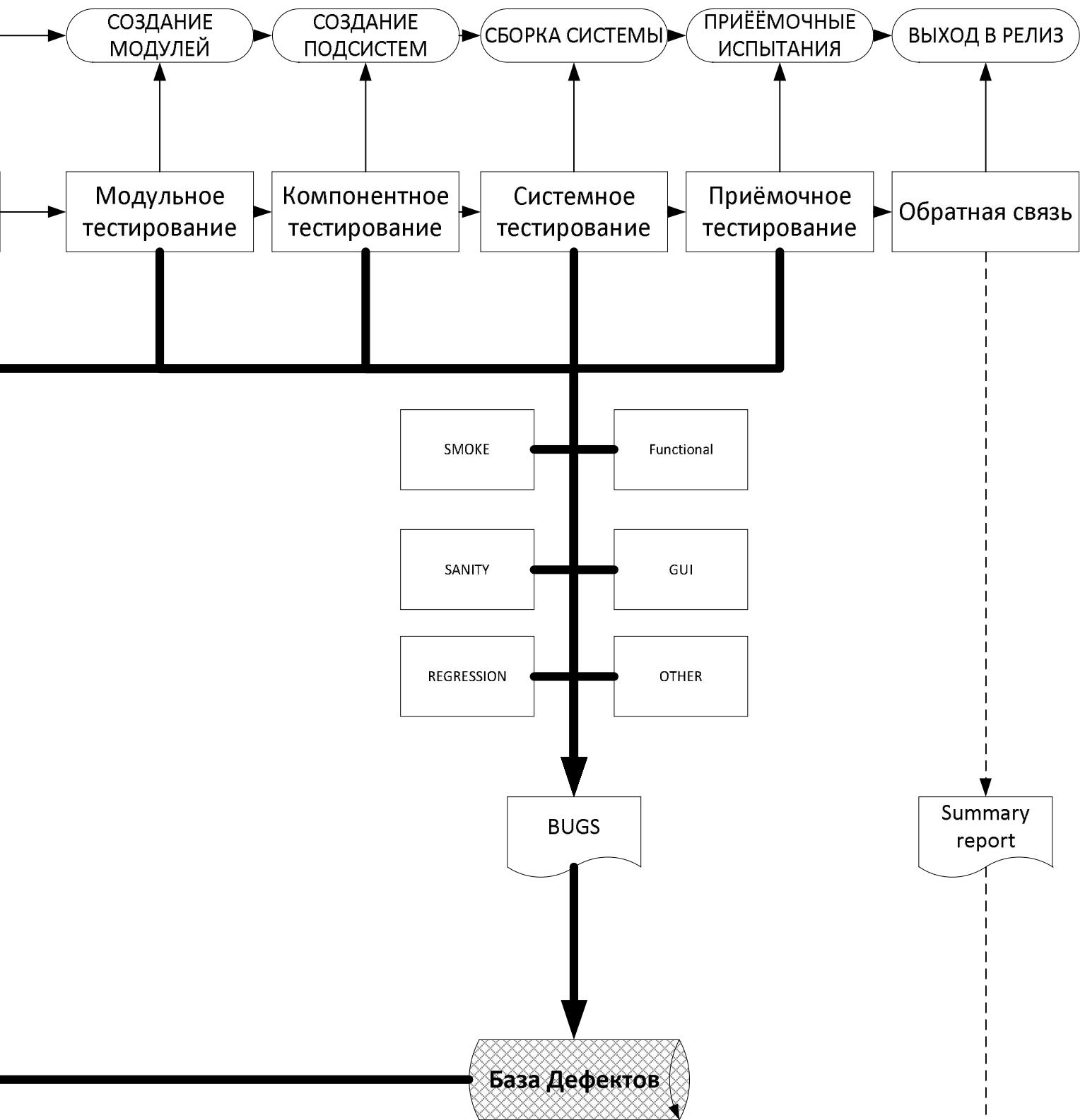
## Ретроспект ива спринта

После каждого спринта (ревью)  
Участвуют все члены команды  
Цель - осознать:  
Что было хорошо?  
Что могло бы быть лучше  
Это обсуждение процесса, а не технических сложностей

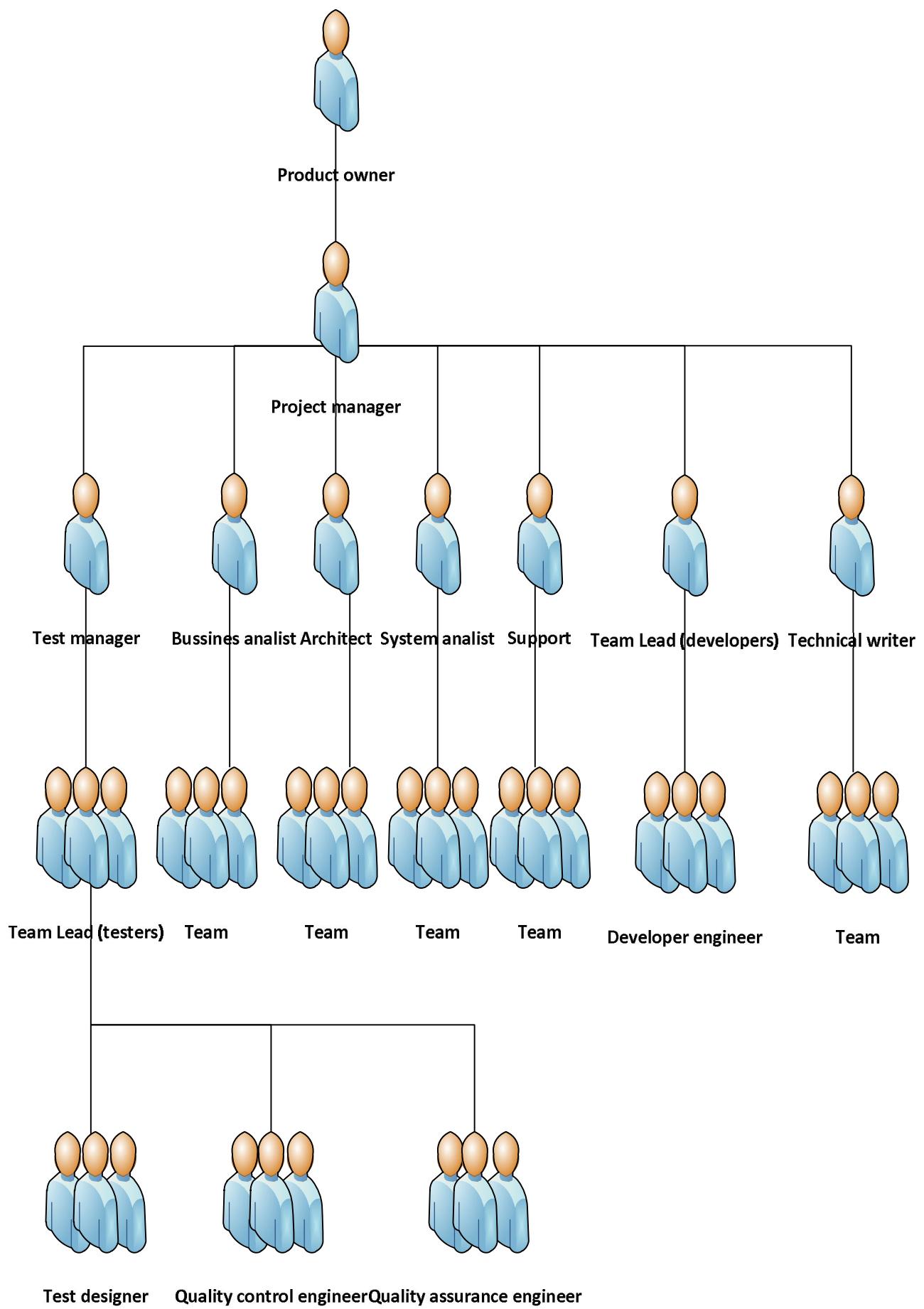
## ЦИКЛ ТЕСТИРОВАНИЯ



## ЦИКЛ ТЕСТИРОВАНИЯ



# КОМАНДА



## ТРЕБОВАНИЕ

Требование -- это потребность или ожидание, которое:

- (а) установлено в явном виде, или
- (б) «наследуется» как обязательное из других систем требований (напр., государственных и ведомственных законоположений), или
- (в) подразумевается (является «обычным»)

## ТИПЫ ТРЕБОВАНИЙ

### Бизнес требования

Цели и задачи продукта.

#### Нефункциональные требования

Описывают как должна работать система.

#### Функциональные требования

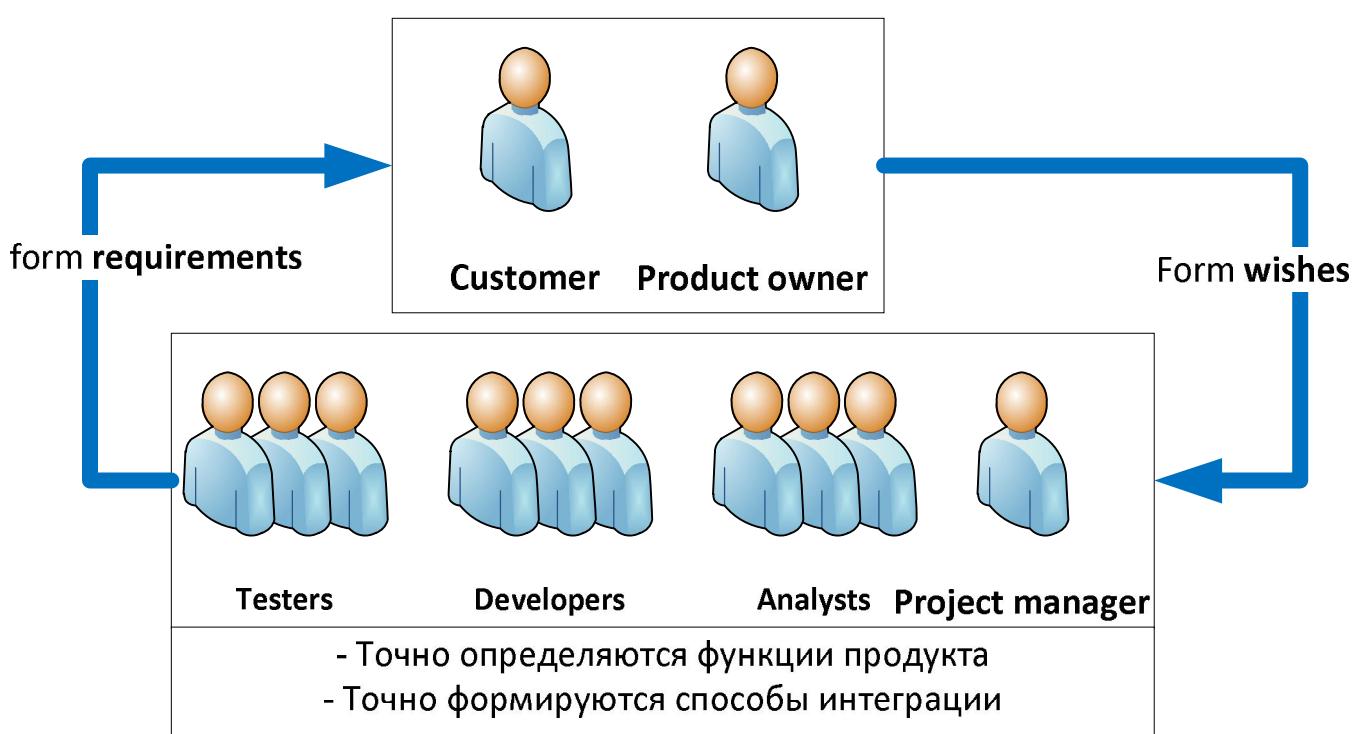
Описывают что должна делать система.

#### Предположения и ограничения

Специфика доменной области, которая накладывает ограничения на поведение или дизайн системы.

