Учебный Центр компьютерного обучения «СПЕЦИАЛИСТ» при МГТУ им. Н.Э.Баумана

Тестирование ПО.

www.specialist.ru

Знакомство

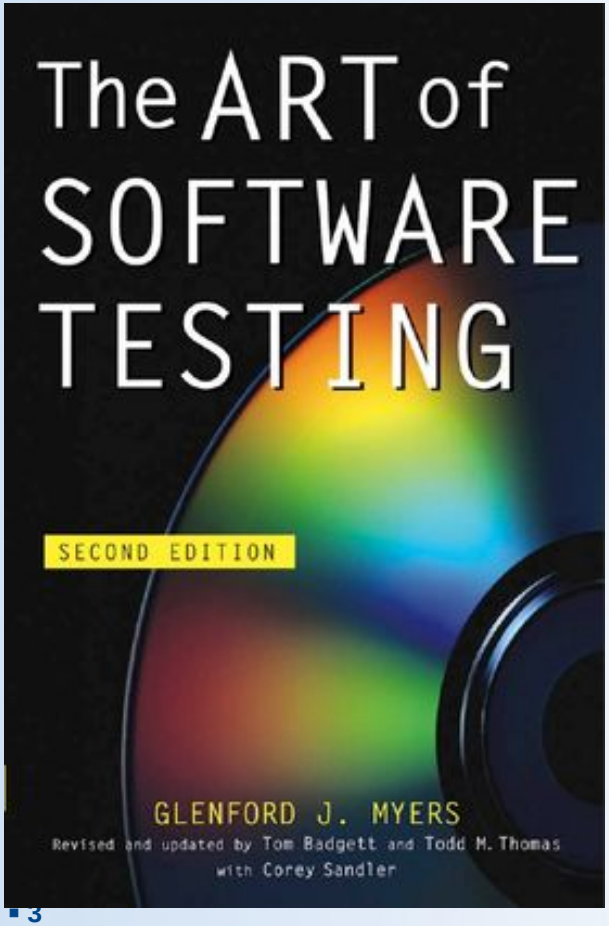
Ушакова Елена Сергеевна –

Руководитель проекта, ведущий аналитик, тест-дизайнер

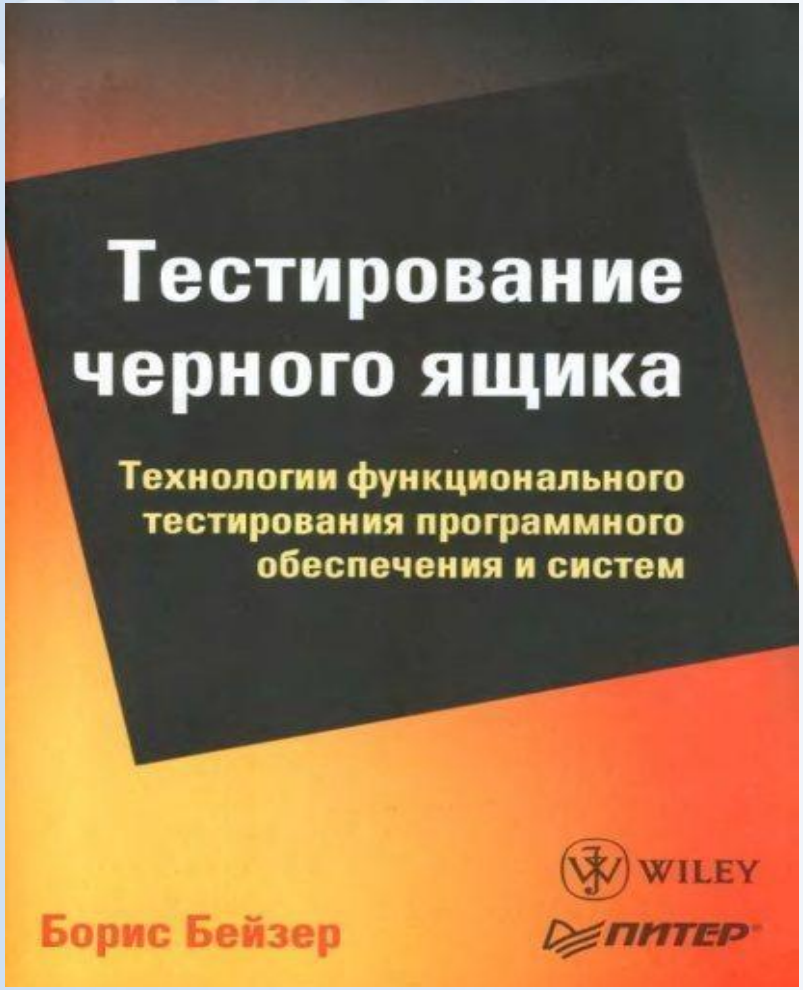
e-mail:eushakova@specialist.ru

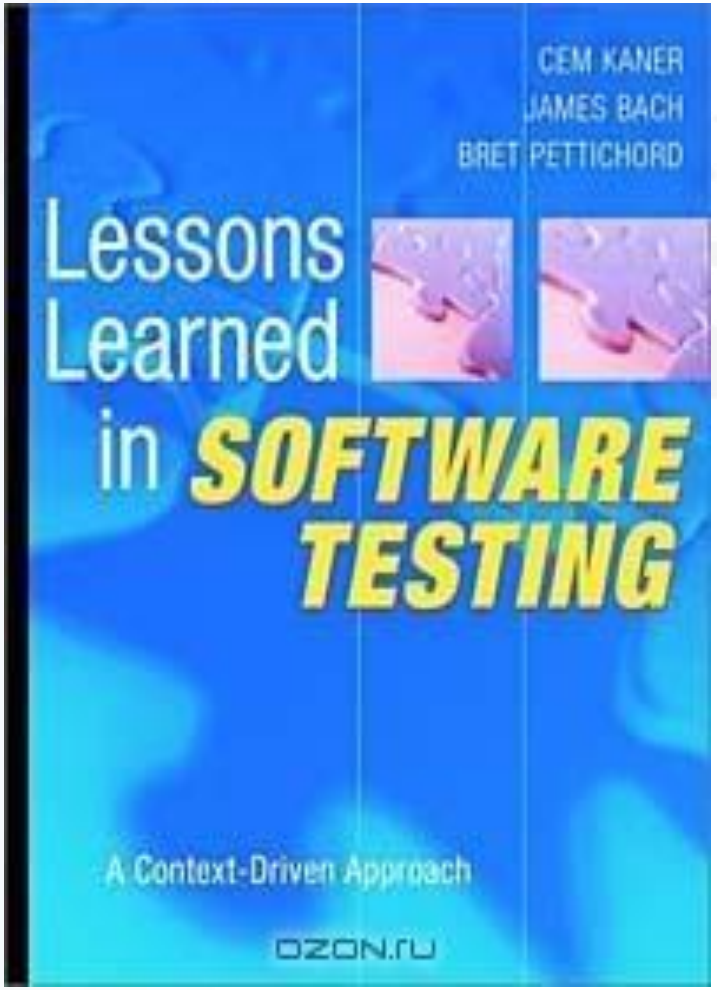
Литература

Майерс Г. Искусство тестирования программ



Тамре Л. Введение в тестирование программного обеспечения





Модуль 1

- Понятие качества ПО.
- Стандарты качества ПО. Атрибуты и характеристики качества ПО.
- Основные определения тестирования.
- Цели и задачи процесса тестирования.
- Полный цикл тестирования. Фазы тестирования.

Что такое тестирование?

Зачем нужно тестирование ?

Зачем тестировать?

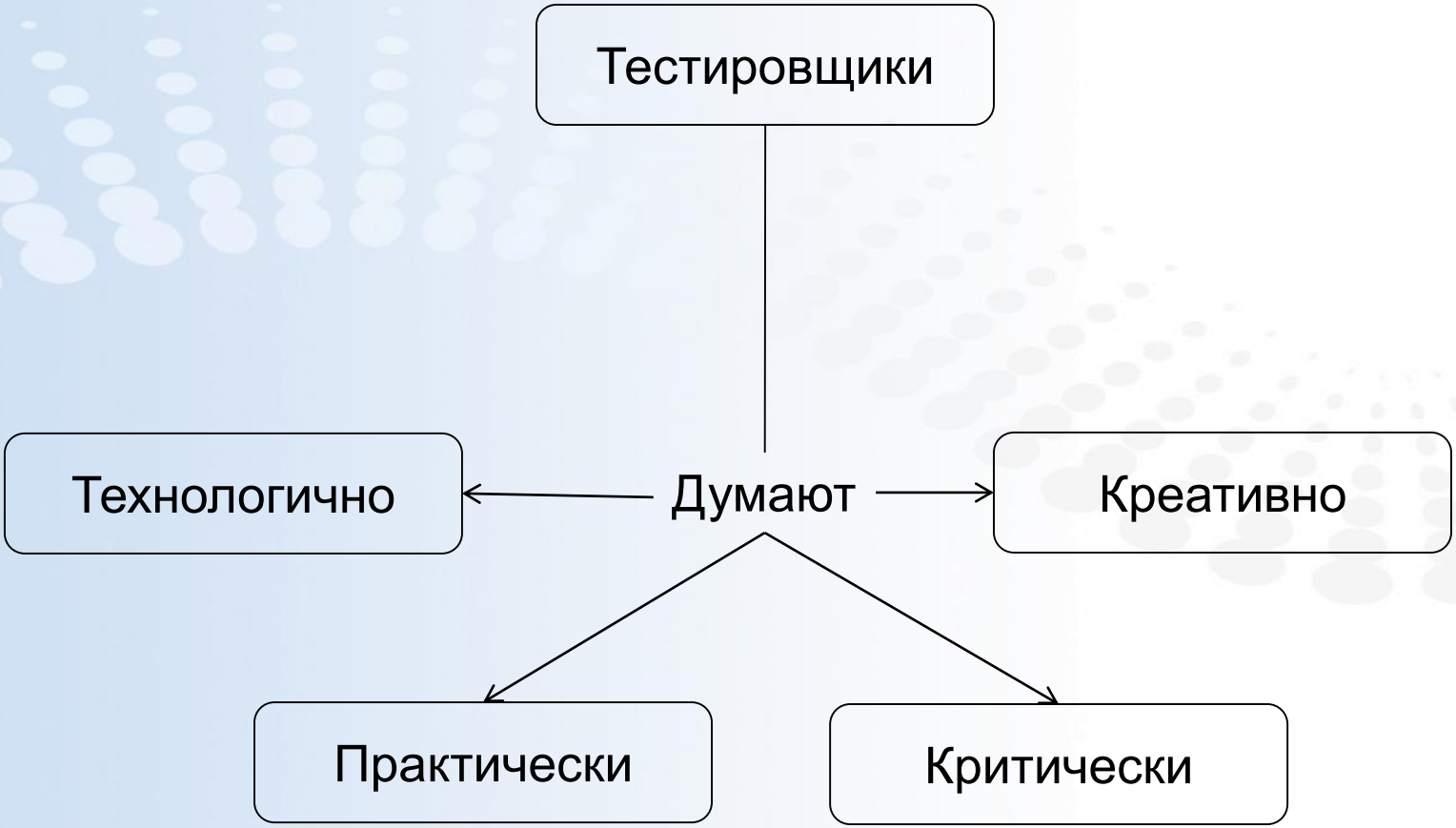
В 2010-2011 годах потери экономики США от некачественного программного обеспечения составили около 25,8 миллиардов долларов в год

(The Economic Impacts of Inadequate Infrastructure for Software Testing. NIST Report, May2012)

- Одна из первых хорошо описанных ошибок такого рода — ошибка в системе управления космическим аппаратом Mariner 1, которая привела к потере этого аппарата 22 июля 1962 года. Именно после этого инцидента управление военно-воздушных сил США приняло решение использовать в процессе разработки ПО экспертизу кода — его просмотр и анализ другими людьми, помимо самого разработчика.
- Ошибка в программном обеспечении, управляющем аппаратом радиационной терапии Therac-25 1985-87 г
- 25 февраля 1991 года американская система ПВО Patriot не смогла сбить иракскую ракету Скад, которая в результате попала в барак американской армии, убив 28 человек и ранив около ста
- Фондовая биржа в 2010году пострадала от технических неполадок во время первой фазы миграции на новую технологическую платформу, торги на альтернативной платформе были возобновлены лишь часом позже (что повлекло за собой значительные убытки для биржи).

Почему мы тестируем программное обеспечение?

- Обеспечение информацией программистов, которую они могут использовать для предотвращения ошибок
- Обеспечение менеджеров информацией, которая необходима им для разумной оценки риска при использовании объекта
- Проверка искаженности объекта с помощью как сформулированных, так и не сформулированных требований.
- Проверить соответствие объекта (убедиться в его действенности), т.е. показать, что он работает правильно.
- Создание проекта поддающегося тестированию, т.е. проекта, который можно легко проверить на соответствие, на искаженность и который будет легко сопровождать.
- Создание объекта максимально свободного от ошибок



Стандарты

ISO 9126 (ГОСТ Р ИСО / МЭК 9126-93) — «Информационная технология. Оценка программного продукта. Характеристики качества и руководство по их применению».

http://protect.gost.ru/document.aspx?control=7&id=135185

ГОСТ 19

ГОСТ 34

CMMI (Capability Maturity Model Integration) –система и модель для оценки зрелости комплекса технологических процессов жизненного цикла ПС (ISO/IEC 15504)

ISO - International Organization of Standardization – Международная организация по стандартизации

IEC - International Electrotechnical Commission – Международная комиссия по электротехнике.