#### Требования пользователей

Потребности отдельных групп заказчиков/пользователей.



## КРИТЕРИИ АНАЛИЗА ТРЕБОВАНИЙ

### Реализуемость

- Каждое из требований возможно реализовать ;
- Учитываются доступные ресурсы и время.

### Корректность (Правильность)

- Каждое требование должно точно описывать то, что должно быть разработано.

### Прослеживаемость

- Уникальный идентификатор;
- Система идентификации позволяет добавлять, удалять и разбивать требования без изменения идентификатора других требований

### Однозначность

- Одноковая интерпретация требования командой;
- Требование описано четко, просто, кратко;
- Все термины и аббревиатуры описаны и определены.

### Полнота

- Все требования задокументированы;
- Каждое требование содержит всю информацию для проектирования, разработки и тестирования;
- Нет ссылок на несуществующие данные (таблицы, иллюстрации, документы);

### Тестируемость

- Требование должно быть сформулировано так, чтобы можно было доказать соответствие системы предъявленному требованию;
- Требование не должно содержать неизмеримых или нетестируемых формулировок.

### Непротиворечивость

- Требование не конфликтует с другими требованиями;
- Требование не конфликтует с под-требованиями;
- Требование не конфликтует с законами, стандартами.

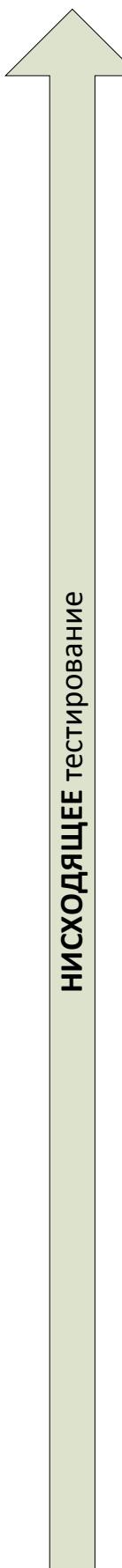
# МЕТОДЫ ТЕСТИРОВАНИЯ ТРЕБОВАНИЙ

ПРОВЕРКА ДОКУМЕНТАЦИИ	АНАЛИЗ ПОВЕДЕНИЯ СИСТЕМЫ	ПРОТОТИПИРОВАНИЕ
<ul style="list-style-type: none"><li>- Последовательный просмотр и проверка всех доступных требований;</li></ul>	<ul style="list-style-type: none"><li>- Формирование требований в формате «вход-выход», «событие-последствие», «условие-ответ».</li><li>- Позволяет проверить на полноту, понятность, однозначность</li></ul>	<ul style="list-style-type: none"><li>- Создание модели будущей системы.</li><li>- Позволяет проверить на полноту, корректность, реализуемость</li></ul>
<p><b>Применяется</b></p> <ul style="list-style-type: none"><li>- Заказчиками;</li><li>- Аналитиками;</li><li>- ПМами;</li><li>- Тестировщиками.</li></ul>	<p><b>Применяется</b></p> <ul style="list-style-type: none"><li>- Тестировщиками (test cases);</li><li>- Аналитиками (use cases).</li></ul>	<p><b>Применяется</b></p> <ul style="list-style-type: none"><li>- Архитекторами;</li><li>- Аналитиками.</li></ul>
<p><b>Плюсы</b></p> <ul style="list-style-type: none"><li>- Простота использования;</li><li>- Отсутствие специальных требований к проверяющему;</li><li>- Покрывает много критериев качества;</li><li>- Меньше затраты времени.</li></ul>	<p><b>Плюсы</b></p> <ul style="list-style-type: none"><li>- Хорошо проверяет требования;</li><li>- Представляет требования в структурированном и понятном виде;</li><li>- Результаты легко использовать для создания тест кейсов.</li></ul>	<p><b>Плюсы</b></p> <ul style="list-style-type: none"><li>- Пользователи получают возможность проверить решение;</li><li>- Наглядное пособие для разработчиков и тестировщиков;</li><li>- Проверка требования на реализуемость.</li></ul>
<p><b>Минусы</b></p> <p>Качество проверки зависит от проверяющего; Вовлечение различных специалистов; Наличие документов с требованиями</p>	<p><b>Минусы</b></p> <ul style="list-style-type: none"><li>- Требует большего количества времени;</li><li>- Требует специальной подготовки.</li></ul>	<p><b>Минусы</b></p> <ul style="list-style-type: none"><li>- Требует значительного времени;</li><li>- Специальная подготовка для создания прототипа.</li></ul>

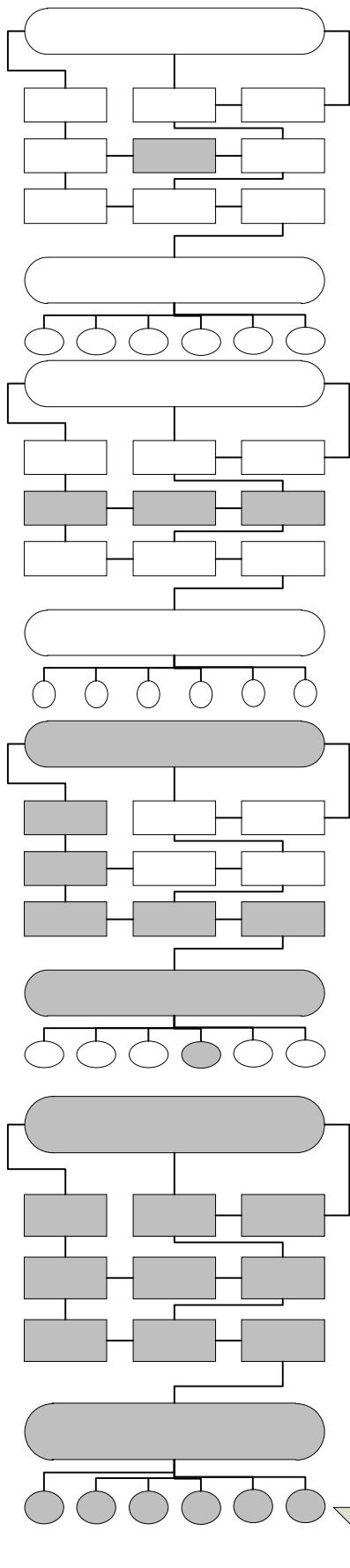
## Уровни тестирования

<b>Модульное тестирование</b>	Цель модульного тестирования состоит в выявлении локализованных в модуле ошибок в реализации алгоритмов, а также в определении степени готовности системы к переходу на следующий уровень разработки и тестирования
<b>Интеграционное тестирование</b>	Тестирование части системы, состоящей из двух и более модулей. Цель интеграционного тестирования - поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями.
<b>Системное тестирование</b>	<b>Система в целом</b> Пользовательский интерфейс <ul style="list-style-type: none"><li>- неверное использование ресурсов системы,</li><li>- непредусмотренные комбинации данных пользовательского уровня,</li><li>- несовместимость с окружением,</li><li>- Непредусмотренные сценарии использования,</li><li>- отсутствующая или неверная функциональность,</li><li>- неудобство в применении и т.д.</li></ul>
<b>Приемочное тестирование</b>	Формальный процесс тестирования, который проверяет соответствие системы требованиям и проводится с целью: <ul style="list-style-type: none"><li>- определения удовлетворяет ли система приемочным критериям;</li><li>- вынесения решения заказчиком или другим уполномоченным лицом принимается приложение или нет</li></ul>

## Уровни тестирования



- Выполняется программистом
- Тестируется минимальный элемент кода
- Тестируются элементов не зависящих от других элементов
- Поиск ошибок внутри кода
- Поиск ошибок в работе алгоритмов



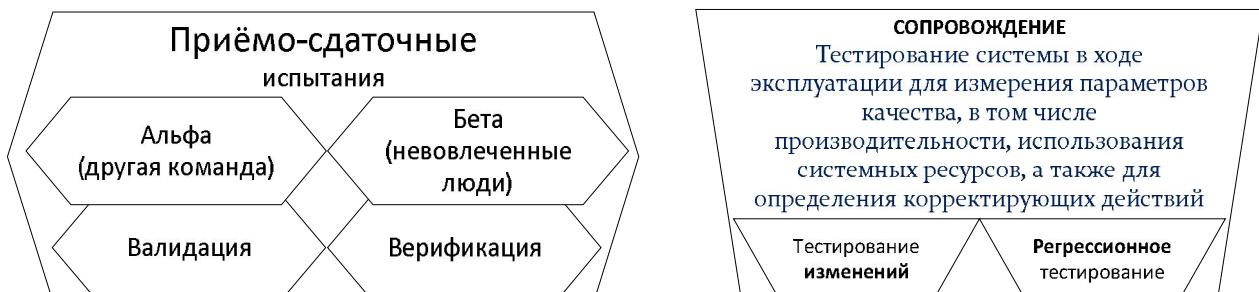
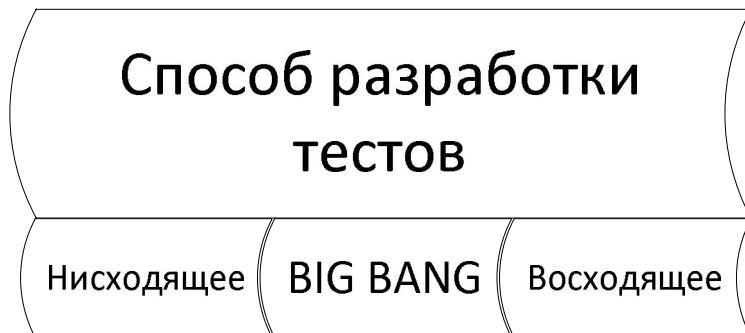
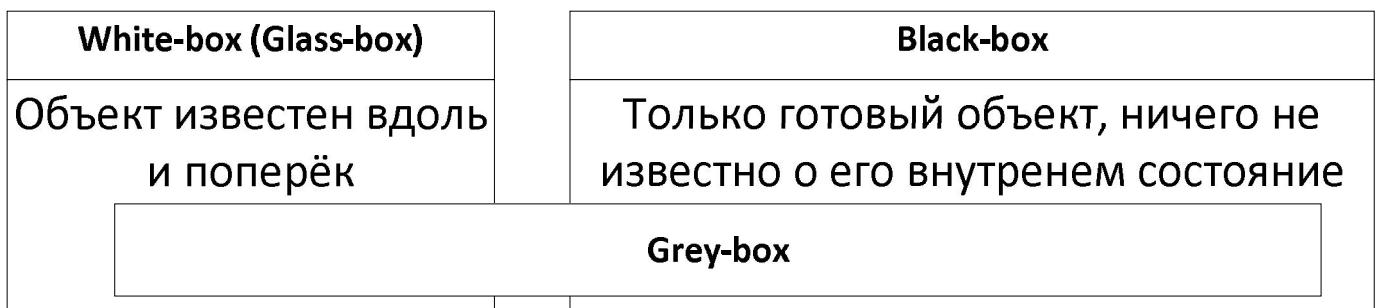
- Выполняется программистом
- проверяет правильность входных и выходных данных в\из модуля
- Проверяется взаимодействие с другими модулями
- Использование заглушок
- Выполняется тестировщиком или программистом
- Проверка взаимодействия двух

- Выполняется тестировщиком
- Проверка правильного использования системы
- Проверка выполнения всех требований
- Проверка не предусмотренных шагов
- Поиск неверной функциональности
- Проверка удобства пользования
- Поиск не заявленных функций

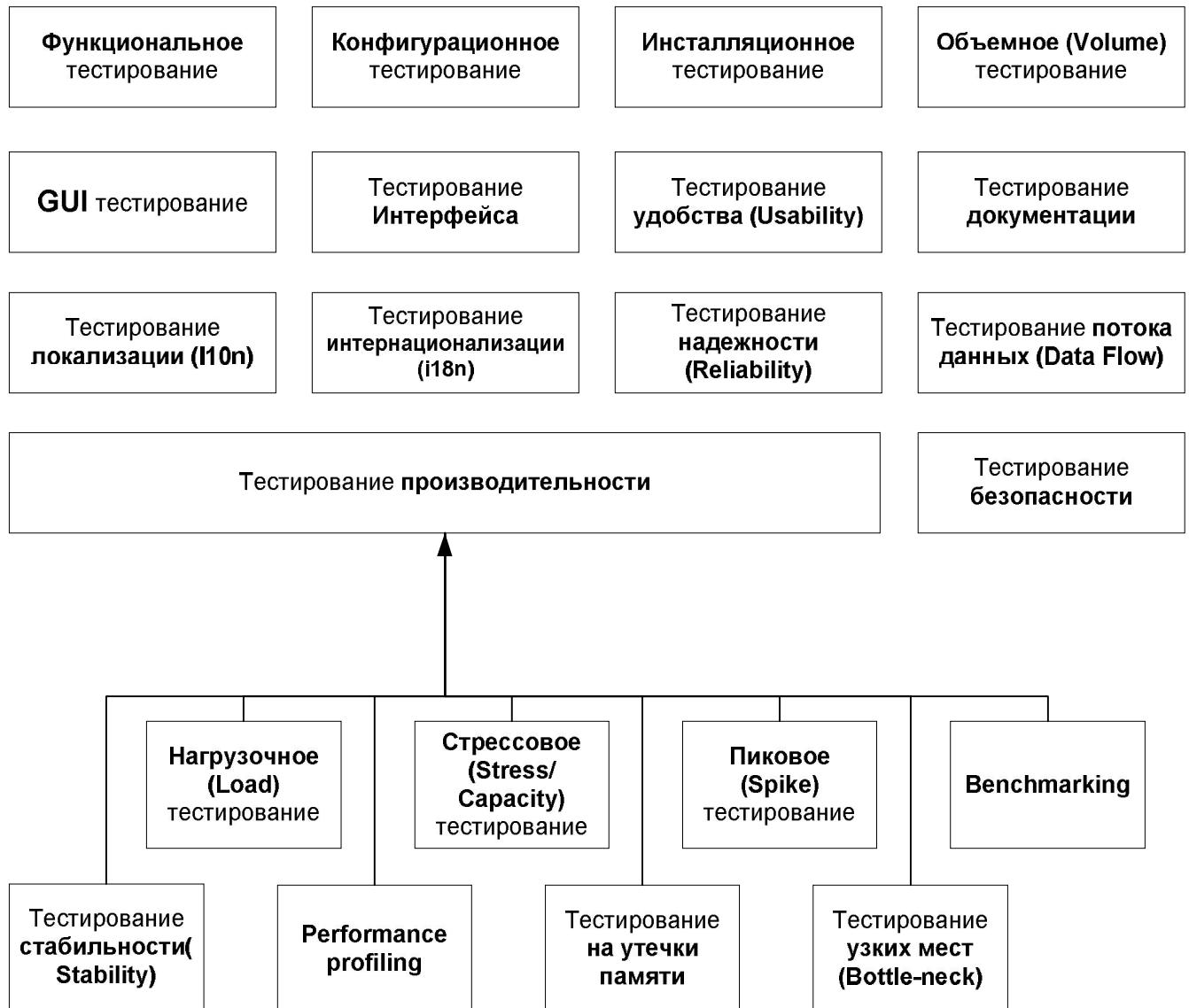
Решение о проведении приемочного тестирования принимается, когда:

- продукт достиг необходимого уровня качества;
- заказчик ознакомлен с Планом Приемочных Работ (Product Acceptance Plan) или иным документом, где описан набор действий, связанных с проведением приемочного тестирования, дата проведения, ответственные и т.д.

## ТИПЫ ТЕСТОВ



## ВИДЫ ТЕСТОВ



## ВИДЫ ТЕСТОВ

- 1) **Функциональное тестирование** Функциональное тестирование проверяет функции и сервисы, которые программа должна выполнять на системном или пользовательском уровне, а так же что продукт НЕ ДОЛЖЕН выполнять. Игнорируются внутренние механизмы системы или компонента и фокусируются на откликах системы (outputs).  
*Тестирование вычисления прибыли в любой день между началом и концом депозита учитывая процентную ставку по депозиту.*
- 2) **GUI Тестирование** Тестирует графические компоненты приложения и их соответствие требованиям ОС. Включает в себя проверку наличия, внешнего вида, поведения и функционирования каждого GUI элемента.  
*Убедиться, что кнопка "Login" располагается по центру экрана, имеет синий цвет (#0000CD), размер 20x15 пикселей.*
- 3) **Тестирование Интерфейса** Проверяет взаимодействие с другими системами или окружением: входные данные, получаемые от этих систем и выходные данные отсылаемые в эти системы. Так же включает проверку формата данных.  
*Проверить, что программа может импортировать данные из документа MS Word.*
- 4) **Объемное (Volume) тестирование** Проверяет систему на большие объемы данных, с которыми она работает.  
Тестируется, что программа может работать с определенным объемом входных, выходных и внутренних данных.  
*Проверить, что программа может импортировать текстовый документ размером до 1Гигабайта.*
- 5) **Инсталляционное тестирование** Проверка установки программы включая любые опции или условия, которые могут повлиять на установку.  
Так же проверяются опции инсталлятора: апгрейд, восстановление и починка, удаление.  
*Проверить, что программа установлена в папку, которая была указана в инсталляторе.*
- 6) **Тестирование потока данных (Data Flow)** Тестирование пути потока данных между модулями системы.  
Проверить, что данные корректно интерпретируются во время их передачи между компонентами системы.  
*Проверить создание пользователя в интернет-магазине. В эту проверку будет входить: Добавление пользователя в базу -> Отсылка активационного письма на указанный адрес -> Проверка активационного номера-> Активация пользователя -> Начисление скидки.*
- 7) **Конфигурационное тестирование** Тестирование продукта на разных конфигурациях программного/ аппаратного (software/hardware) обеспечения.  
Зачастую для этого требуется список самых часто используемых конфигураций.  
Так же может включать в себя проверку переноса (portability) на другие конфигурации.  
*Убедиться, что система работает на Lion OS 10.7.3 и Safari 6.0*
- 8) **Тестирование удобства (Usability)** Проверка удобства использования продукта для конечного пользователя.  
Тестирование «человеческого фактора» – удобство, ясность, простота, легкость, интуитивная понятность, единый стиль, “user-friendly” и т.д.  
*Убедиться, что самые часто используемые функции программы находятся на самых видных местах (верхняя панель программы, первые пункты выпадающих меню).*
- 9) **Тестирование безопасности** Проверка несанкционированного доступа к данным или проведения несанкционированных операций. Тестирование уровней доступа для разных пользователей и/или групп пользователей. Так же включает проверку триальных, ограниченных и истекаемых (trial/limited/expirable) версий продукта.  
*Проверить, что доступ к админ странице сайта есть только у пользователей из группы «admins».*
- 10) **Тестирование надежности (Reliability)** Сбор и сравнение статистических параметров: частота сбоев, время между сбоями, распределение сбоев по severity.  
Тестирование механизмов резервного копирования и восстановления (backup and recovery).  
*Убедиться, MS Word делает резервную копию текущего файла с определенным интервалом и что данные можно восстановить из этого файла после сбоя.*
- 11) **Тестирование интернационализации (i18n)** Проверка интернационализации продукта.  
Тестирование возможности работы продукта на разных платформах (в смысле языка и культуры).  
*Убедиться, что продукт работает на китайской версии Windows.*
- 12) **Тестирование локализации (l10n)** Проверка локализации продукта и его работы на определенной платформе (в смысле языка и культуры).  
Проверка GUI элементов, справок, документации, установка и апгрейд на локализованном окружении.  
*Тестирование арабской версии продукта (вязь, написание справа налево, праздники в календаре и т.д.).*
- 13) **Тестирование документации** Проверка что вся документация описанная в требованиях есть в наличии и доступна пользователям. Документация адекватно описывает возможности и функции программы Ссылки и кросс-указатели(cross-references) присутствуют и работают.  
Документация имеет определенный внешний вид, формат и правописание  
Она опубликована правильным способом, доступна в определенном месте  
*Проверить, что Readme.txt и Release Notes.txt есть в пакете приложения.*

## ВИДЫ ТЕСТОВ

# Тестирование производительности

Проверка производительности системы на соответствие требованиям.

**1) Нагрузочное (Load) тестирование** Тестирование программы в реальных условиях эксплуатации и под реальными нагрузками.

Разнообразные виды нагрузки приложения, но в рамках ожидаемых условий.

*Вход в приложение разного количества пользователей но меньше максимально разрешенных 100.*

**2) Стressовое (Stress/Capacity) тестирование** Тестирование программы в условиях близких к

максимальным и превышающих их. Позволяет убедиться, что производительность системы не падает и система работает при максимальной нагрузке. *Одновременный вход в приложение максимально разрешенного количества пользователей.*

**3) Пиковое (Spike) тестирование** Проверка нагрузкой, которая в несколько раз превышает допустимую.

Позволяет проверить как система реагирует на чрезмерную нагрузку и в какой момент сломается.

*Одновременный логин на сайт тысячи пользователей. Проверить как сайт (сервер) будет работать после такой нагрузки.*

**4) Benchmarking** Тестирование производительности в сравнении с похожими продуктами и/или предыдущими версиями. Позволяет обнаружить ухудшение производительности новых версий.

*Сравнение производительности Windows XP и Windows Vista.*

**5) Тестирование стабильности(Stability)** Позволяет определить доступность программы на ожидаемом временном отрезке.

Позволяет проверить что определенные характеристики продукта (время отклика, время обработки запроса) остаются в допустимых пределах на длительном периоде времени.

*Время логина в программу должно занимать не больше 30 секунд на протяжении 6 месяцев после рестарта.*

**6) Performance profiling** Прямое слежение за характеристиками производительности:

Время выполнения определенной операции, Использование CPU, Использование памяти.

*Приложение должно использовать не больше 20% CPU.*

**7) Тестирование на утечки памяти** Тестирование нацелено на определение ситуаций когда программа не высвобождает память. Это может привести к постепенному заполнению ОП и существенному ухудшению производительности.

*Выполнение определенной операции большое количество раз. Если данная операция имеет проблемы с выделением памяти, то при замере использования ОП эта проблема будет выявлена.*

**7) Тестирование узких мест (Bottle-neck)** Нахождение критических мест в потоке данных, которые замедляют работу системы . Bottle-neck (бутылочное горлышко) – модуль и компонент системы, увеличив производительность которого мы добьемся увеличения производительности программы.