IEEE — читается «ай-трипл-и», Institute of Electrical and Electronic Engineers, Институт инженеров по электротехнике и электронике.

Стандарт ISO 9126

ISO 9126 это международный стандарт, определяющий оценочные характеристики качества программного обеспечения.
Стандарт разделяется на 4 части:
- Характеристики качества
- Руководство по их применению

Качество (quality) -весь объем признаков и характеристик продукции или услуги, который относится к их способности удовлетворять установленным или предполагаемым потребностям

Качество программного обеспечения (software quality) - весь объем признаков и характеристик программной продукции, который относится к ее способности удовлетворять установленным или предполагаемым потребностям.

Атрибуты качества

Гост 19

19.301-79 ЕСПД. Порядок и методика испытаний

- объект испытаний;
- цель испытаний;
- требования к программе;
- требования к программной документации;
- состав и порядок испытаний;
- методы испытаний.

ГОСТ 34

34.201-89 Виды, комплектность и обозначение документов при создании АС

- Программа и методика испытаний (компонентов, комплексов средств автоматизации, подсистем, систем)

34.601-90 Стадии создания АС

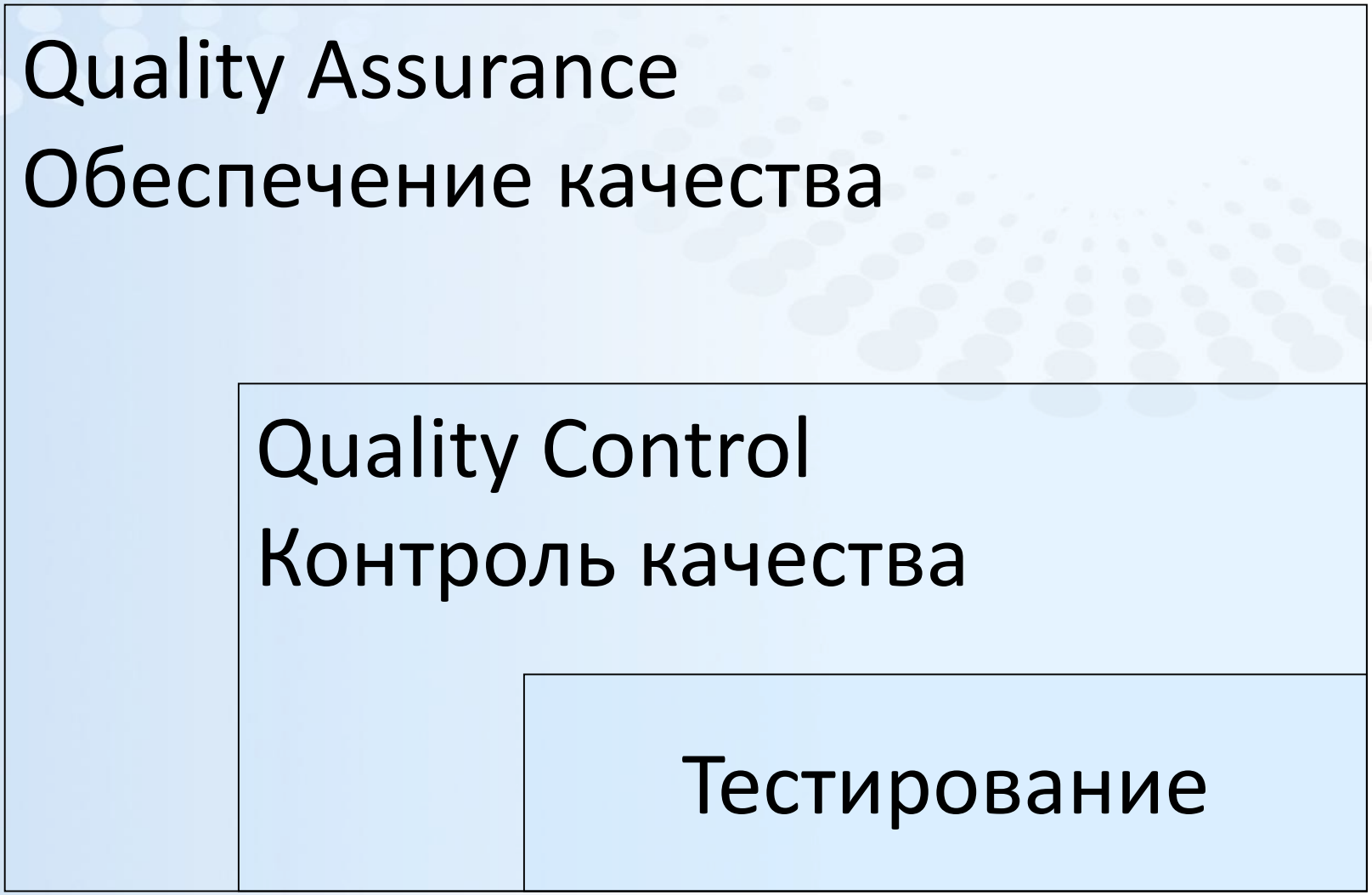
- 7. Ввод в действие

34.602-89 Техническое задание на создание автоматизированной системы

34.603-92 Виды испытаний АС

- предварительные;
- опытная эксплуатация;
- приемочные

Тестирование – QC - QA



Классификация методов контроля качества

- Методы и техники, связанные с выяснением свойств ПО во время его работы.
- Методы и техники определения показателей качества на основе симуляции работы ПО с помощью моделей разного рода. К этому виду относятся **проверка на моделях (model checking)**, а также **прототипирование (макетирование)**, используемое для оценки качества принимаемых решений.
- Методы и техники, нацеленные на выявление нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации. К методам такого рода относится **инспектирование кода.**
- Методы и техники обычного или формализованного анализа проектной документации и исходного кода для выявления их свойств.

Методы контроля качества

- **Верификация** обозначает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.

- **Валидация** — это проверка того, что сам продукт правилен, т.е. подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.

Валидация подтверждает, что «вы создали правильный продукт», а верификация подтверждает, что «вы создали продукт так, как и намеревались это сделать».

История определений

1980

- Процесс выполнения программы с намерением найти ошибки. *[Г.Майерс. Надежность программного обеспечения. М:Мир, 1980]*

1987

- Процесс наблюдения за выполнением программы в специальных условиях и вынесения на этой основе оценки каких-либо ее аспектов. *[ANSI/IEEE standard 610.12-1990: Glossary of SE Terminology. NY:IEEE, 1987]*

1990

- Это не действие. Это интеллектуальная дисциплина, имеющая целью получение надежного программного обеспечения без излишних усилий на его проверку. *[B. Beizer. Software Testing Techniques, Second Edition. NY:van Nostrand Reinhold, 1990]*

1999

- Техническое исследование программы для получения информации о ее качестве с точки зрения определенного круга заинтересованных лиц. *[C. Kaner, 1999]*

2004

- Проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом. *[IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004]*

Тестирование – это

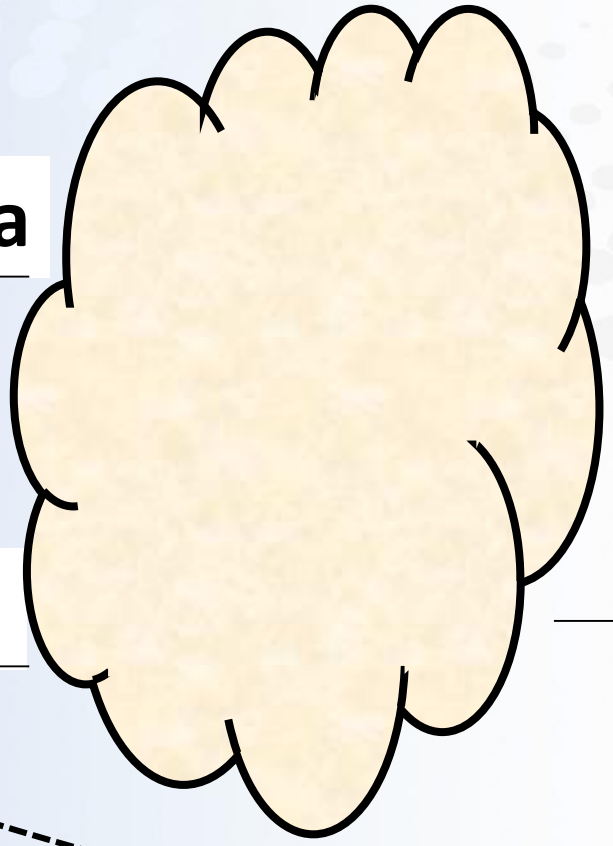
проверка соответствия программы требованиям,

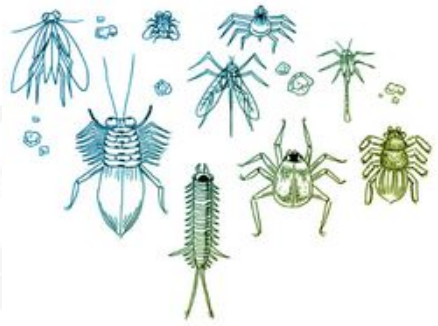
осуществляемая путём наблюдения за её работой в

специальных, искусственно созданных ситуациях,

выбранных определённым образом

Схема тестирования





Информация о несоответствиях

Алексей Баранцев

Тест – это специальная, искусственно созданная ситуация, выбранная определённым образом, и описание того, какие наблюдения за работой программы нужно сделать для проверки её соответствия некоторому требованию.
Тест – это набор входных значений, условий выполнения и ожидаемых значений на выходе, разработанных для проверки конкретного пути выполнения программы.

Стимулы – это данные, которые подаются на вход программе.
Реакции — это то, что получается на выходе.
Оракул**,** любой агент (человек , программа, документ…), оценивающий поведение программы, формулируя вердикт - тест пройден (“pass”) или нет (“fail”).

Методы тестирования – это совокупность правил, регламентирующих последовательность шагов по тестированию.

Критерии тестирования – соображения, позволяющие судить о достаточности проведенного тестирования

Цели тестирования

Подтвердить, что продукт соответствует спецификации и он делает то, что прописано в спецификации и не делает того, чего в документе нет.

- Получить адекватную и актуальную информацию о состоянии проекта (что и в каком объеме реализовано)
- Определить степень готовности продукта к выпуску
- Снизить риски финансовых и не финансовых потерь (как заказчика, так и исполнителя)

Грамотно организованное тестирование дает гарантию того, что:

а) система удовлетворяет требованиям;

б) система ведет себя в соответствии с требованиями во всех предусмотренных ситуациях.

- **Поиск ошибок**, то есть расхождений между наблюдаемым поведением ПО и требованиями к нему
- **Общая оценка качества ПО**. Помимо прямого обнаружения ошибок, оно должно давать информацию о том, где, скорее всего, находятся еще не найденные ошибки, насколько они могут быть серьезны, насколько тестируемая система надежно и корректно работает в целом, насколько корректны и стабильны ее отдельные функции и компоненты.
- ***Обеспечение стабильного развития системы***, предоставление средств для отслеживания изменений в ней и предсказания возможных проблем системы после внесения в нее тех или иных изменений