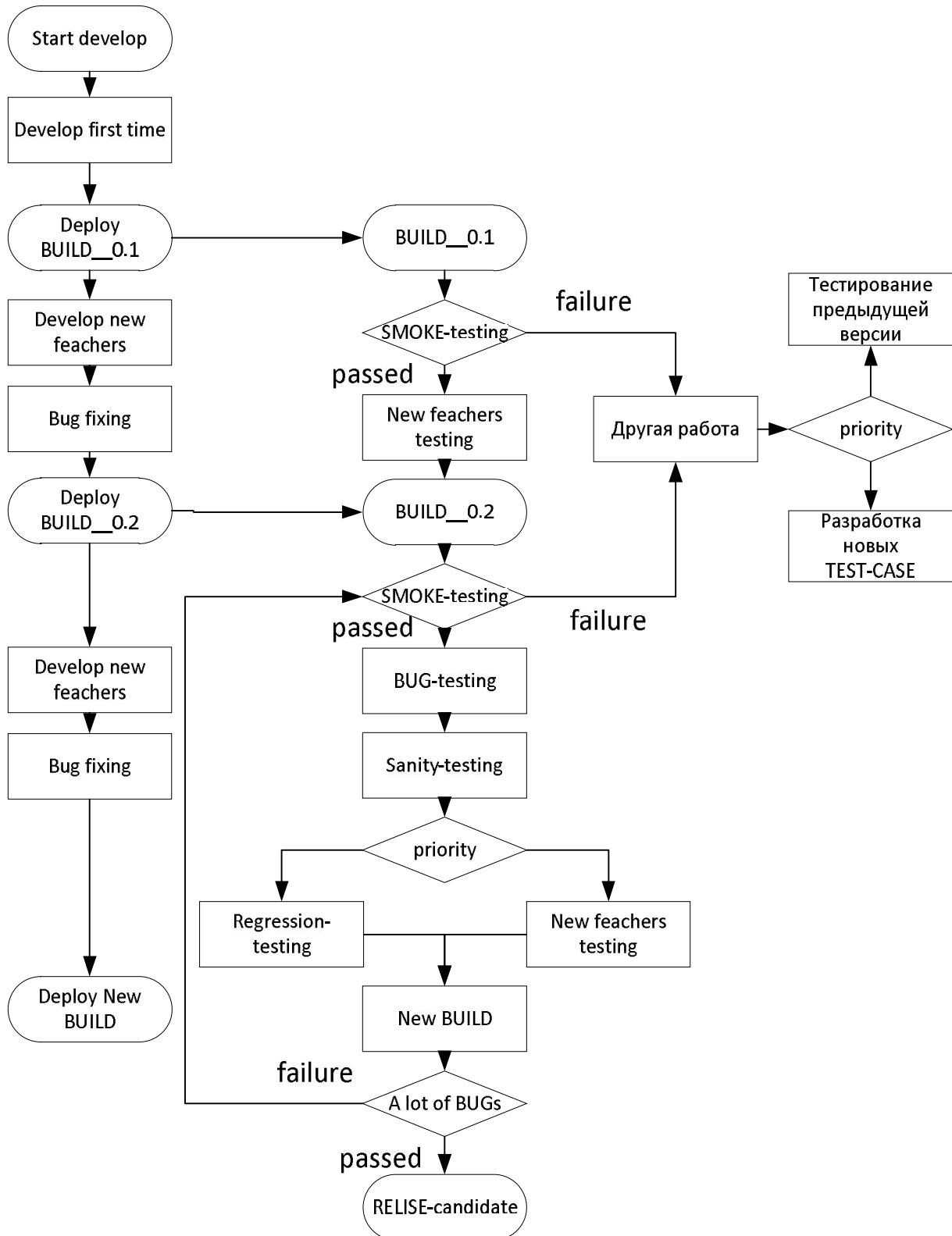
*Модуль записи в БД использует неоправданно сложный запрос, который делает полное сканирование базы.*

**8) Тестирование «условия гонок» (Race conditions)** Тестирование систем, которые позволяют одновременный доступ к ресурсам.

«Условия гонок» - аномальное поведение программы при одновременном возникновении событий и/или одновременном доступе к одному ресурсу.

*Попытка одновременного редактирования одного файла на Sharepoint.*

# ЦИКЛ ТЕСТИРОВАНИЯ



# РОЛИ В ПРОЦЕССЕ УПРАВЛЕНИЯ ДЕФЕКТОМ



**Project manager**

- + Первичная настройка репозитория (роли, код проекта, и т.п)
- + Назначение исполнителей для поиска дефектов, отслеживание выполнения, переназначение
- + Контроль за отложенными дефектами
- + Предварительный анализ дефекта (отклонение, планирование работ по фазам, дупликация) – вместе с разработчиками
- + Определение процесса в рамках которого родился дефект – с участием архитектора, аналитика, ТМ



**Bussines analist**

- + Анализ дефектов
- + Поиск и определение возможных причин дефектов на уровне требований
- + Исправление дефектов в требованиях



**Developer engineer**

- + Исправление дефектов и реализация запросов на изменение
- + Передача сборки на тестирование



**Technical writer**

- + Назначение исполнителей для поиска дефектов в документации, отслеживание выполнения, переназначение
- + Контроль за отложенными дефектами документации
- + Предварительный анализ дефектов документации (отклонение, планирование работ по фазам, дубликаты)
- + Определение процесса в рамках которого родился дефект



**Test manager**

- + Настройка репозитория
  - Система уведомления
  - Настройка динамических списков
  - Ведение номеров сборок
  - Ведение списка сценариев тестирования (test cases) и log folders
  - Связь с системой поддержки требований
  - Дополнительные запросы, отчеты, правила, поля в проекте
- + Регистрация дефектов, полученных от заказчика, от членов проектной команды, не уполномоченных на регистрацию

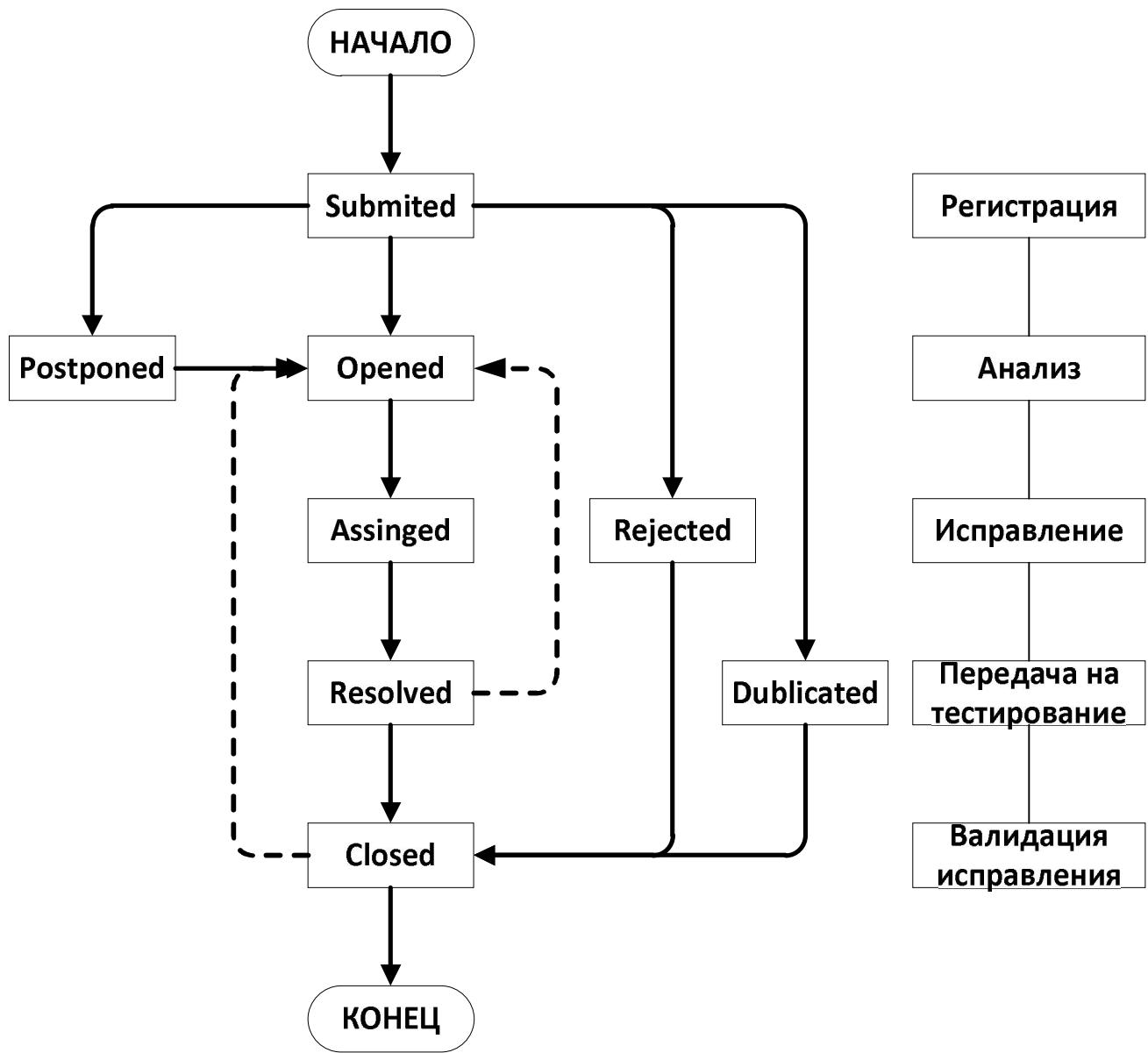


**Quality control engineer**

- + Контроль за соблюдением правил регистрации и обработки дефектов
- + Анализ списков дефектов, диаграмм распределения и трендов для планирования работ и отчетности
- + Контроль за дефектами, обнаруженными вне раунда тестирования -> доработка планов тестирования
- + Обработка отклоненных дефектов
- Контроль за неподтвержденными дубликатами

- + Регистрация дефектов, обнаруженных в раунде тестирования
- + Контроль реализации запросов на изменения и исправления дефектов, включая дубликаты
- + Обработка отклоненных дефектов по запросу ТМ

## ЖИЗНЕНЫЙ ЦИКЛ ДЕФЕКТА





## ЖИЗНЕНЫЙ ЦИКЛ ДЕФЕКТА ПО РОЛЯМ

Customer



Quality control engineer



Project manager



Developer engineer



## ТИПЫ НЕИСПРАВНОСТЕЙ

### Error (ошибка)

несоответствие между вычисляемым, наблюдаемым, или измеренным значением или состоянием и ожидаемым теоретически правильным значением или состоянием.

### Defect (дефект)

проблема внешнего вида программного продукта, поведения, несоответствующего спецификации, отсутствие ожидаемого функционала.

### Fault (неисправность)

неверный шаг, процесс или данные, которые влекут за собой непреднамеренные или непредвиденные действия в программе.

### Failure (отказ, сбой)

неспособность системы или компонента системы выполнять требуемые функции при заданных требованиях к производительности.

### Bug

ошибка в программе, которая заставляет программу выполнять непреднамеренные или непредвиденные действия.

## Ущерб

### Критические (Critical)

— Ошибки, из-за которых возможен отказ от работы с системой и переход на другую.  
(уничтожение пользовательских данных, уничтожение оборудования)

### Блокирующие (Blocker)

— Ошибки, из-за которых дальнейшая работа с системой становится невозможной.

### Важные (Major)

— Из-за таких ошибок система, в целом, работает, но что-то работает не так

### Обычные (Normal)

— Как правило, к этой категории баги относят очень редко. В качестве примера, могу написать что-то вроде не работает кнопка «Запомнить меня» на сайте

### Малозначимые (Minor)

— к таким, как правило, относятся небольшие баги, типа опечаток, «плавания» вёрстки в IE6 на определенной странице в админке и т.п. Редко исправляются по одному, собираются в несколько десятков/сотен/тысяч в зависимости от продукта и фиксируются «пачкой»

## ПРИОРИТЕТ

### FIX IN RELEASE

— Пофиксить в новой версии продукта. Как правило, относится к багам, обнаруженным в процессе тестирования нового функционала

### MUST FIX

— Пофиксить как можно быстрее. Как правило включает в себя блокирующие баги, которые должны быть исправлены в специальном сервис паке, до выхода новой версии

### FIX IF TIME

— «Пофиксить, если есть время» — к этой категории как правило относятся мелкие баги

### NEVER FIX

— «Не фиксить никогда». Например, какая-то фича будет удалена из следующей версии продукта, либо найдена в продукте, который уже более не поддерживается, или его поддержка прекратится в ближайшее время

## ОПРЕДЕЛЕНИЕ ДЕФЕКТА

**Дефект** есть несоответствие между фактическими и требуемыми характеристиками объекта тестирования

**Дефект** есть несоответствие фактического поведения системы разумным ожиданиям пользователя

### ДЕФЕКТ НАЙДЕН

Определены наиболее простые и наиболее общие условия возникновения ошибки

Найдены другие пути возникновения той же самой ошибки

Установлены связанные проблемы

Найдены последствия, к которым ошибка может привести

Проверена устойчивость воспроизведения дефект

### ОТЧЕТ О ДЕФЕКТЕ

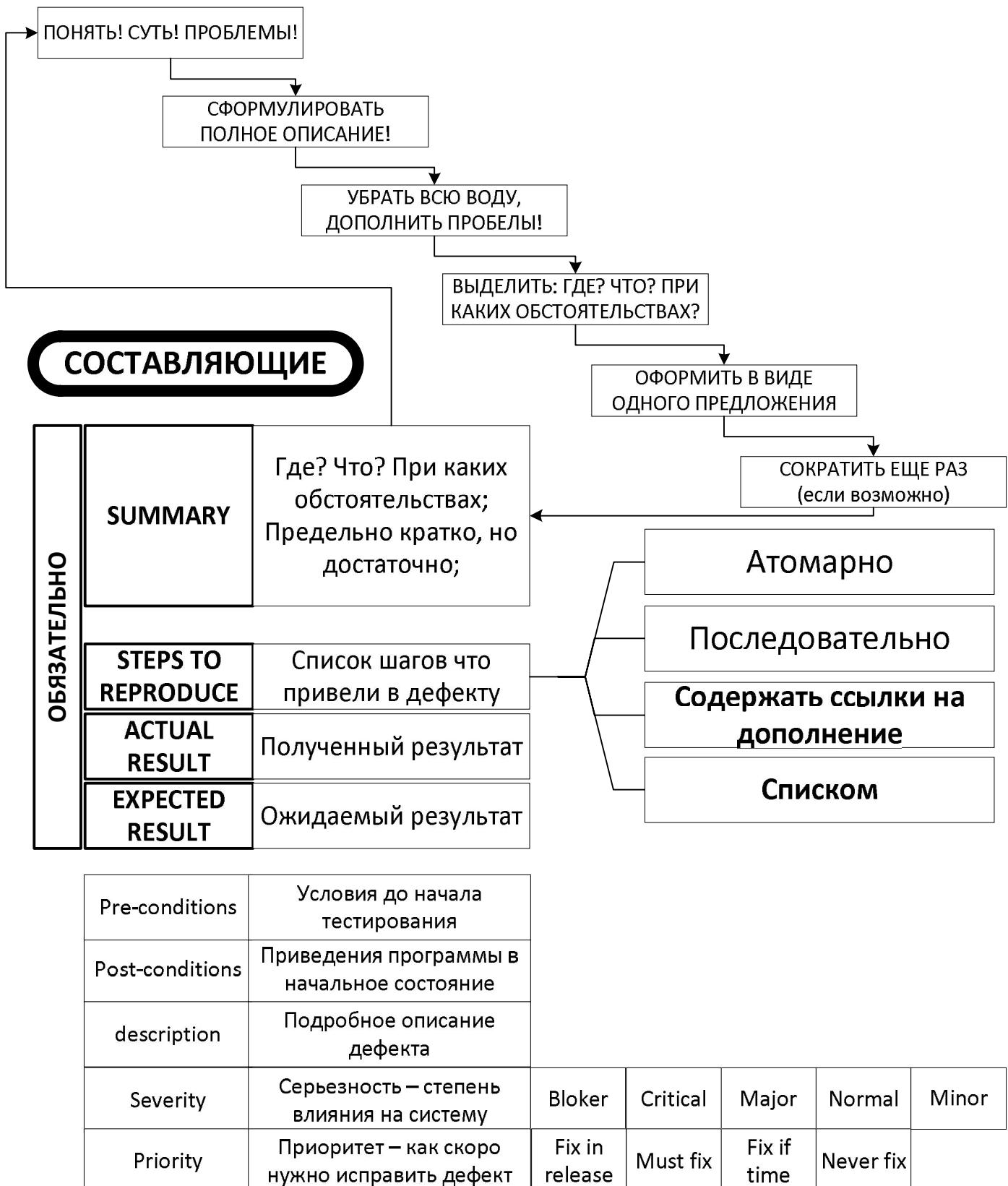
**Наблюдаемый дефект** - это расхождение, неувязка между ожидаемым и полученным результатом, неверное поведение программного продукта.

Отказ это симптом = внешнее проявление внутреннего изъяна, наблюдаемое при некоторых условиях  
Тестеры (и пользователи) наталкиваются на отказы/сбои, иначе: наблюдают симптомы

**Внутренний дефект, или изъян** – это причина отказа, то, что привело к отказу; как правило, дефект в коде программы, но и: дефект в дизайне.

Разработчики находят и исправляют внутренние дефекты (ставят диагноз и проводят «лечение»)  
Изъян кода может не приводить к отказу, т.е. может быть не замечен тестировщиком  
Один изъян может приводить к нескольким отказам в разных частях системы

# ОТЧЕТ О ДЕФЕКТЕ

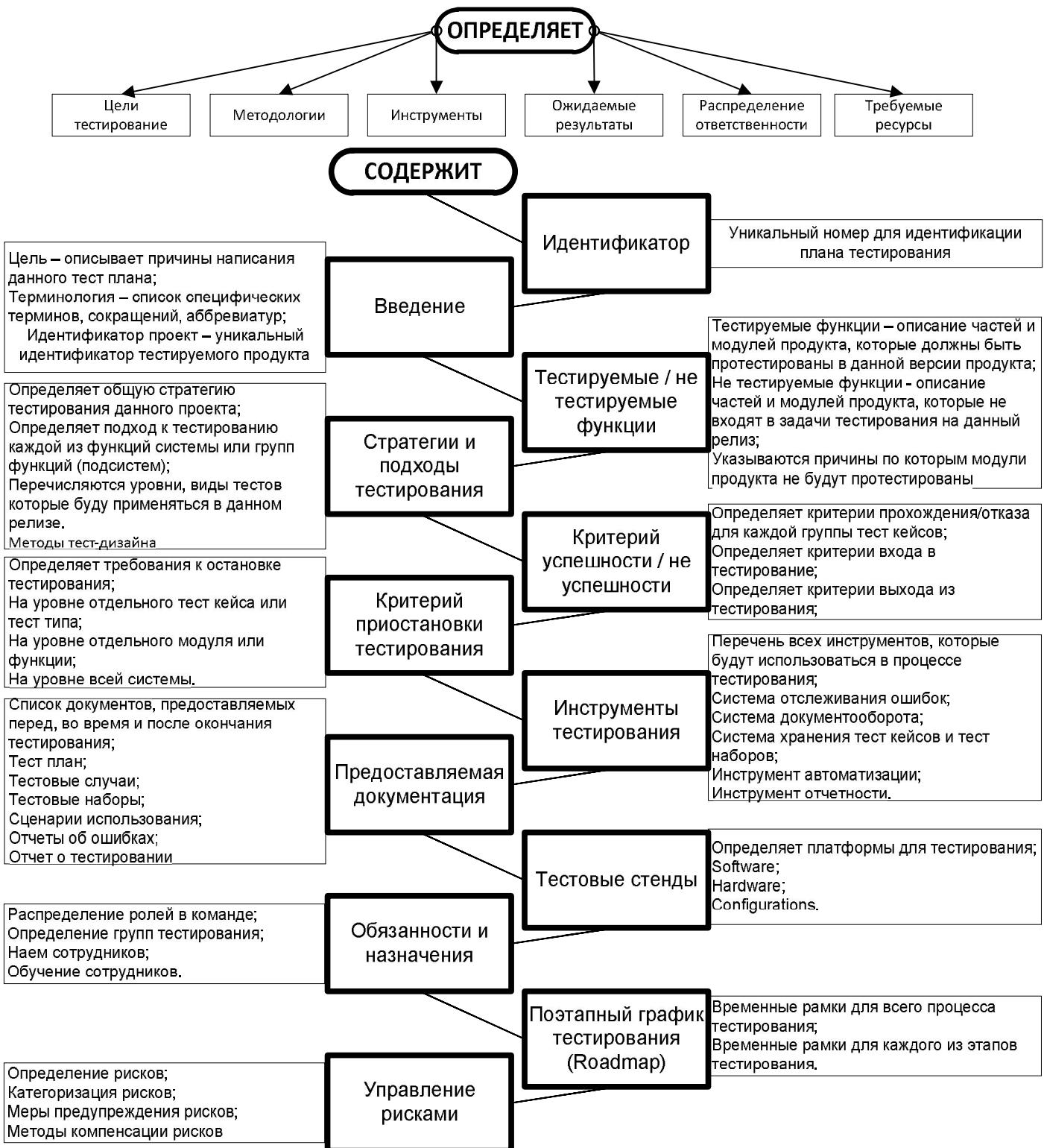


# ХОРОШИЙ ОТЧЕТ О ДЕФЕКТЕ

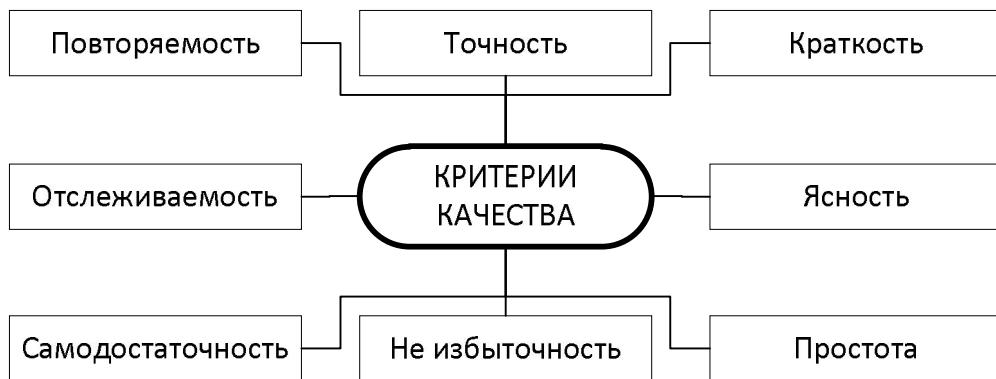
ПРОСТОТА	ПОЛНОТА	ОБЪЕКТИВНОСТЬ	НЕЙТРАЛЬНОСТЬ	НИЧЕГО ЛИШНЕГО
<ul style="list-style-type: none"> <li>+ Простые грамматические структуры</li> <li>+ однозначные выражения</li> <li>+ короткие и ясные фразы</li> <li>+ СПИСОК (идеальный вариант)</li> </ul>	<ul style="list-style-type: none"> <li>+ Только необходимую информацию для понимания дефекта</li> <li>- без лишних уточнений</li> </ul>	<ul style="list-style-type: none"> <li>+ предоставить объективные данные для принятия решений</li> <li>+ не указывать на виновника</li> </ul>	<ul style="list-style-type: none"> <li>+ суть излагать ясным и деловым языком</li> <li>- без эмоций</li> <li>- без юмора и сарказма</li> </ul>	Только то что необходимо

СНИМКИ ЭКРАНА	КРАТКОЕ ОПИСАНИЕ	ПОДРОБНОЕ ОПИСАНИЕ
<ul style="list-style-type: none"> <li>+ Доказательство работы тестировщика и программиста.</li> <li>+ Снимок лишь части экрана - проблемной зоны (лучше в формате JPEG или PNG)</li> <li>+ Снимок нужно прикрепить к отчёту о дефекте.</li> <li>+ Название адекватно – названию дефекта</li> </ul>	<p>+ Краткое (в одну строку) описание проблемы.</p> <p>+ Используется ПМ-ом при анализе списка неисправленных дефектов.</p> <p>+ Позволит выделить и подробнее рассмотреть только значимые дефекты.</p> <p>Оно должно включать :</p> <ul style="list-style-type: none"> <li>- Краткое, но достаточно точное описание, позволяющее понять суть дефекта.</li> <li>- Краткое указание на область и условия проявления дефекта (насколько проявление дефекта зависит от условий выполнения программы)</li> <li>- Указание на серьезность дефекта (помогающее представить последствия его наличия в продукте)</li> </ul>	<p>Подробное описание сути проблемы.</p> <ul style="list-style-type: none"> <li>- Используется ПМ-ом и разработчиком при углубленном изучении сути дефекта.</li> <li>- Должно быть самодостаточным, концентрирующим в одном месте максимум информации о дефекте</li> </ul> <ul style="list-style-type: none"> <li>- Перечислите все переменные окружения (config и т.п.).</li> <li>- Если ошибка трудновоспроизводима (требует специальных условий - специфических входных данных, предварительных действий, напрямую не связанных с этой ошибкой т.п.), то четко опишите эти условия.</li> <li>- Правильное подробное описание не вызывает вопросов.</li> </ul>