Задачи тестирования

- **Проверка выполнения требований.**

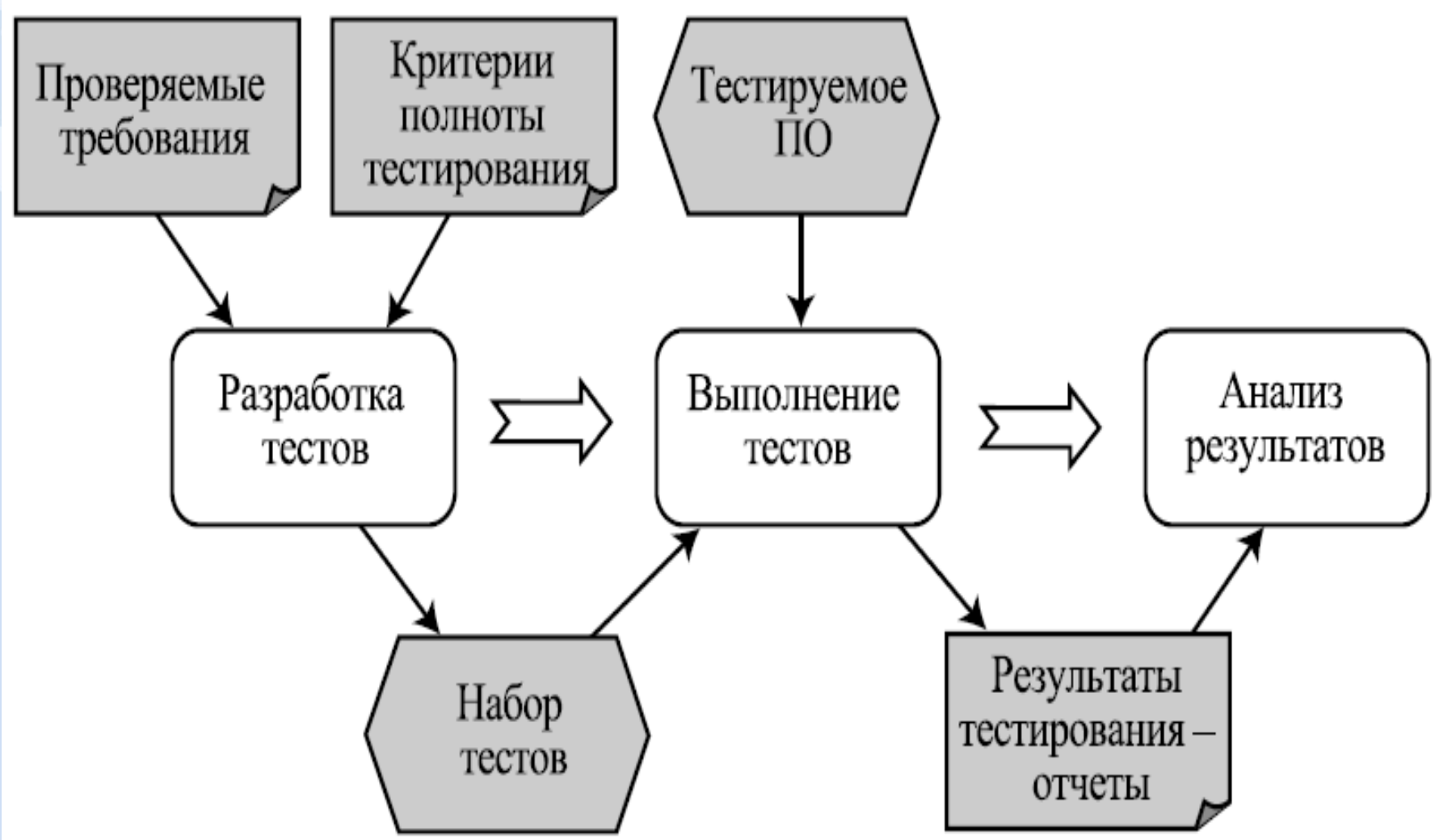
  часто затрудняется отсутствием документов с понятным, однозначным, непротиворечивым и полным описанием требований. Чаще всего при разработке тестов приходится заодно уточнять требования к тестируемой системе и дорабатывать представляющие их документы, делая их более ясными и полными, а также устраняя имеющиеся противоречия.

- **Определение критериев полноты тестирования.**

- **Построение полного набора тестовых ситуаций.**

- **Создание отчетов с информацией о результатах тестирования.**

- **Организация тестового набора для обеспечения удобства его модификации, выполнения и анализа получаемых результатов.**

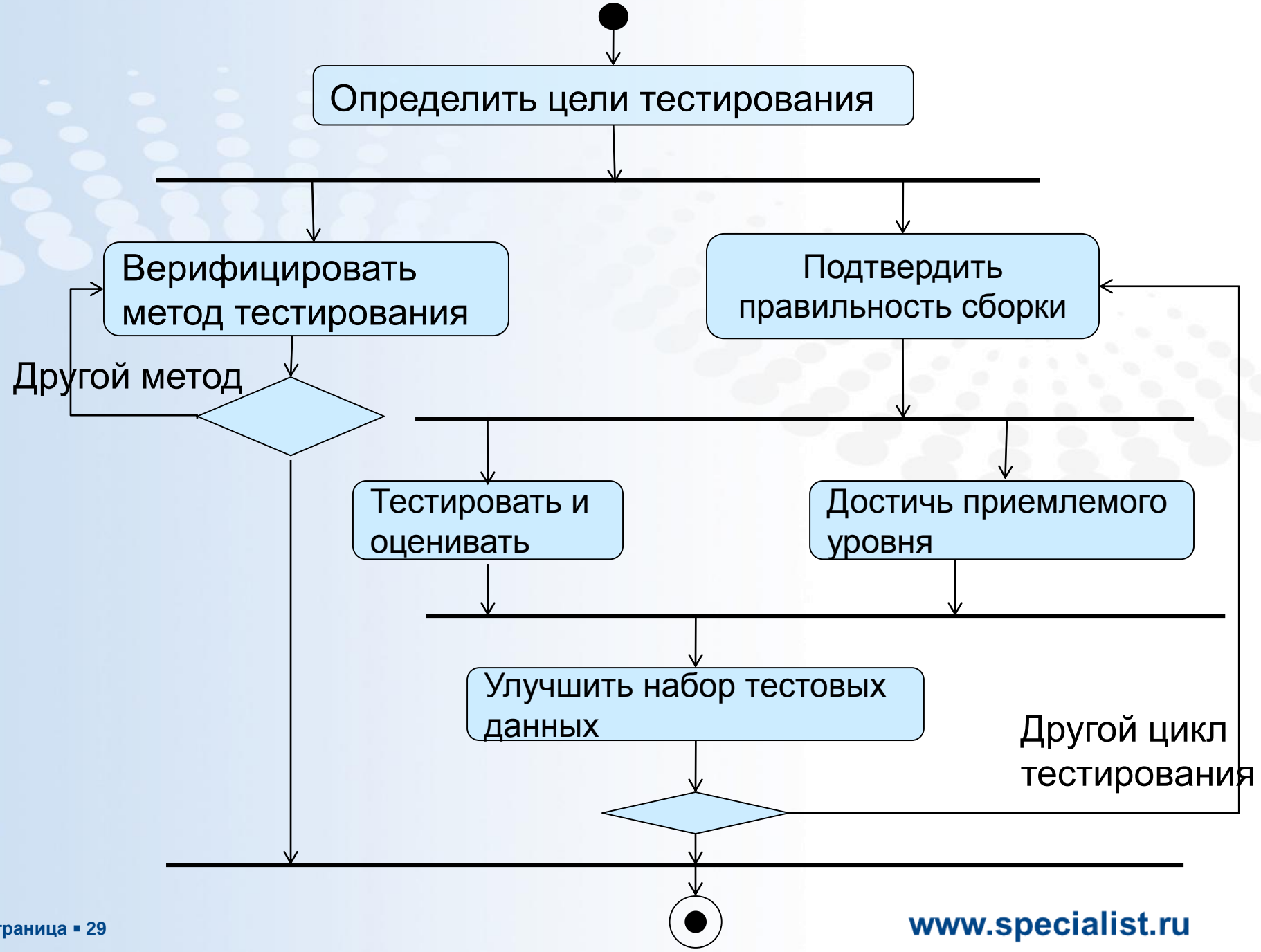


Фазы процесса тестирования

1. Определение целей
2. Планирование
3. Разработка тестов
4. Выполнение тестов
5. Анализ результатов

Тестовый цикл

- Проверка готовности системы и тестов к проведению тестового цикла.
- Подготовка тестовой машины в соответствии с требованиями, определенными на этапе планирования.
- Воспроизведение среза системы.
- Прогон тестов в соответствии с документированными процедурами.
- Сохранение тестовых протоколов (test log).
- Анализ протоколов тестирования и принятие решения о том прошел или не прошел каждый из тестов (Pass/Fail).
- Анализ и документирование результатов цикла.



Полный цикл тестирования

•Планирование тестов (Plan Test)

✓Определение требований к тестам
✓Оценка рисков
✓Разработка стратегии тестирования
✓Определение ресурсов
✓Создание расписания/последовательностей
✓Разработка Плана тестирования

•Дизайн тестов (Design Test)

✓Анализ объёма работ
✓Определение и описание тестовых случаев
✓Определение и структурирование тестовых процедур
✓Обзор и оценка тестового покрытия

•Разработка тестов (Implement Test)

✓Запись или программирование тестовых скриптов
✓Определение тесто-критичной функциональности в Дизайне и Модели реализации
✓Создание/подготовка внешних наборов данных

- **Выполнение тестов (Execute Test)**

✓Выполнение тестовых процедур

✓Оценка выполнения тестов

✓Восстановление после сбойных тестов

✓Проверка результатов

✓Исследование неожиданных результатов

✓Запись ошибок

- **Оценка тестов (Evaluate Test)**

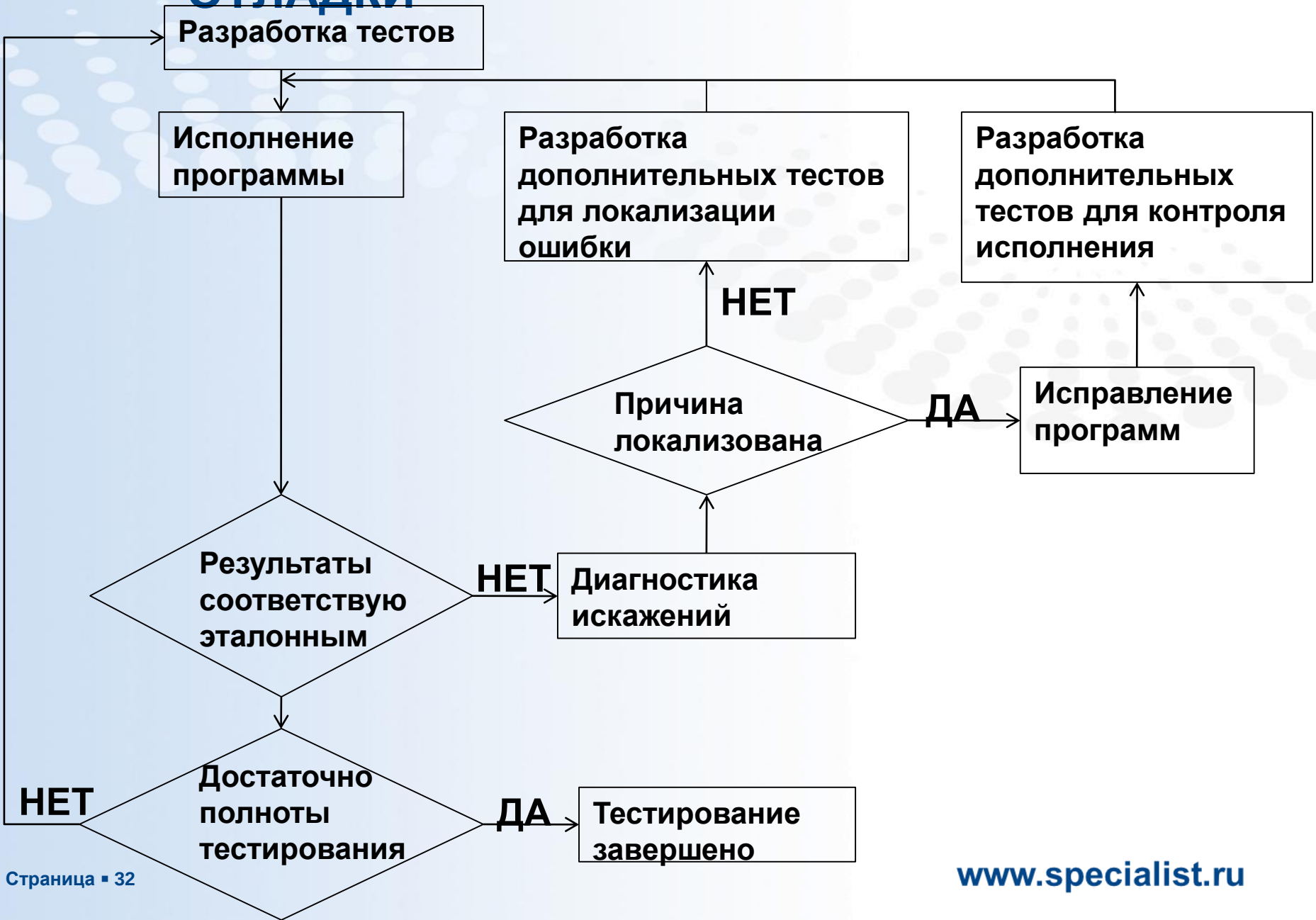
✓Оценка покрытия тестовыми случаями

✓Оценка покрытия кода

✓Анализ дефектов

✓Определение критериев завершения и успешности тестирования

СХЕМА ПРОЦЕССА ТЕСТИРОВАНИЯ И ОТЛАДКИ



Модуль 2

- Методы и виды тестирования
- Критерии покрытия тестирования
- Требования к ПО
- Анализ требований с точки зрения пригодности к тестированию

Динамическое тестирование

ТЕСТИРОВАНИЕ – это проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом [SWEBOK, 2004]

Часто, когда специалисты думают о тестировании, они имеют в виду динамическое тестирование, в рамках которого предусматривается эксплуатация системы с целью выявления дефектов. Вообще говоря, динамическое тестирование состоит из прогона программы и сравнения ее фактического поведения с ожидаемым. Если фактическое поведение отличается от ожидаемого, это значит, что обнаружен дефект.

Статическое тестирование

Статическое тестирование по существу есть все, что можно сделать для выявления дефектов без прогона программного кода.

Цель: подтвердить вывод о том, что рабочий продукт (например, спецификация проекта) правильно реализует все системные требования, и с целью контроля качества проекта.

Преимущество: выявление дефектов на ранних стадиях разработки, благодаря чему достигается существенная экономия времени и затрат на разработку.