ВОСПРОИЗВОДИМОСТЬ		
<ul style="list-style-type: none"> <li>+ Всегда описывайте воспроизводимость дефекта.</li> <li>- Никогда не говорите «да», пока не поймете, как добиться повторения этой ошибки. (Научитесь воспроизводить ошибку до составления отчета.)</li> <li>- Если после неоднократных попыток воспроизвести ошибку не удается, то напишите «нет» и объясните, какие действия вы предпринимали для ее воспроизведения.</li> </ul>	<ul style="list-style-type: none"> <li>+ Всегда описывайте воспроизводимость дефекта.</li> <li>- Если дефект проявляется нерегулярно и вы все еще не понимаете, почему, то напишите «иногда» и дайте поясните.</li> <li>- Не всегда возможно воспроизвести ошибку. Например: дефект, описанный пользователем, проявляющийся в специфической ситуации, которую трудно воспроизвести.</li> </ul> <p><b>Обязательно!</b> Тестировщик должен воспроизвести ошибку в присутствии разработчика, если разработчик говорит, что не может ее повторить.</p>	<p><b>Как воспроизвести дефект.</b></p> <ul style="list-style-type: none"> <li>+ Опишите шаги для воспроизведения этого дефекта.</li> <li>- Начните описание с известного места (например, с запуска программы или открытия входного документа) и</li> <li>- Затем опишите каждый шаг до проявления ошибки.</li> <li>- <b>НУМЕРУЙТЕ ШАГИ.</b></li> <li>- Отделяйте шаги один от другого.</li> </ul> <ul style="list-style-type: none"> <li>+ Опишите ожидаемое и фактическое поведение.</li> <li>- Разница между ними и будет сущностью дефекта.)</li> </ul>



## TEST CASE



Header		
<b>Identifier</b>	Уникальный идентификатор для каждого теста	
<b>Owner</b>	Составитель или исполнитель	
<b>Date</b>	дата написания или исполнения	
<b>Requirement</b>	Необходимые условия	
<b>Configuration</b>	Конфигурация на которой производились испытания	
Details		
<b>Purpose</b>	Цель	
<b>Reason</b>	Причина проведения	
<b>Complexity</b>	Сложность проведения	
<b>Estimated time</b>	Предположительное необходимое время	
<b>Notes</b>	Заметки (разное)	
<b>Dependencies</b>	Зависимости от разных условий	
Scenario		
<b>Initialization</b>	Шаги перед началом работы	
Step	Action	Expected Result
№ шага	действие	ожидаемый результат от действия
<b>Finalization</b>	шаги восстановления системы в исходное состояние	

## TEST DESING



Набор входных данных, условий выполнения и ожидаемых результатов, разработанных с целью проверки соответствия **Test designer** тестируемого продукта выдвигаемым к нему требованиям.

**Анализ Границных  
Значений (Boundary Value  
Analysis - BVA)**

Метод проверки  
переменных программы  
на их границах

**Эквивалентное Разделение  
(Equivalence Partitioning -  
EP)**

Метод сокращения  
числа тестов путем  
выбора одного теста из  
эквивалентного набора

**Предугадывание ошибки  
(Error Guessing - EG)**

Проверка важных  
проблем, которые могут  
возникнуть

**Причина / Следствие  
(Cause/Effect - CE)**

Проверка продукта по  
наиболее частым и  
важным сценариям  
использования – use  
cases

**Ичерпывающее  
тестирование (Exhaustive  
Testing - ET)**

Проверить все  
возможные комбинации  
входных значений, и в  
принципе, это должно  
найти все проблемы

## Задачи тест дизайна

### 1. Планирование

### 2. Тест дизайн

Составление тест плана

**Анализ рисков**

**Написание тест кейсов**

Описание процесса тестирования

**Создание приемочных проверок**

Составление графика работ

**Составление списка функций продукта**

Распределение  
ресурсов

**Расстановка приоритетов тестирования**

**Анализ требований**

**Анализ жалоб пользователей**

**Построение таблиц принятия решений**

-----Анализ тестового покрытия-----

**Исследовательское тестирование**

Анализ эффективности тестирования

**Приемочное тестирование**

Анализ найденных ошибок

**Регрессионное тестирование**

Обработка ошибок от пользователей

**Тестирование нового функционала**

Составление отчетов  
по тестированию

**Заведение отчетов об ошибках**

### 4. Анализ результатов

### 3. Выполнение тестов

## ИСХОДНЫЕ ДАННЫЕ

A

1 2 3 4 5

B

6 7 8 9 10

C

11

## Границные условия

### Границные условия

- + Границные значения – это значения, на которых программа меняет свое поведение;
- + Границные значения – это границы классов эквивалентности;
- + Формула проверки граничных значений: {BV-1, BV, BV+1}.

границные условия

приграничные условия

A

1

5

B

6

10

C

11

0 1 2 4 5 6

5 6 10 11

10 11 12

## Классы эквивалентности

A

1 2 3 4 5

B

6 7 8 9 10

C

11

### Классы

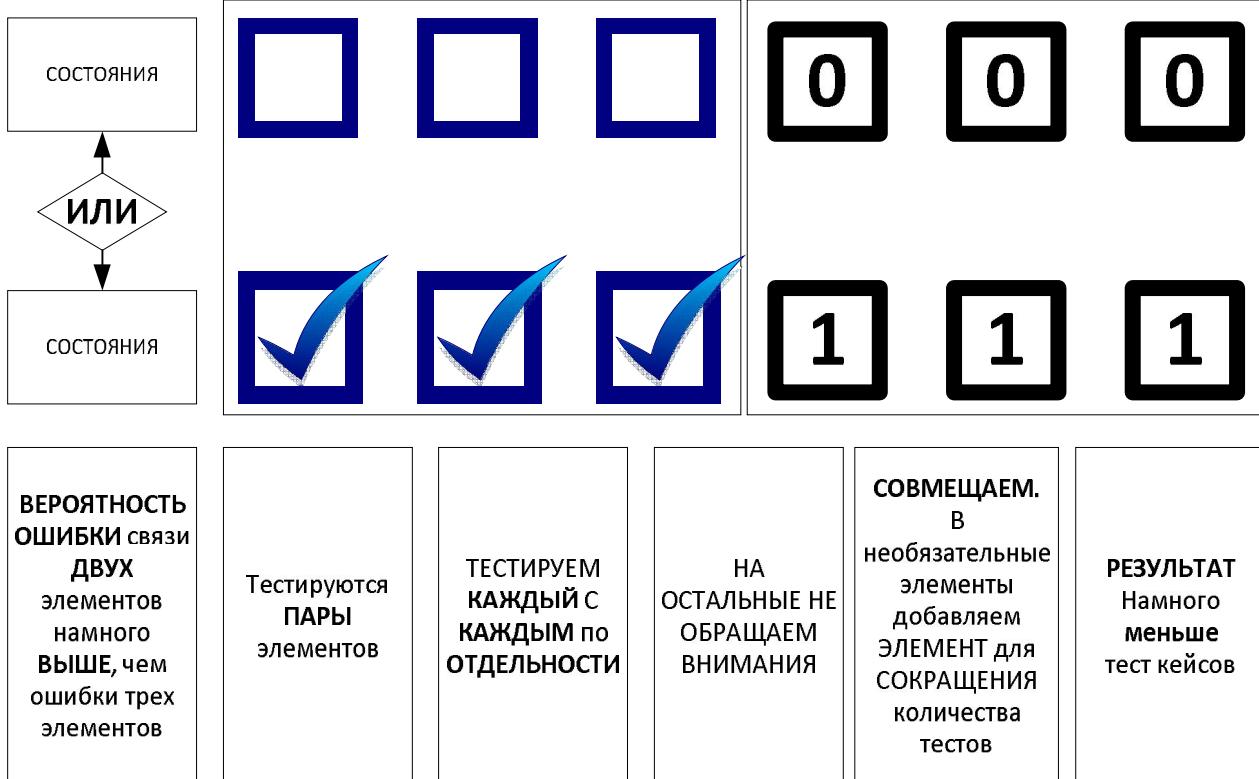
### эквивалентности

- + Класс эквивалентности — это класс, в рамках которого все тесты являются эквивалентными;
- + Эквивалентные тесты — это тесты, которые приводят к одному и тому же результату.

## МЕТОД (ВСЕХ) ПАР

ОПЦИИ

ТОЖЕ САМОЕ, Но другой вид



ВСЕ ВОЗМОЖНЫЕ ТЕСТ-КЕЙСЫ

0	0	0
1	0	0
0	1	0
0	0	1
0	1	1
1	0	1
1	1	0
1	1	1

МЕТОД ВСЕХ ПАР

РЕЗУЛЬТАТ

ручной

программный

0	0	0
1	1	0
1	0	1
0	1	1

## МЕТОД (ВСЕХ) ПАР

ПЕРЕБОР всех возможных вариантов

ПЕРЕБОР всех ПАР по вариантам

0 0 0	0 0 ?	? 0 0	0 ? 0
1 0 0	1 0 ?	? 0 0	1 ? 0
0 1 0	0 1 ?	? 1 0	0 ? 0
0 0 1	0 0 ?	? 0 1	0 ? 1
0 1 1	0 1 ?	? 1 1	0 ? 1
1 0 1	1 0 ?	? 0 1	1 ? 1
1 1 0	1 1 ?	? 1 0	1 ? 0
1 1 1	1 1 ?	? 1 1	1 ? 1

Убираем дубликаты

0 0 ?	? 0 0	0 ? 0
1 1 ?	? 1 0	1 ? 0
1 0 ?	? 0 1	1 ? 1
0 1 ?	? 1 1	0 ? 1

← СОВМЕЩАЕМ

## ТЕСТОВОЕ ПОКРЫТИЕ

Тестовое покрытия – одна из важнейших метрик оценки качества тестирования, представляющая из себя плотность покрытия тестами требований либо исполняемого кода.

### Покрытие требований (Requirements Coverage)

$$T_{cov} = (L_{cov}/L_{total}) * 100\%$$

где:

$T_{cov}$  - тестовое покрытие

$L_{cov}$  - количество требований, проверяемых тест кейсами

$L_{total}$  - общее количество требований

Оценка покрытия тестами функциональных и нефункциональных требований к продукту путем построения матриц трассировки (traceability matrix)

Требование 1

Test case 1.1

Test case 1.2

Test case 1.3

Test case 1.4

Требование 2

Test case 2.1

Test case 2.2

Test case 2.3

Test case 2.4

### Покрытие кода (Code Coverage)

$$T_{cov} = (L_{tc}/L_{code}) * 100\%$$

где:

$T_{cov}$  - тестовое покрытие

$L_{tc}$  - кол-ва строк кода, покрытых тестами

$L_{code}$  - общее кол-во строк кода.

Оценка покрытия исполняемого кода тестами, путем отслеживания непроверенных в процессе тестирования частей программного обеспечения.

Функция  
(метод) 1

Test case 1.1

Test case 1.2

Test case 1.3

Test case 1.4

Функция  
(метод) 2

Test case 2.1

Test case 2.2

Test case 2.3

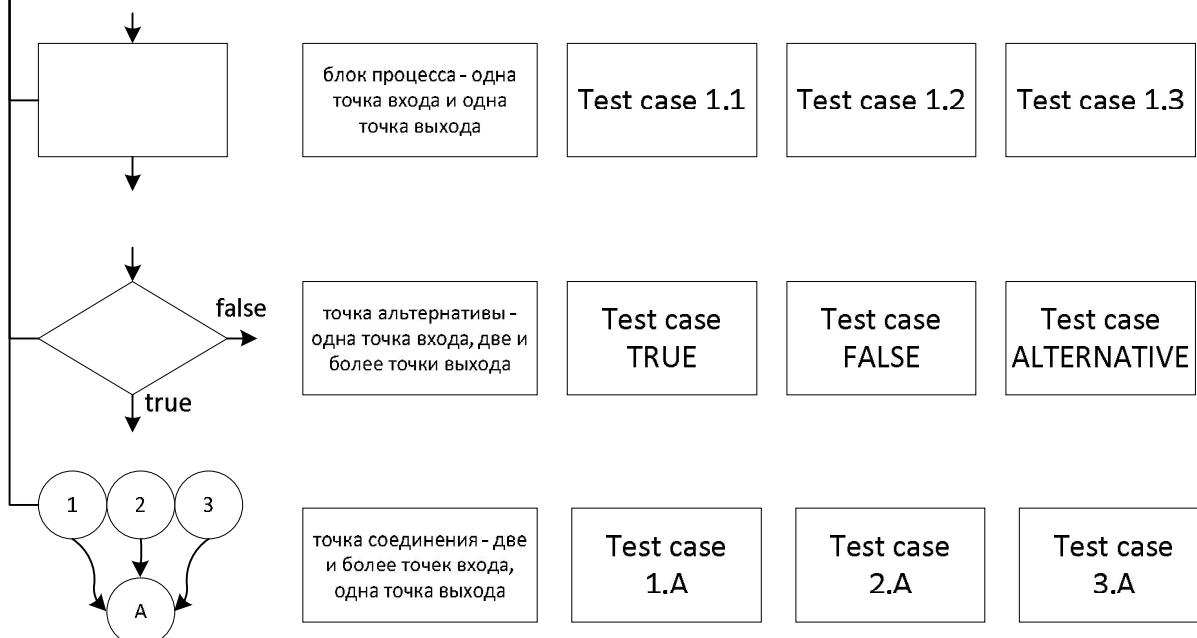
Test case 2.4

## ТЕСТОВОЕ ПОКРЫТИЕ

### Покрытие на базе анализа потока управления

Оценка покрытия основанная на определении путей выполнения кода программного модуля и создания выполняемых тест кейсов для покрытия этих путей.

Фундаментом для тестирования потоков управления является построение графов потоков управления (Control Flow Graph)



### Уровни тестового покрытия

Уровень	Название	Краткое описание
Уровень 0	--	“Тестируй все что протестируешь, пользователи протестируют остальное” На английском языке это звучит намного элегантнее: “Test whatever you test, users will test the rest”
Уровень 1	Покрытие операторов	Каждый оператор должен быть выполнен как минимум один раз.
Уровень 2	Покрытие альтернатив [2] / Покрытие ветвей	Каждый узел с ветвлением (альтернатива) выполнен как минимум один раз.
Уровень 3	Покрытие условий	Каждое условие, имеющее TRUE и FALSE на выходе, выполнено как минимум один раз.
Уровень 4	Покрытие условий альтернатив	Тестовые случаи создаются для каждого условия и альтернативы
Уровень 5	Покрытие множественных условий	Достигается покрытие альтернатив, условий и условий альтернатив (Уровни 2, 3 и 4)
Уровень 6	“Покрытие бесконечного числа путей”	Если, в случае зацикливания, количество путей становится бесконечным, допускается существенное их сокращение, ограничивая количество циклов выполнения, для уменьшения количества тестовых случаев.
Уровень 7	Покрытие путей	Все пути должны быть проверены

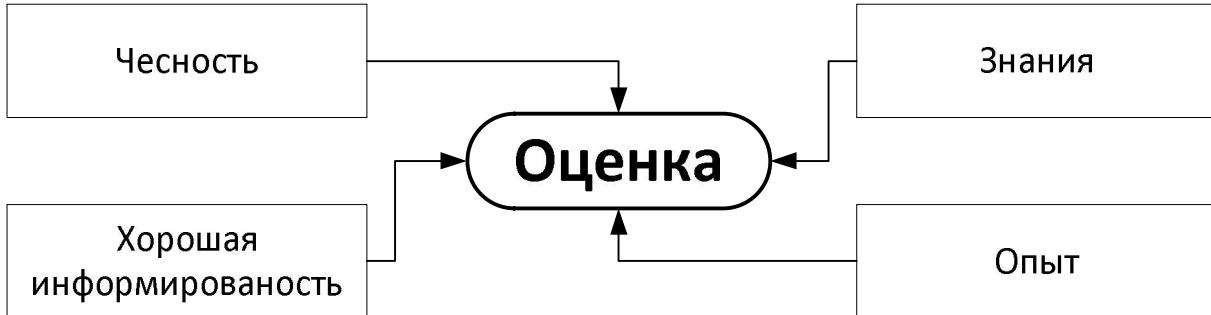
## МАТРИЦА ПОКРЫТИЯ

Таблица, содержащая  
отношение требований к  
подготовленным тестовым  
сценариям (Test Cases)

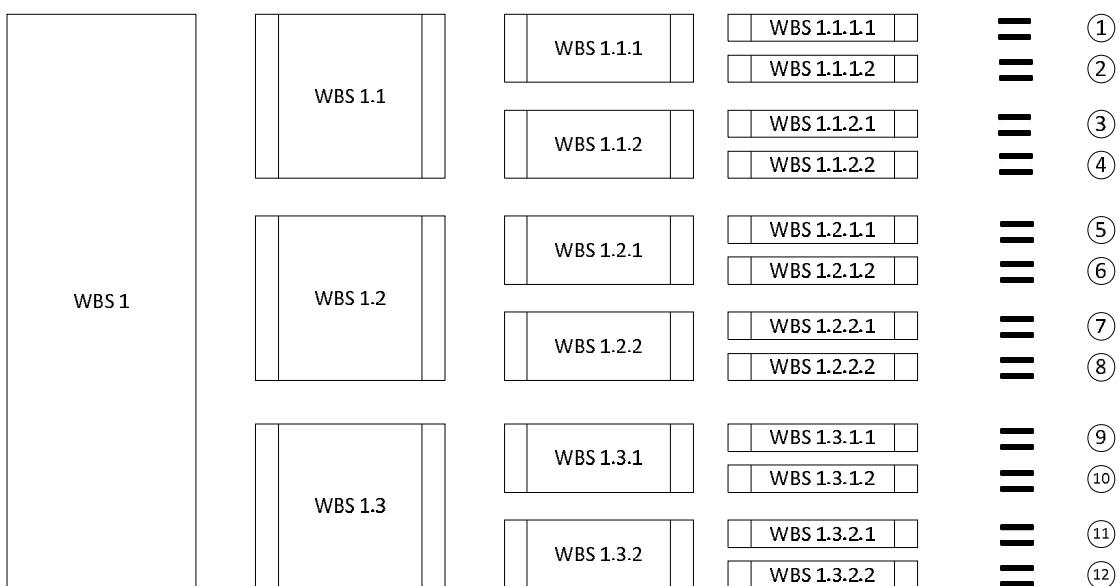
В заголовках колонок –  
требования, в заголовках  
строк – тестовые сценарии.  
На пересечении – отметка  
покрытия

		Раздел «основное»	Раздел «второстепенное»	Раздел «интерфейс»			
		Test case 1	Test case 2	Test case 3	Test case 4	Test case 5	Test case 6
Требование 1		X					
Требование 2			X	X			
Функция (метод) 1					X		X
Функция (метод) 2						X	
Путь данных 1	Функция (метод) 1		X	X	X		
Путь данных 2	Функция (метод) 2	X	X				X
	Функция (метод) 3						

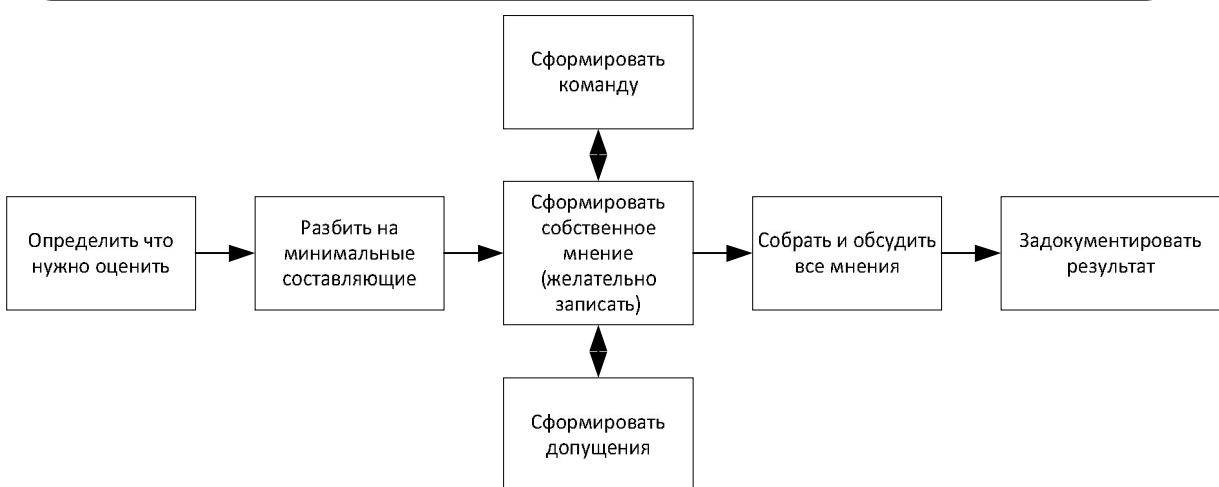
## ESTIMATE TIME



## WORK BREAKDOWN STRUCTURE



## АЛГОРИТМ ОЦЕНКИ ВРЕМЕНИ



# ПЕРВЫЙ ДЕНЬ

1

Знакомство с командой

2

Узнать Ментора (куратора)