Сюда входят проверки, сквозной контроль и экспертные оценки проектов, программных кодов и прочих рабочих продуктов, равно как и статический анализ с целью обнаружения дефектов в синтаксисе, структурах данных и других компонентах программного кода.

Виды тестирования

- По проверяемым свойствам
  - ISO 9126
    - Функциональность
    - Надежность
    - Производительность
      - Нагрузочное
    - Переносимость
    - Удобство использования
    - Удобство сопровождения
  - Регрессионное
  - Аттестационное (соответствия)
- По исполнителю
  - При разработке
  - Альфа
  - Бета
  - Приемочное
- По уровню
  - Модульное (Компонентное)
  - Интеграционное
  - Системное
- По источнику данных
  - «Черного ящика»
  - «Белого ящика»
  - «Серого ящика»
- На отказ
  - «Дымовое»
  - Стрессовое

Критерии покрытия тестирования

Критерии покрытия тестирования – соображения, позволяющие судить о достаточности проведенного тестирования

Покрытие, основанное на спецификации или на требованиях.

Этот критерий оценивает степень покрытия, принимая во внимание требования Заказчика или системные спецификации. Основой может быть, например, таблица требований, use case модель и диаграмма состояний-переходов. Набор тестов должен покрывать все или конкретно определенные функциональные требования. Данный критерий показывает в процентном отношении количество покрытых тестами требований. Чаще всего данный критерий используется при тестировании методом «черного ящика».

Покрытие, основанное на коде

Имеет отношение к потоку управления и потоку данных программы. Чаще всего данный критерий используется при тестировании методом «белого ящика».

Чёрное

Полностью покрыты все …

- … входные данные
- … комбинации входных данных
- … последовательности комбинаций входных данных

Белое

Полностью покрыты все …

- … строки кода программы
- … ветви в коде программы
- … пути в коде программы

Особенности требований к программному обеспечению

IEEE Standard Glossary определяет требования как:

Условия или возможности, необходимые пользователю для решения проблем или достижения целей;

Требования к ПО состоят из трех уровней — бизнес-требования, требования пользователей и функциональные требования. Вдобавок каждая система имеет свои нефункциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы.

Требования пользователей описывают цели и задачи, которые пользователям даст система.

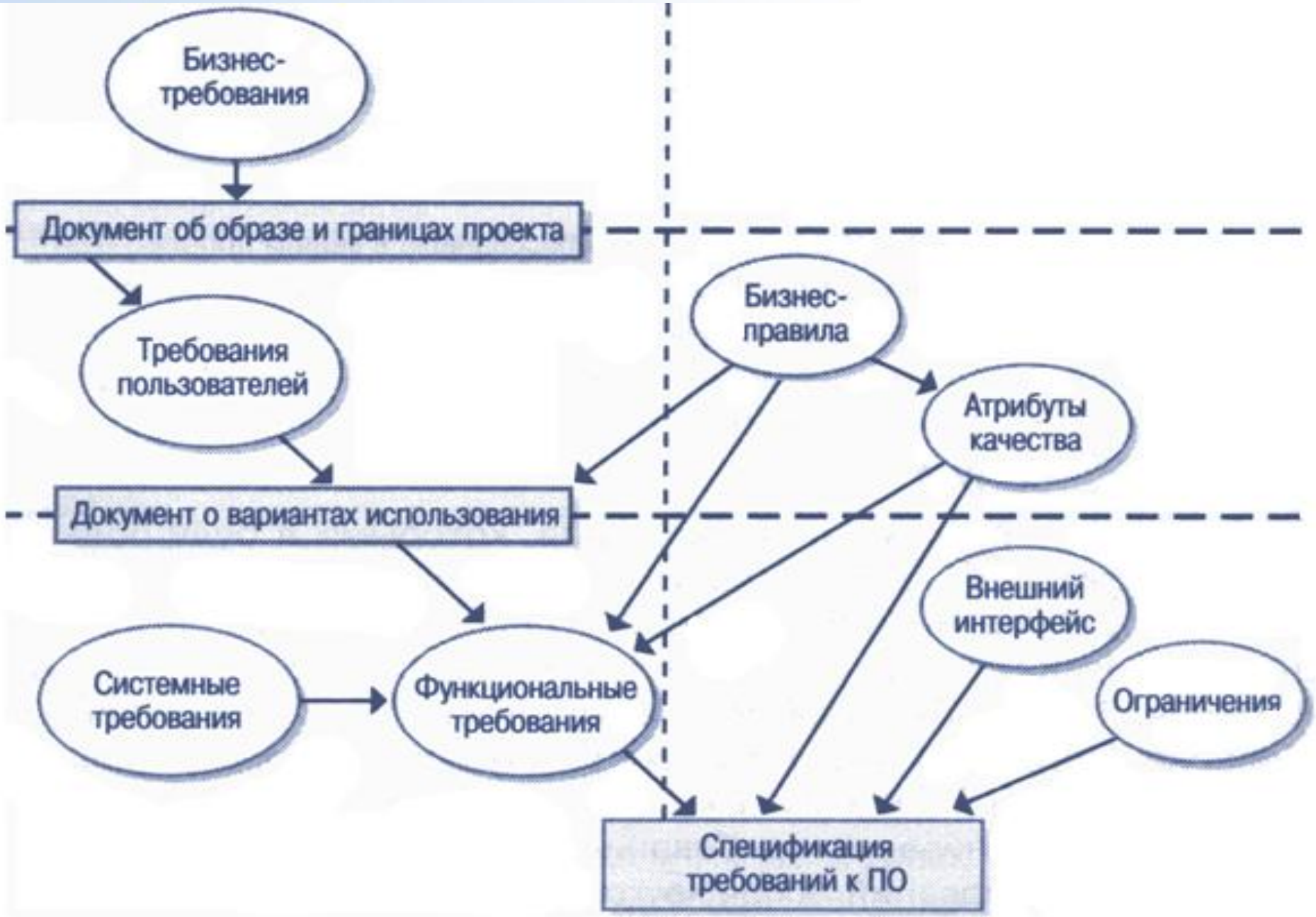
Функциональные требования определяют функциональность ПО, которую разработчики должны построить, чтобы пользователи смогли выполнить свои задачи в рамках бизнес-требований. Они содержат положения с традиционным «должен» или «должна»: «Система должна по электронной почте отправлять пользователю подтверждение о заказе». Функциональные требования документируются в спецификации требований к ПО (software requirements specification, SRS), где описывается так полно, как необходимо, ожидаемое поведение системы.

Системные требования (system requirements) - это высокоуровневые требования к продукту, которые содержат многие подсистемы. Говоря о системе, мы подразумеваем программное обеспечение или подсистемы ПО и оборудования. Люди — часть системы, поэтому определенные функции системы могут распространяться и на людей

Нефункциональные требования описывают цели и атрибуты качества. Атрибуты качества (quality attributes) представляют собой дополнительное описание функций продукта, выраженное через описание его характеристик, важных для пользователей или разработчиков. К таким характеристикам относятся:

- легкость и простота использования
- легкость перемещения
- целостность
- эффективность и устойчивость к сбоям
- внешние взаимодействия между системой и внешним миром
- ограничения дизайна и реализации. Ограничения (constraints) касаются выбора возможности разработки внешнего вида и структуры продукта

Функциональные Нефункциональные



Пример ТЗ

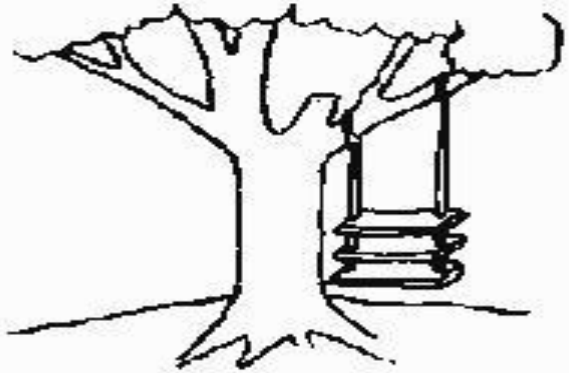
- Подсистема **Front-office** должна располагаться в открытой сети Интернет и представлять собой интернет-портал, обеспечивающий выполнение следующих функций:
- Поиск и отображение ограниченной информации о похищенных, утерянных или утраченных культурных ценностях, зарегистрированных в Системе. Алгоритмы, по которым осуществляется ограничение информации должны быть определены на стадии технического проектирования.
- Авторизация пользователей.
- Автоматизированное заполнение авторизованными пользователями заявлений о хищении, утрате или пропаже культурных ценностей.
- Возможность отслеживания для авторизованных пользователей статусов заявлений и при необходимости автоматизированного предоставления дополнительной информации.

Функции подсистемы Front-office

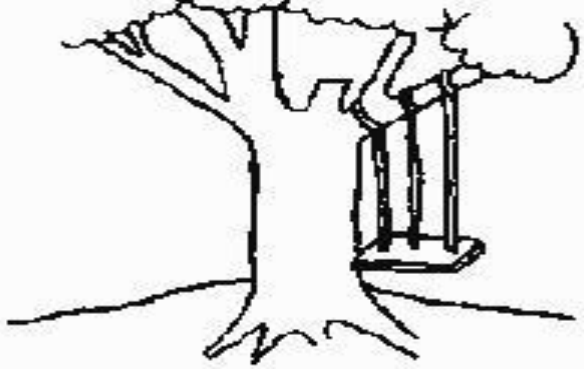
- Подсистема **Front-office** располагается в открытой сети Интернет и представляет собой веб-сайт, обеспечивающий выполнение следующих функций:
- Поиск и отображение ограниченной информации о похищенных, пропавших или утраченных культурных ценностях, зарегистрированных в Системе. Ограничения информации о культурных ценностях касается изображений и надписей, у которых выставлен атрибут «не публиковать». Также пользователям не предоставляется никакая информация о событиях (фактах) хищения, пропажи или утраты, по которым проходят найденные культурные ценности.
- Авторизация пользователей осуществляется посредством ввода на стартовой странице веб-сайта логина и пароля.
- Автоматизированное заполнение авторизованными пользователями заявлений о хищении, утрате или пропаже культурных ценностей. Автоматизация процессов ввода заключается в использовании специализированных справочников при заполнении полей заявления, что обеспечивает единообразие процедур описания и последующей идентификации культурных ценностей.
- Возможность отслеживания для авторизованных пользователей статусов заявлений и при необходимости автоматизированного предоставления дополнительной информации. Статусы поданных пользователем заявлений, определяющие этапы их обработки, отображаются в личном кабинете пользователя, в таблице со списком заявлений.

Источники требований

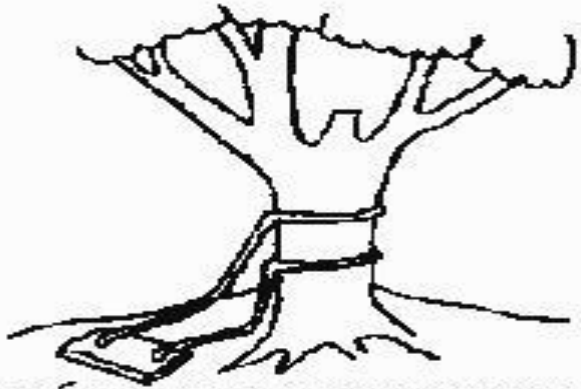
- Требования обычно фиксируются в документе, создаваемом при решении о разработке системы и называемом *техническим заданием* (а по-английски — *requirements specification, спецификация требований*).
- Важным источником требований являются *стандарты*, регламентирующие функции и состав систем, работающих в определенной предметной области. Нормы и законы тоже являются разновидностью действующих стандартов.
- Разработчики сами могут осознать, что что-то должно быть сделано определенным образом — так обычно возникают *внутренние ограничения задач*, не соблюдая которые, невозможно решить их правильно.
- Иногда ряд требований к новой системе можно сформулировать на основе анализа *уже существующих систем* для решения схожих задач.
- Б***о***льшая часть требований формулируется на основе явно высказываемых *пожеланий* пользователей системы, их руководителей, заказчиков ее разработки и других заинтересованных лиц.
- Наконец, наиболее нечетким, но, тем не менее, достаточно важным источником требований являются невысказанные явно *потребности и нужды* пользователей создаваемой системы, которые, несмотря на это, все же часто поддаются анализу.



1. Как было предложено организатором разработки



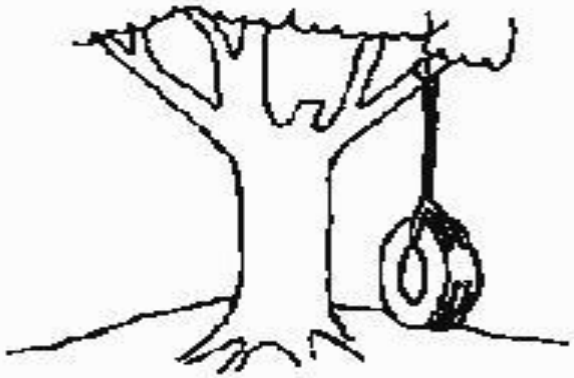
2. Как было описано в техническом задании



3. Как было спроектировано ведущим системным специалистом

4. Как было реализовано программистами

5. Как было внедрено



6. Чего хотел пользователь

Характеристики качества требований

- Корректность
- Однозначность
- Полнота
- Непротиворечивость
- Осуществимость
- Назначение приоритетов.
- Проверяемость

IEEE 830 и IEEE 1233

Анализ требований с точки зрения пригодности к тестированию.

Тестопригодные требования – степень выраженности требований в терминах, допускающих начало работы над разработкой тестов (и в последствии над тестовыми сценариями) и выполнение тестов для определения соответствия заявленным требованиям [IEEE 610]

- **корректность**
- **однозначность;**
- **полнота;**
- **непротиворечивость;**
- **упорядоченность по важности и стабильности;**
- **проверяемость (верифицируемость или тестопригодность);**
- **модифицируемость;**
- **трассируемость;**
- **понятность.**

- **Корректность.** SRS является корректной, если, и только, если каждое требование, изложенное в ней, является требованием, которому должно удовлетворять программное обеспечение.
- **Однозначность.** SRS является однозначной, если и только, если каждое изложенное в ней требование может интерпретироваться только однозначно.
- **Завершенность (полнота).**
- **Непротиворечивость.** Внутренняя непротиворечивость.

  а) Могут входить в конфликт заданные характеристики реальных объектов.

  б) Между двумя заданными действиями может существовать логический или временной конфликт.

  в) Два или более требований могут описывать один и тот же реальный объект, но использовать для этого объекта различные условия.
- **Упорядочивание по значимости.** SRS является упорядоченной по значимости, если каждое требование в ней имеет идентификатор, указывающий или значимость или устойчивость этого конкретного требования

Ранжирование функционала по важности (или по рискам).

- Обязательно
- Важно
- Полезно
- Иногда пригождается
- На всякий случай

По частоте использования / востребованности

- Для всех
- Для большинства
- Для многих
- Для некоторых
- Для единиц

- **Проверяемость.** Требование является проверяемым, если и только, если существует некий конечный эффективный процесс, используя который пользователь или машина могут убедиться, что программное изделие удовлетворяет этому требованию.

Непроверяемые требования включают формулировки типа "работает хорошо", "хороший интерфейс с пользователем" и "обычно должно происходить".

Выходные данные программы должны вырабатываться в пределах 20 секунд в течение 60 % временного интервала события; и должны вырабатываться в пределах 30 секунд в течение 100 % временного интервала события.

Время отклика системы должно находится в приемлемых рамках

Время отклика	(Отклика на какой операции?)
системы	(что такое система в этом требовании: UI, DB, client + server + network?)
должно находится	(Условия? Нагрузка?)
в приемлемых рамках	(Цифры?)

Пример тестопригодного требования

- Время отклика системы с точки зрения конечного пользователя (end-to-end) во время продуктивной нагрузки (50 пользовательских сессий в режиме «менеджер» / 15 пользовательских сессий в режиме «аналитик») при загруженности пропускного канала от клиентской системы до сервера приложений в пределах 50% для сети 100 Mb/sec и утилизации ресурсов сервера приложений (CPU, RAM) в рамках 70-80%, а клиентской машины в рамках 40-60%, не должно превышать 1 секунды для операций создания записи (сущности) и 3 секунд для операций поиска.
- Время выполнения аналитических отчётов определяется отдельно для каждого отчёта

Валидация и верификация требований

Валидация представляет собой проверку корректности и полноты требований, то есть проверяет, что зафиксированные в требованиях ограничения и пожелания действительно представляют потребности пользователей, заказчиков и других заинтересованных лиц, а также, что все их существенные потребности нашли соответствующее отражение в требованиях.

Верификация проверяет внутреннюю согласованность, непротиворечивость и однозначность требований, а также их проверяемость и возможность проследить связи требований друг с другом, с кодом, тестами и другими проектными документами.

Примеры требований: проанализируйте следующие требования с точки зрения тестопригодности.

Решение квадратного уравнения. Программа получает на вход три вещественных числа и интерпретирует их как коэффициенты квадратного уравнения. Результатом работы программы являются два числа — корни этого квадратного уравнения.

Распознавание треугольника. Программа получает на вход три натуральных числа и интерпретирует их как длины сторон треугольника. Результатом работы программы будет сообщение является ли треугольник разносторонним, равнобедренным или равносторонним.

Банкомат должен выполнять следующие операции:
- Прием пластиковой карты
- Проверка ПИН-кода
- Выдача наличных
- Выдача чека с информацией об остатке на счете

Web-сайт библиотеки

Библиотека должна позволять:

- Ведение реестра книг
- Выдача и возврат книг
- Одновременную работу большого числа пользователей

Составление тестов на основе требований

- Связывание рисков и требований выявляет плохо сформулированные или отсутствующие требования.

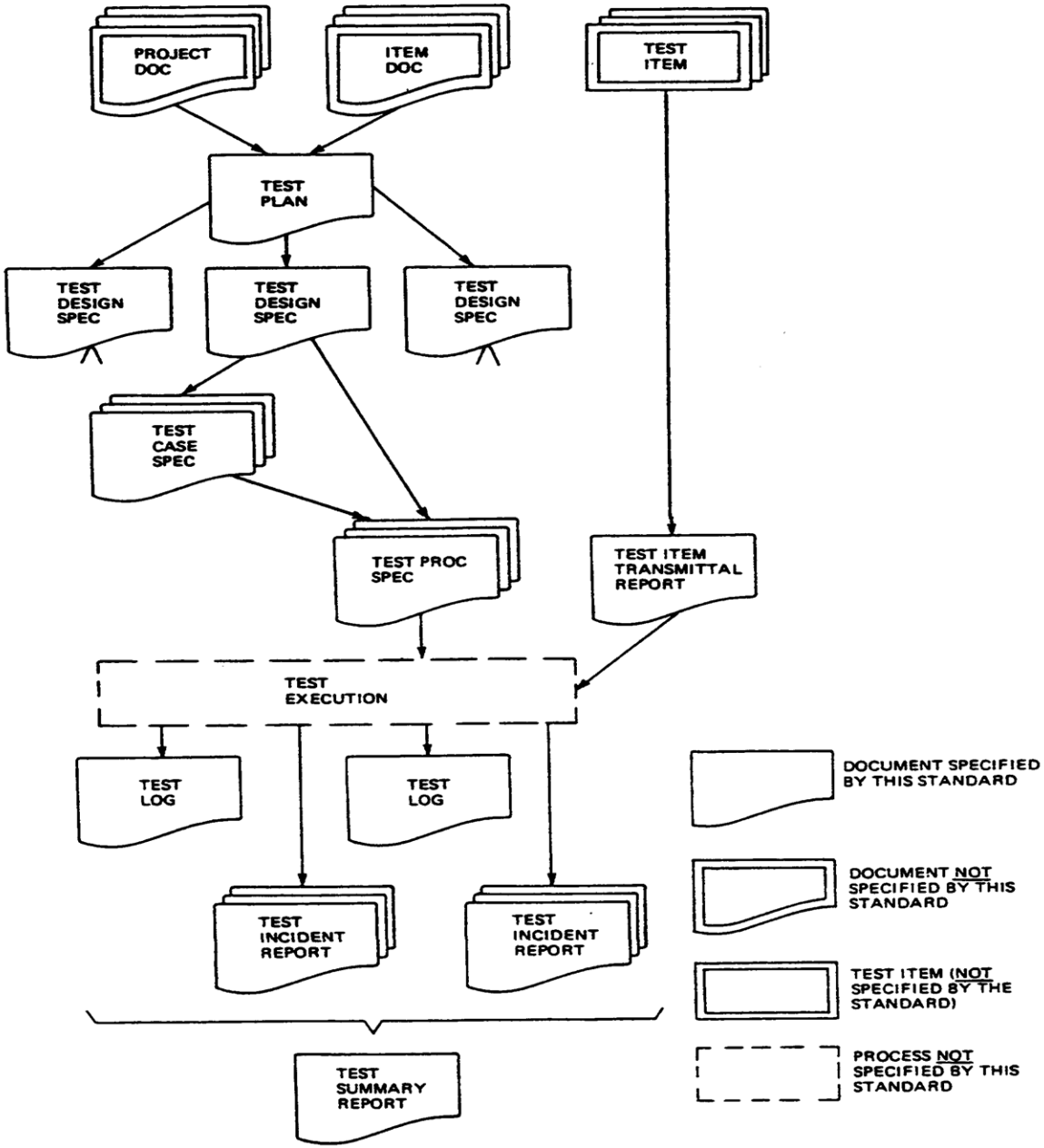
- Тестирование может быть сфокусировано на наиболее важных частях информационной системы, «атакуя» самые опасные технологические риски.

- Работа и коммуникация идет «на едином языке» заказчика и других заинтересованных лиц. Например, заказчику будут понятны отчеты о ходе тестирования проекта, и будет легче принимать решение, инвестировать ли ресурсы в дополнительное тестирование, или риск уже приемлем и можно запускать систему в эксплуатацию.

Модуль 3. Тестовая документация

- Документы, создаваемые в процессе тестирования.
- Тест план
- Связь тестовых планов с другими типами документов.
- Тест – дизайн.
- Возможные формы подготовки тест-дизайна.

IEEE 829 Standard for Software Test Documentation



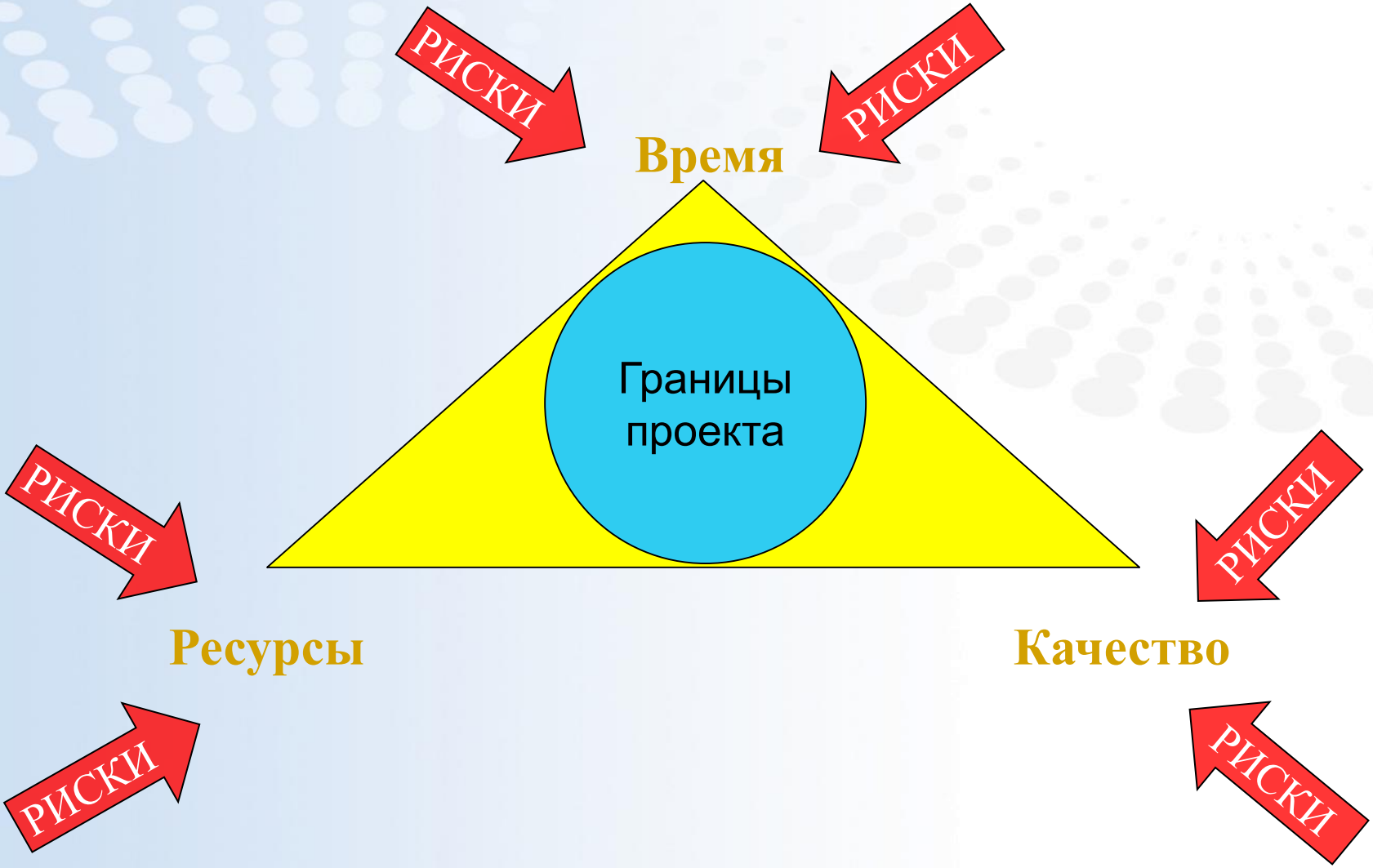
В [IEEE 829] задается формат и указывается содержимое следующих типов документов.