3

Обсудить с ментором задачи на  
ближайшее время

4

Изучение продукта – ищем всю  
возможную информацию о  
продукте и изучаем её

5

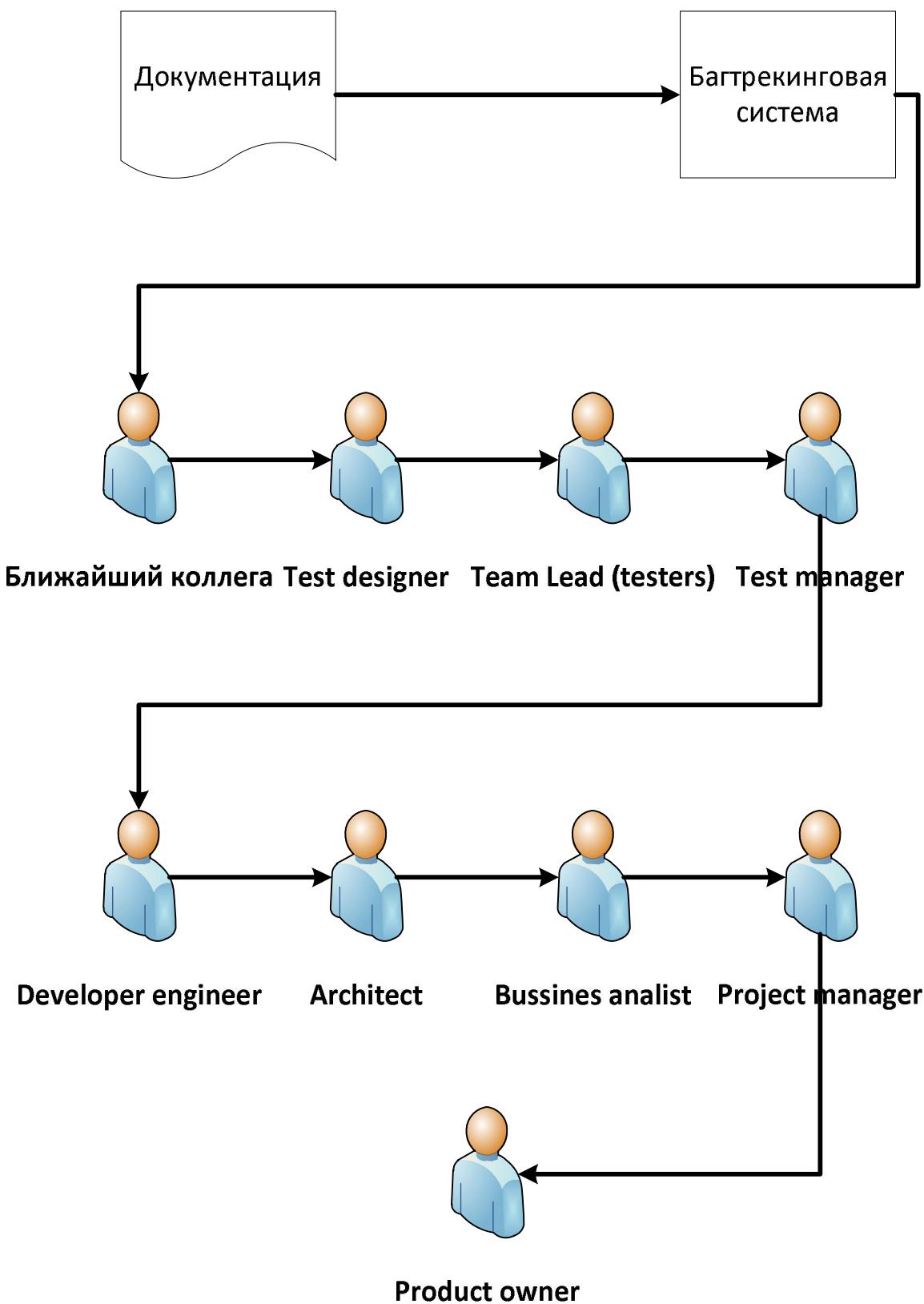
Изучение рабочей зоны

6

Отчет

## ПЕРВЫЙ ДЕНЬ

### ПОИСК ИНФОРМАЦИИ О ПРОДУКТЕ



## BUILDS AND VERSIONS

ONE DEVELOPER and VERSIONS



**Developer engineer**

MANY DEVELOPER and SYSTEM VERSION CONTROL



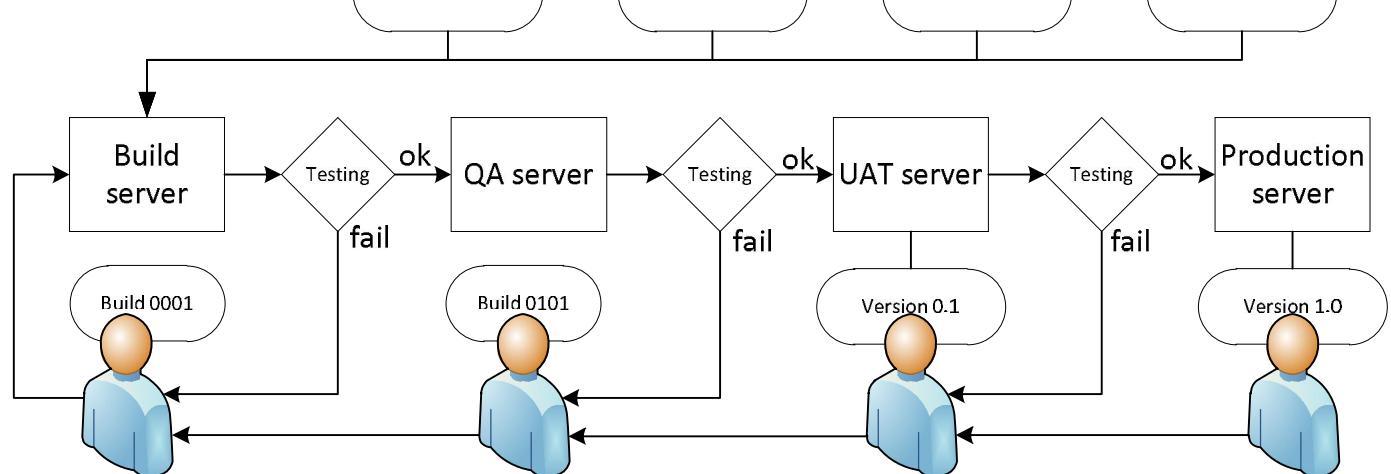
**Developer engineer**



**Developer engineer**



**Developer engineer**



**Developer engineer**

**Quality control engineer**

**Product owner**

**Customer**

## ОСОБОЕ ВНИМАНИЕ

### ЛОГИЧЕСКИЕ УТВЕРЖДЕНИЯ

предыдущая  
включая заканчивая

входит  
исключая  
меньше  
или от  
начиная  
до  
больше

за

входя дважды

следующая

**Не отрицая диапазон**

### ЗАПРЕЩЕННЫЕ К ИСПОЛЬЗОВАНИЮ

немедленно  
качественно

удачно  
хорошо  
плохо  
Быстро  
Должно

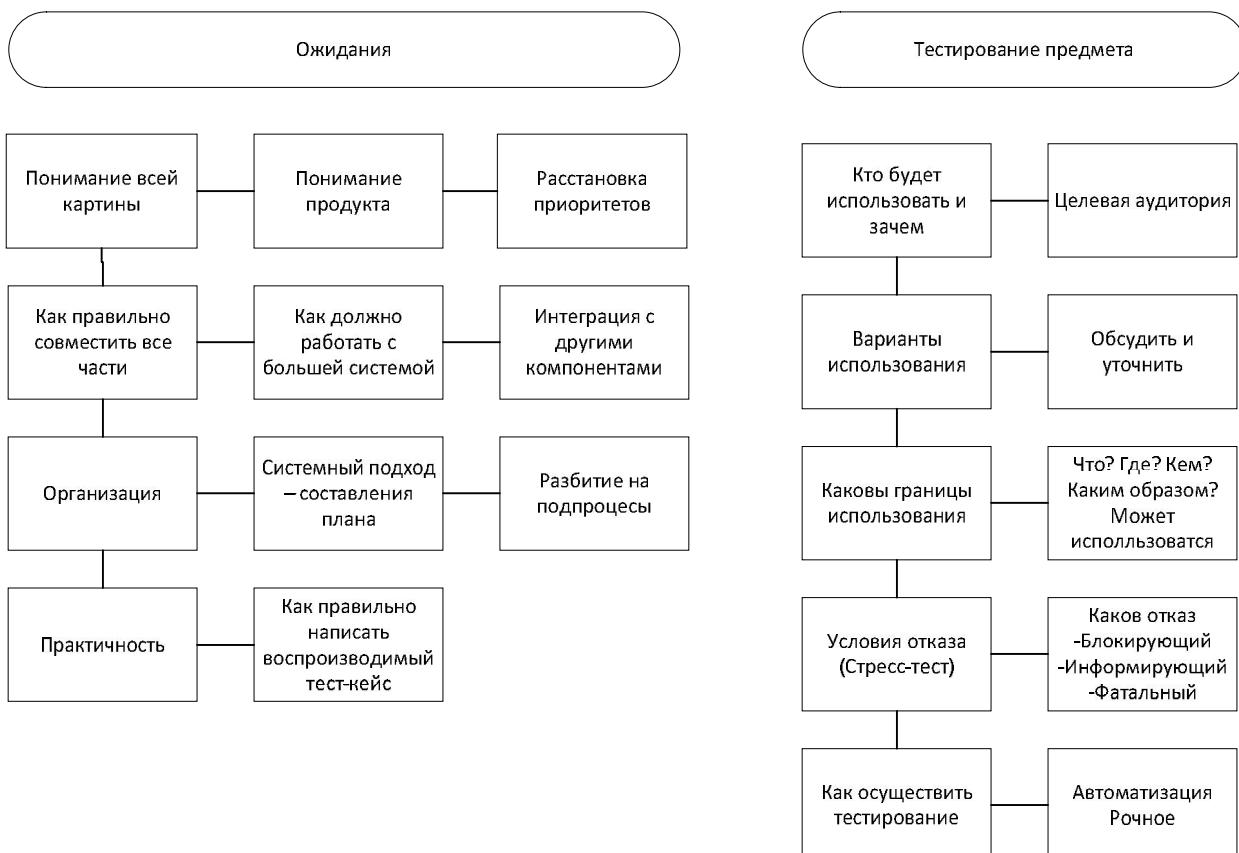
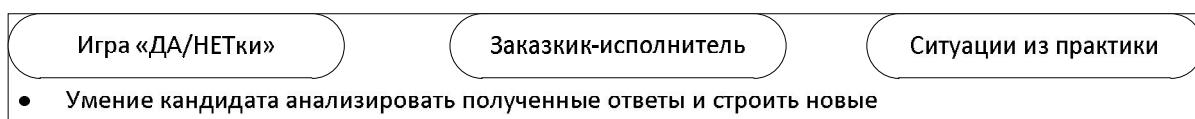
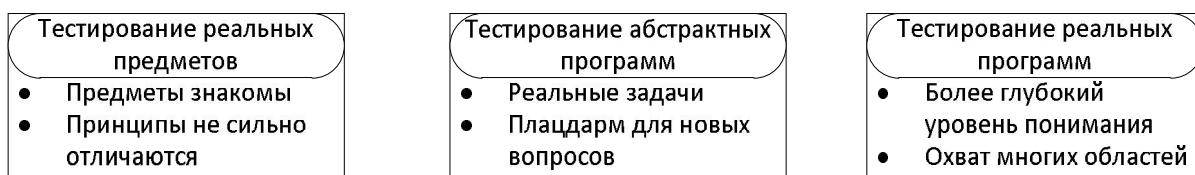
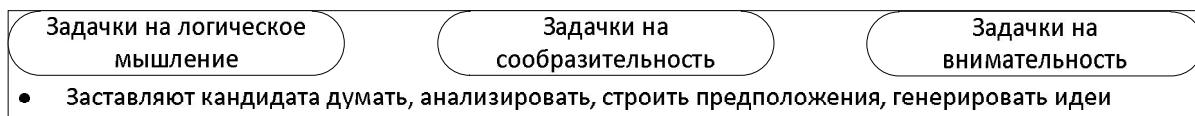
**красиво  
валидно  
нормально**

разный регистр  
целые цифры  
НОЛЛЬ поле  
пробелы ввод границы  
кодировка вывод  
буквы *NUL*  
спец символы дробные  
пустое  
комбинации

### ERROR GUESSING

**Ограничения**

# СОБЕСЕДОВАНИЕ



# СОБЕСЕДОВАНИЕ

Поиск и устранение неисправностей



Информация о проекте



## КАЧЕСТВА ТЕСТИРОВЩИКА

### **Воспитанность**

- обладание тактом и дипломатичностью

### **Грамотность -**

Умение устно и письменно излагать проблему

### **Понимание -**

Способность представить себя на месте другого

### **Наблюдательность и внимательность -**

умение замечать неточности и противоречия

### **Логическое мышление**

### **Креативность, инициативность, азарт**

### **Антилогическое мышление**

– умение мыслить глубже, шире и не так как того требует логика

### **Критическое мышление**

– умение во всем находить недостатки

### **Ясновиденье**

- умение предвидеть где находятся ошибки

### **Многозадачность -**

Умение одновременно выполнять несколько задач

### **Таймменеджмент -**

Умение распределять свое время и работать по расписанию

Склонность

### **экспериментировать**

-

## ВОПРОСЫ НА СОБЕСЕДОВАНИИ