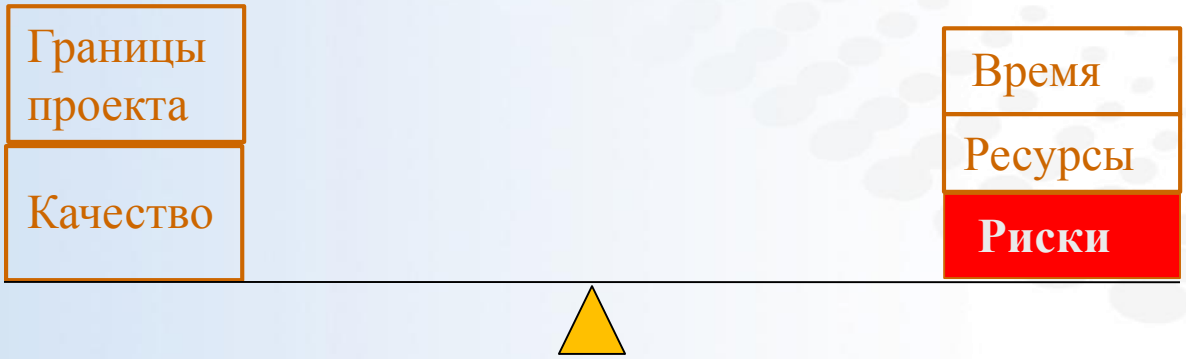
1. **План тестирования**. Цель плана тестирования – обеспечить полноту процесса тестирования. План тестирования разрабатывается на основе ТЗ – требований к продукту. В плане тестирования описываются способы, виды и критерии тестирования для всех требований, необходимые ресурсы и порядок выполнения тестирования.
2. **Спецификация тест-дизайна.** Этот документ определяет, как будет тестироваться каждая функция или группа функций программы.
3. **Спецификация тестов.** Разрабатывается отдельно для тестирования каждой функциональности
4. **Спецификация проектирования процедуры тестирования**: устанавливает этапы, необходимые для выполнения набора тестов. Разрабатывается, если последовательность действий имеет какие-либо специфичные особенности, не описанные в плане тестирования.
5. *Отчет о передаче тестируемого элемента*: определяет тестируемый элемент, указывает локализацию элемента, среду развертывания, а также имя специалиста, ответственного за тестирование этого элемента.
6. *Журнал тестирования*: содержит важные детали, касающиеся выполнения теста.
7. *Отчет о происшествиях*, возникших в ходе тестирования: документировать обнаруженные во время тестирования события, которые требуют дополнительного изучения. Система составления отчетов о неполадках — это один из способов регистрации сведений.
8. **Итоговый отчет о тестировании**: дает общую оценку работе по тестированию.

Планирование

- Определение стратегии тестирования
- Определение состава и структуры испытательной системы (аппаратных и программных средств)
- Оценка трудозатрат (ресурсы и график работ)
- Оценка рисков невыполнения графика работ и подготовка плана мероприятий по смягчению последствий от овеществления этих рисков.
- Подготовка и пересмотр (утверждение) документов с планом проведения испытаний.

Пять характеристик основы плана





Стратегия тестирования

- должна определять объемы тестовых работ,
- типы методик тестирования, которые должны применяться для обнаружения дефектов,
- процедуры, уведомляющие об обнаружении и устраняющие дефекты,
- критерии входа и выхода из испытаний, которые управляют различными видами тестирования.

Объемы тестовых работ

- В первую очередь тестируются требования с наивысшим приоритетом.
- Новые функциональные возможности и программный код, который изменялся.
- Те участки, в которых наиболее вероятно присутствие проблем (ошибки имеют свойство накапливаться).
- Функции и конфигурации, с которыми наиболее часто будет иметь дело конечный пользователь.

Тест план

Тестовый план - это документ, или набор документов, описывающий весь объем работ по тестированию, содержащий следующую информацию:

- Идентификатор;
- Введение;
- Описание тестируемой программы и ее компонент;
- Перечень тестируемых функций/частей программы;
- Перечень не тестируемых функций/частей программы;
- Подход к тестированию (ручное, автоматическое…);
- Критерии успешности тестирования (условия достижения целей тестирования, метрики – если есть);

- Критерии приостановки и возобновления тестирования;
- Что производим в результате тестирования (отчеты);
- Команда и описание сферы ответственности каждого члена команды;
- Расписание (какие тесты и когда выполняются);
- Перечень рисков;
- Ссылка на дополнительные документы;
- Согласования.

Название раздела	Описание
Введение (Introduction)	В разделе приводятся ссылки на исходные документы, описываются общий подход, обеспечивающий полноту тестирования, описываются требования к итерационности разработки на основе снижения рисков и стоимости проведения полного тестирования.
Тестируемые требования (Requirements to be tested)	Приводятся тестируемые требования (указываются ссылки на требования). Устанавливаются правила идентификации и прослеживаемости документов для гарантированного тестирования всех запланированных требований.
Не тестируемые требования (Requirements not to be tested)	Описываются требования (указываются ссылки), для которых не планируются проведение тестирования.
Методы тестирования (Approach)	Основной раздел плана. Включает следующую информацию по всем группам требований, планируемых к тестированию: • Ссылка на требования (идентификатор требования) • Метод тестирования: указывается общий способ тестирования (подход к тестированию), тип тестирования (ручное или автоматизированное), при необходимости дается обоснование специальных методов тестирования • критерий успешности тестов • требования к среде тестирования • требуемые ресурсы • ссылка на тестовую спецификацию (идентификатор тестовой спецификации)
Требования к среде тестирования (Environmental needs)	Указываются общие требования к установке стенда, инструментальным средствам, среде тестирования, требования к разработке дополнительных программ (имитационных, управляющих, поддерживающих) и пр.
Требуемые Ресурсы (Staffing and Training Needs)	Указываются общие потребности в персонале с учетом уровня квалификации, необходимость обучения для проведения тестирования, требования к времени тестирования
Этапы тестирования (Schedule)	Указывается этапы тестирования в связи с этапами разработки и указанием видов тестирования: модульное тестирование, интеграционное тестирование, комплексное тестирование, системное тестирование, опытная эксплуатация (beta – тестирование).
Критерии тестирования (Pass criteria)	Указываются критерии завершения тестирования на различных этапах тестирования. В качестве стандартного критерия завершения тестирования принимается достижение заданного уровня плотности ошибок

Пример критерия окончания тестирования

- Проверены главные функциональные возможности.
- Все важные дефекты исправлены.
- Менее важные дефекты зарегистрированы.
- Обновлена документация.
- Сформирован итоговый отчет по результатам тестирования.

Функциональные требования:
Тестируемая система представляет собой программу, которая ищет корни квадратного уравнения.
Программа на вход получает три действительных числа, которые понимает как коэффициенты a, b,c квадратного уравнения вида:
$ax^2 + bx + c = 0$
Если одно из вводимых значений не число, то в ответ должна выдаваться реплика: «Допустимы только действительные числа».
Если a=0, b=0, c=0, то в ответ должна выйти реплика: «Решением является любое действительное число»
Если a=0, b=0, c<>0, то в ответ должна выйти реплика: «Решений нет».
Если a=0, b<>0, c<>0, то в ответ должна выйти реплика: «Уравнение линейное x=-c/b».
Если дискриминант меньше 0, то в ответ должна выйти реплика: «Решений нет».
В остальных случаях выдается ответ :
$x1=(-b-sqrt(b^2-4ac))/2a;$
$x2=(-b+sqrt(b^2-4ac))/2a;$

Пример плана тестирования

План тестирования, TP:

План тестирования программы поиска корней квадратного уравнения

1.Ссылки

- [SRS] Software Requirement Specification — см. функциональные требования,
- [TDS] Test Design Specification — см. Приложение 1.
- [TCS] Test Case Specification — см. Приложение 2

2. Введение

Данный документ представляет собой план тестирования программы, которая ищет корни квадратного уравнения. Этот план предназначен для учебных целей. В рамках данного плана предполагается выполнить функциональное тестирование программы в режиме вычисления корней. Тестирование производится с точки зрения конечного пользователя, и разработанные тесты могут быть использованы для приёмочного тестирования.

3. Тестируемая система

Тестируемая система представляет собой программу, которая ищет корни квадратного уравнения.

Требования к системе описаны в [SRS].

У программы можно выделить два режима работы — (R1) проверка что все входные данные действительные числа и (R2) вычисление корней квадратного уравнения. Второй режим соответствует ситуации, когда все входные данные действительные числа

4. Тестируемые аспекты

В рамках данного плана предполагается выполнить:

Функциональное тестирование системы в режиме (R2).

5. Нетестируемые аспекты

В рамках данного плана не предполагается выполнять:

Функциональное тестирование системы в режиме (R1).

Нефункциональное тестирование, в том числе нагрузочное тестирование, тестирование производительности, тестирование удобства использования (usability)

6. Подход к тестированию

Уровень тестирования: системное, с точки зрения конечного пользователя.

Специальные средства тестирования: отсутствуют, тестирование будет производиться вручную.

7. Критерии успешности тестирования

Метрики: покрытие путей в модели, описанной в [TDS]. В рамках данного плана предполагается создать комплект тестов, полный относительно этой метрики.

Система передается в эксплуатацию, когда разработан полный комплект тестов и все разработанные тесты выполняются без ошибок.

8. Критерии приостановки и возобновления тестирования

Система возвращается на доработку, если хотя бы один из разработанных тестов обнаруживает ошибку. После исправления ошибки система снова передается на тестирование.

9.Поставка

В результате выполнения данного плана должны появиться:

Документ [TDS]

Комплект тестов, оформленный в виде [TCS].

10. Требования к окружению

Для выполнения тестов требуется установленная тестируемая программа.

Написать тест план для тестирования странички входа в систему. Страничка содержит: поля ввода «имя пользователя», «пароль», кнопку «войти», ссылку «забыли пароль?» , ссылку «зарегистрироваться»

Если пользователя нет в системе, выдавать сообщения об ошибке : «Имя пользователя указано неверно»

Если пользователь в системе зарегистрирован, но пароль введен не правильно, то выдавать реплику «пароль указан неверно» и очищать поле пароль

Тест - дизайн

Тест – дизайн – это этап процесса тестирования ПО, который включает создание/проектирование тестовых сценариев и определение необходимых типов тестов, для достижения заданного уровня тестового покрытия приложения или системы под тестом.

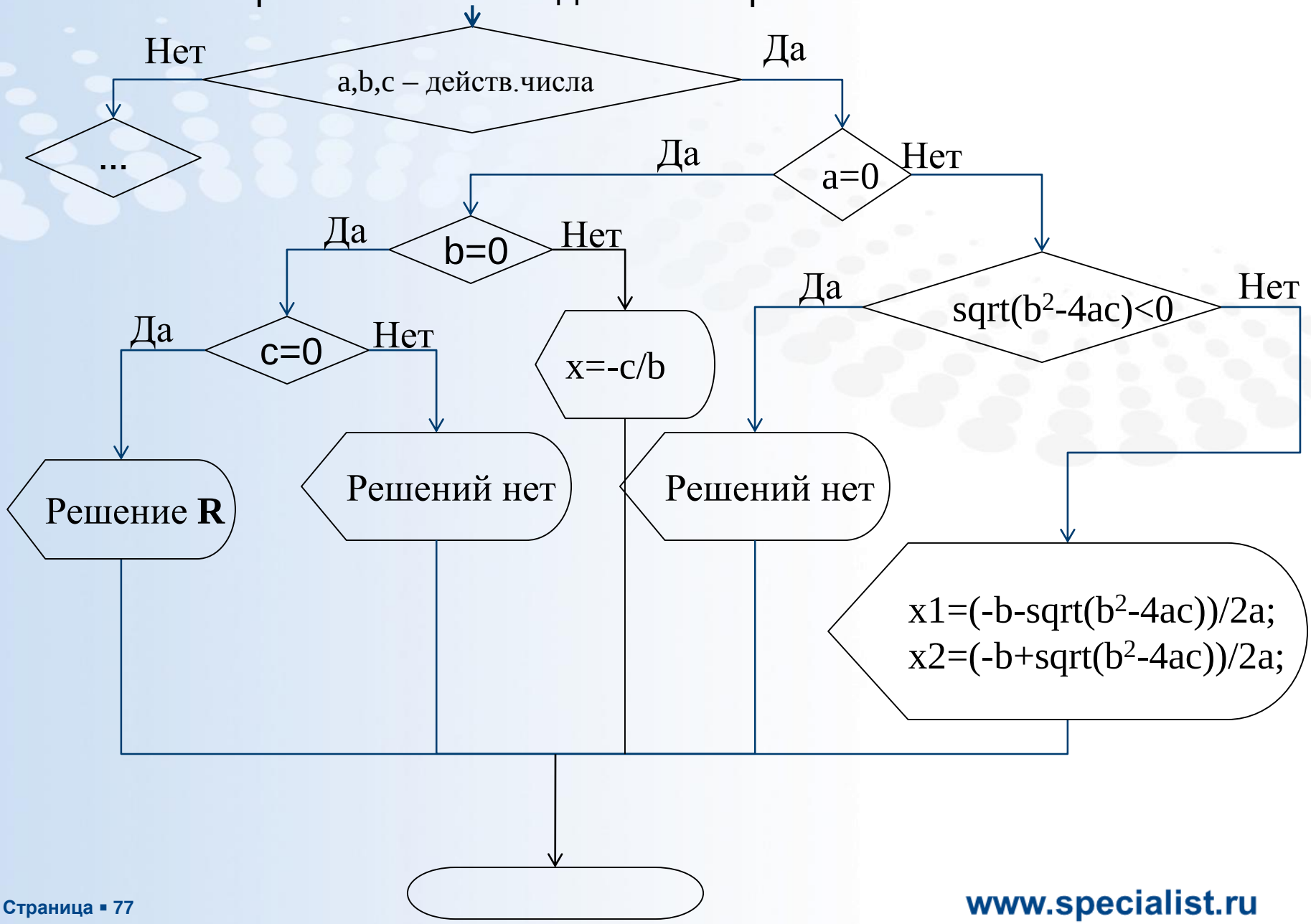
- Идентификатор
- Что тестируем
- Подходы к тестированию (более подробно, чем в тест плане – инструменты, шаблоны, методы…)
- Ссылки на тесты
- Критерии успешности тестирования

«Треугольник»

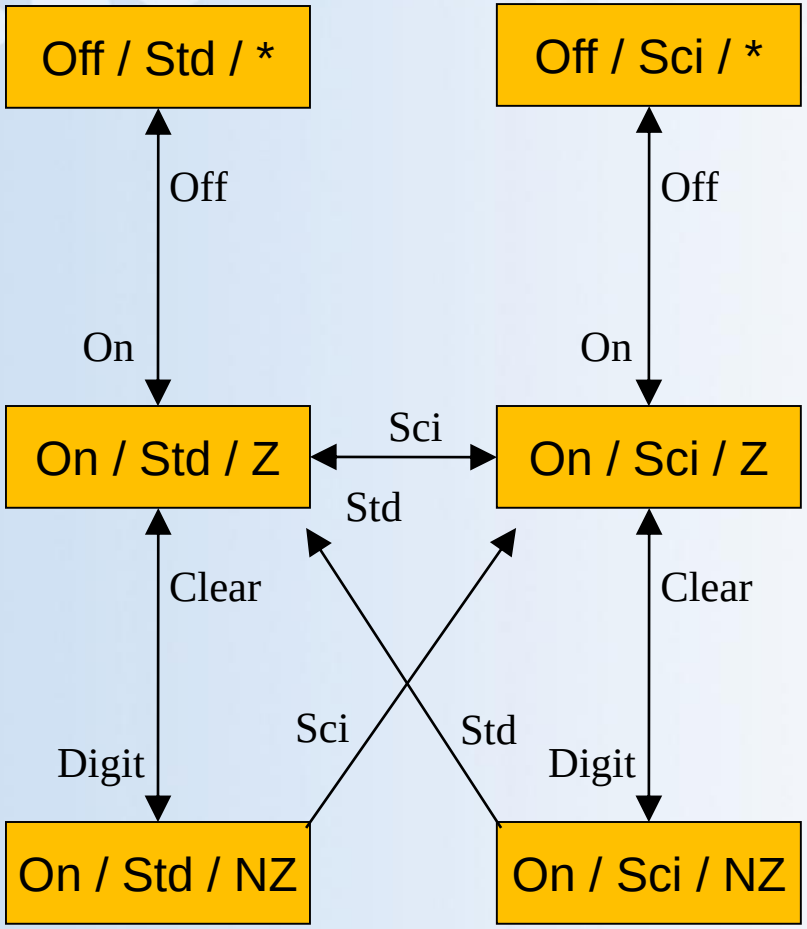
- Программа запрашивает три натуральных числа
- Определяется тип треугольника, имеющего стороны введенной длины: равносторонний, равнобедренный, разносторонний

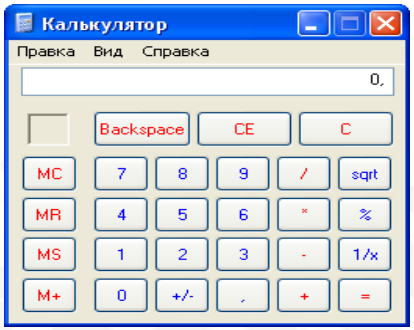
- Корректный разносторонний треугольник
- Корректный равносторонний треугольник
- Три корректных равнобедренных треугольника (a=b, b=c, a=c)
- Одна, две или три стороны равны нулю (5 тестов)
- Одна сторона отрицательная
- «Почти» выполняется правило треугольника (три варианта a+b=c, a+c=b, b+c=a)
- Не выполняется правило треугольника (три варианта a+b<c, a+c<b, b+c<a)
- Нецелое число или не число
- Неправильное количество аргументов

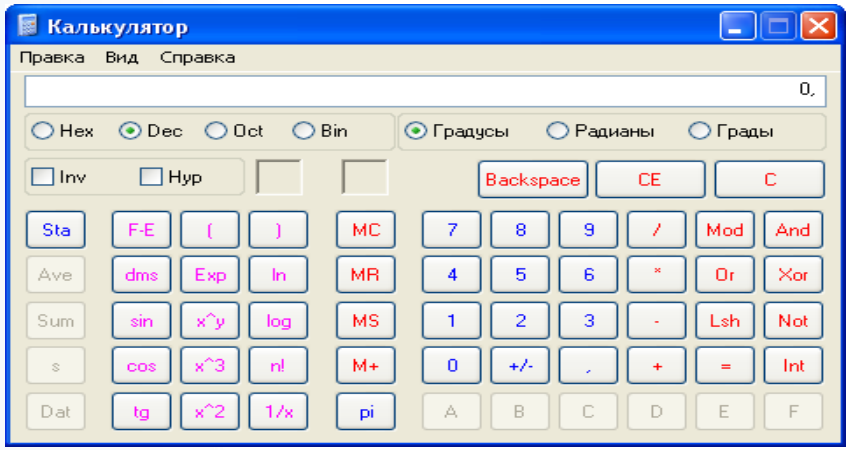
Приложение 1. Модель тестирования TDS



Пример







Модуль 4. Тестовая документация (Test Case). Тестовая документация (отчет о прохождении тестов).

- Определение Test Case.
- Правила написания, степень детализации, независимость.
- Ведение системы отслеживания ошибок (багтрекинговые системы).
- Правила составления описаний ошибок, понятие приоритета, критичности.
- Составление отчетов по результатам тестирования.

Test case

Тестовый сценарии (test case) есть набор входных данных тестов, условий выполнения тестов и ожидаемых результатов, который ориентирован на достижение конкретной цели.

Описание test case содержит:

- Что тестируем
- Входные данные
- Выходные данные
- Окружение
- Порядок действий
- Зависимости

Два подхода к созданию test cases

Каскадные тесты – test case строятся друг за другом.

Например в тестировании базы данных рассмотреть эти тесты:

- Создать запись
- Прочитать запись
- Изменить записи
- Прочитать запись
- Удалить запись
- Прочитать удаленную запись

Каждый из этих тестов может быть построен на предыдущих тестах.

Преимуществом является то, что каждый тест, как правило, меньше и проще.

Недостатком является то, что если один тест не пройден, последующие тесты могут быть недействительными.

Независимые тесты - Каждый тест является полностью автономным. Тесты не зависят друг на друга или не требуют, чтобы другие тесты были успешно выполнены.

Преимуществом является то, что любое количество тестов могут быть выполнены в любом порядке.

Недостатком является то, что каждый тест, как правило, более объемный и более сложный и, следовательно, его труднее проектировать, создавать и поддерживать.

Шаблон test case

Информация о тестовом случае	
Идентификатор тестового случая	SC03 ver3.0
Владелец теста	Джин Дуглас (Jean Douglas)
Местонахождение теста (путь)	TestServer:D:\TestProject\TestSuite\SC03.doc
Дата последнего пересмотра	Месяц/день/год
Тестируемое требование	SC101
Конфигурация средств тестирования	ST02
Взаимозависимость тестовых случаев	Выполнить прогон теста SC01 и его настройку перед прогоном данного теста.
Цель теста	Проверить, что допустимые входные значения веса отправляемого груза дают правильные значения стоимости доставки, и что недопустимые входные значения приводят к выдаче сообщений об ошибке.

Методика тестирования

Настройка на прогон теста Не проводится N/A

Шаг	Действие	Ожидаемый результат	Отметка (V)
1.	Щелкнуть на элементе "Стоимость доставки" в главном меню.	Отображается меню "Стоимость доставки"	(V)
2.	Ввести "101" в поле веса доставляемого груза	Сообщение об ошибке "Неправильно указан вес доставляемого груза"	(V)
3.	Ввести "0" в поле веса доставляемого груза	Сообщение об ошибке "Неправильно указан вес доставляемого груза"	X
4.	Ввести "100" в поле веса доставляемого груза	Указан вес доставляемого груза "100 унций"	(V)
5.	Ввести "1" в поле веса доставляемого груза	Указан вес доставляемого груза "1 унция"	(V)

Очистка после прогона теста Не производится **N/A**

Результаты теста

Тестировщик: JD **Дата прогона теста:** месяц/день/год **Результат теста (P/F/B): F**

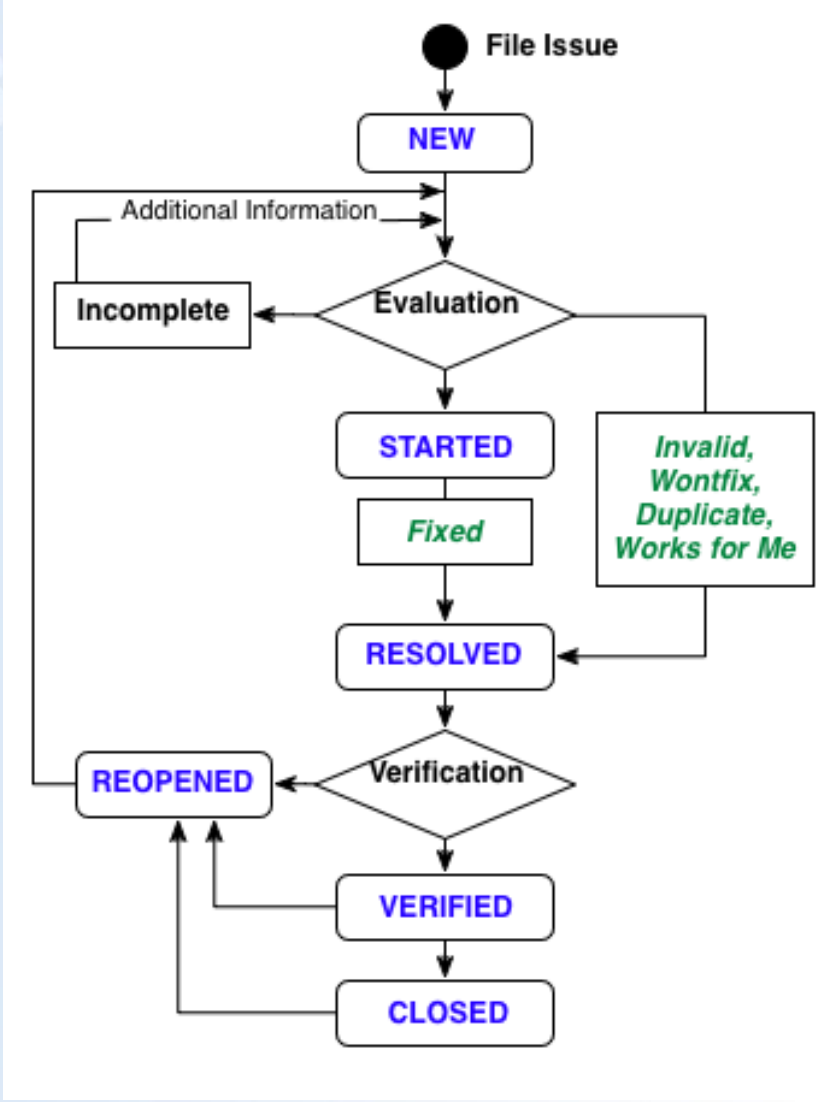
Примечания:

- Тест потерпел неудачу на шаге 3.
- При возникновении неисправности выдается код ошибки BR1011.

Шаблон test case

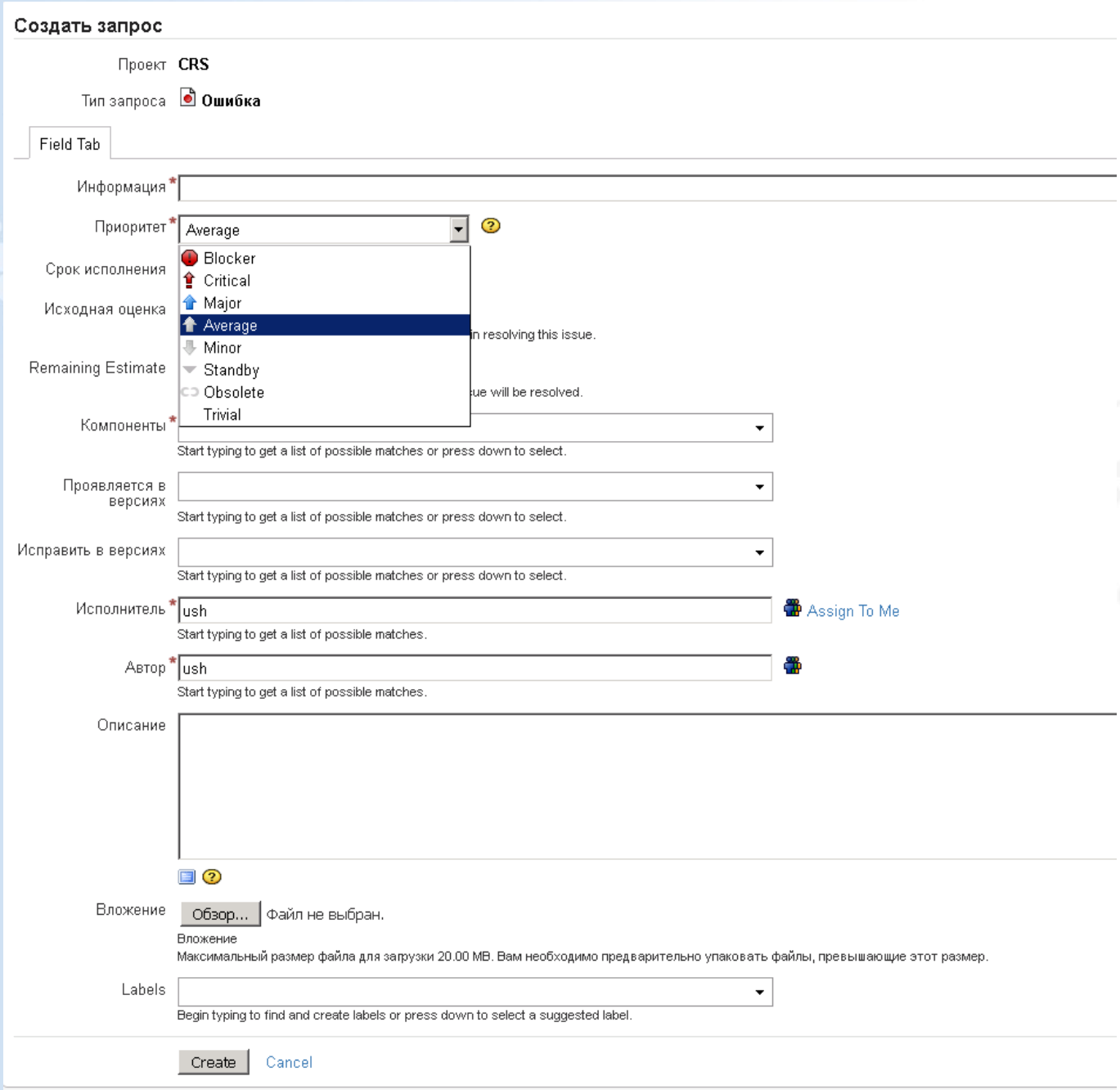
GENERAL INFORMATION			
Test Stage:	☐Unit ☐Functionality ☐Integration ☐System ☐Interface ☐Performance ☐Regression ☐Acceptance ☐Pilot Specify the testing stage for this test case.		
Test Date:	mm/dd/yy	**System Date, if applicable:**	mm/dd/yy
Tester:	Specify the name(s) of who is testing this case scenario.	**Test Case Number:**	Specify a unique test number assigned to the test case.
Test Case Description:	Provide a brief description of what functionality the case will test.		
Results:	☐Pass ☐Fail	**Incident Number, if applicable:**	Specify the unique identifier assigned to the incident.
INTRODUCTION			
Requirement(s) to be tested:	Identify the requirements to be tested and include the requirement number and description from the Requirements Traceability Matrix.		
Roles and Responsibilities:	Describe each project team member and stakeholder involved in the test, and identify their associated responsibility for ensuring the test is executed appropriately.		
Set Up Procedures:	Describe the sequence of actions necessary to prepare for execution of the test.		
Stop Procedures:	Describe the sequence of actions necessary to terminate the test.		

Жизненный цикл дефекта



ВАЖНОСТЬ И ПРИОРИТЕТ ДЕФЕКТА

- **Важность** (**Severity**) - это атрибут, характеризующий влияние дефекта на работоспособность приложения.
- **Приоритет** (**Priority**) - это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Можно сказать, что это инструмент менеджера по планированию работ. Чем выше приоритет, тем быстрее нужно исправить дефект.



Logged in as: *guestqa* (developer) *2008-06-03 10:13 CDT* Project: Demo Switch

Main | My View | View Issues | Report Issue | Change Log | Roadmap | Docs | Wiki | My Account | Billing | Logout Issue # Jump

Recently Visited: 0003568

Viewing Issue Advanced Details [Jump to Notes] [Send a reminder] [Wiki] [Issue History] [Print]

ID	Category	Severity	Reproducibility	Date Submitted	Last Update
0003569	[Demo] GUI	text	sometimes	2008-06-03 10:13	2008-06-03 10:13

Reporter	guestqa	**View Status**	public		
Assigned To	567qwerty				
Priority	normal	**Resolution**	open	**Platform**	
Status	assigned			**OS**	
Projection	none			**OS Version**	
ETA	none	**Fixed in Version**		**Product Version**	1.0
		Target Version	1.4	**Product Build**	

Summary	0003569: Тестовая запись про динозавров
Description	Это тестовый баг-репорт. Он рассказывает нам о том, как трудно жилось девелоперам в эпоху динозавров.
Steps To Reproduce	Открыть левый глаз. Посмотреть влево. ОЖИДАЕМЫЙ РЕЗУЛЬТАТ: увидеть банановую рощу. РЕЗУЛЬТАТ: мерещатся голодные динозавры-менеджеры. См скриншот.
Additional Information	Иногда динозавры издают гортанные звуки, отнюдь не напоминающие словосочетания типа "Идем выпьем пива, брат!"
Tags	No tags attached.
Attach Tags	(Separate by ',') Existing tags Attach
Attached Files	

Update Issue | Assign To: guestqa | Change Status To: new | Monitor Issue | Create Clone | Move Issue | Delete Issue

У БАГОВ ТОЖЕ ЕСТЬ ЧУВСТВА

НАЙДЯ БАГ:
СООБЩИТЕ О НЕМ

БАГИ НЕ ЛЮБЯТ
БЫТЬ ЗАБЫТЫМИ

НАЙДЯ БАГ:
ИЗУЧИТЕ ЕГО

БАГИ ЛЮБЯТ
БЫТЬ ПОНЯТЫМИ

НАЙДЯ БАГ:
СДЕЛАЙТЕ СНИМОК

БАГИ ЛЮБЯТ ХРАНИТЬ
ПАМЯТЬ О ВСТРЕЧЕ

НАЙДЯ БАГ:
УЗНАЙТЕ ЕГО ДРУЗЕЙ

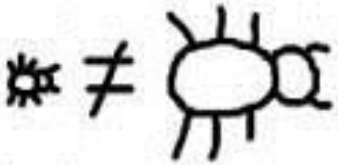
БАГИ ОБЩИТЕЛЬНЫ

НАЙДЯ БАГ:
СООБЩИТЕ О НЕМ БЫСТРО

ИНАЧЕ БАГИ ОБЖИВУТСЯ И
ПОСТРОЯТ СЕБЕ ДОМА

НАЙДЯ БАГ:
БУДЬТЕ ЧЕСТНЫ

БАГИ НЕ ЛЮБЯТ
СПЛЕТЕН



НАЙДЯ БАГ:
ЗАПОМНИТЕ КАК ВЫ
ВСТРЕТИЛИ ЕГО

БАГИ РОМАНТИЧНЫ

НАЙДЯ БАГ:
НЕ ИГНОРИРУЙТЕ ЕГО

БАГИ МОГУТ УКУСИТЬ,
ЕСЛИ ИХ НЕ ЦЕНИТЬ

МОДУЛЬ 5. Техники тестирования

- Покрытие входных данных
- Эквивалентное разбиение
- Анализ граничных значений
- Предположение ошибок

Техники тестирования

Техники, базирующиеся на опыте и интуиции инженера

1. Специализированное тестирование
2. Исследовательское тестирование

Техники, базирующиеся на спецификации

1. Эквивалентное разделение
2. Анализ граничных значений
   тесты оценки живучести (robustness testing)
3. Таблицы принятия решений
4. Тесты на основе user cases
5. Тесты на основе конечного автомата
6. Случайное тестирование

Тестирование, ориентированное на код

- Тесты, базирующиеся на блок-схеме
- Тесты на основе потоков данных
- Ссылочные модели для тестирования, ориентированного на код

Тестирование, ориентированное на дефекты

- Предположение ошибок «Дефекты существа социальные: они стремятся группироваться вместе» (Дороти Грэхэм)
- Тестирование мутаций

Техники, базирующиеся на условиях использования

- Операционный профиль
- Тестирование, базирующееся на надежности инженерного процесса

Техники, базирующиеся на природе приложения

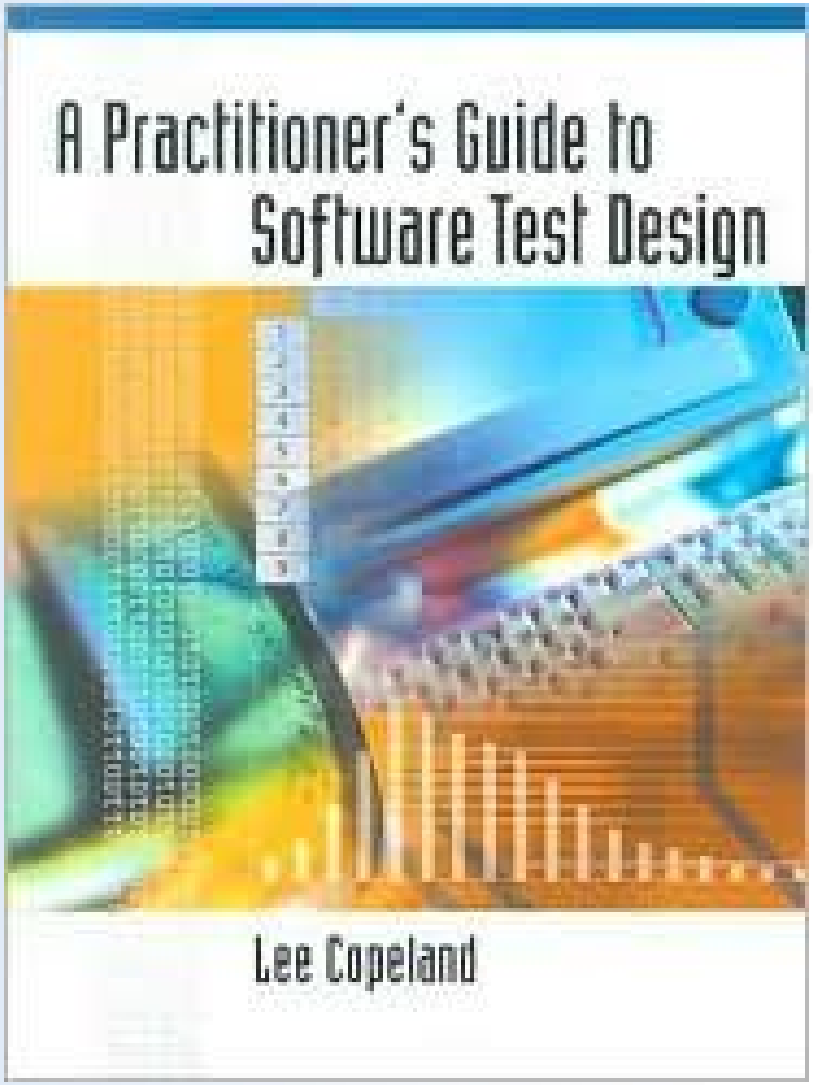
- Объектно-ориентированное тестирование
- Компонентно-ориентированное тестирование
- Web-ориентированное тестирование
- Тестирование на соответствие протоколам
- Тестирование систем реального времени

Выбор и комбинация различных техник

- Функциональное и структурное

- Определенное или случайное

Обычно тесты можно распределить по данным группам на основе используемой политики выбора или определения входных параметров тестов.



```
int blech (int j)
{
j = j -1;      // should be j = j + 1
j = j / 30000;
return j;
}
```

16 бит
минимально входное значение -32768
и максимально 32767.

65536 возможных входных данных

Классы эквивалентности

Класс эквивалентности (Equivalence Classes) – это входные (а иногда и выходные) данные, которые обрабатываются приложением одинаково или обработка которых приводит к одному и тому же результату.

Тестирование классами эквивалентности(Equivalence Class Testing)- это техника тест дизайна способная резонно уменьшить количество ваших тест-кейсов. Использовать ее можно на всех уровнях тестирования - unit, integration, system, and system-integration test levels.

Основная идея, заключается в том, чтобы разбить область ввода программы на классы данных. Если проектировать тесты для каждого класса данных, но не для каждого члена класса, то общее количество требуемых тестов уменьшается.

Авиакомпания предоставляет скидки в зависимости от возраста пассажира.

- 0 – 2 лет 100%
- 2 – 12 лет 50%
- 12 – 21 года 15%
- Свыше 60 лет 15

Возраст пассажира не может превышать 100 лет.

На входе в программу задается дата рождения пассажира

Допустимые значения: от 1 до 100

- 0- 2
- 2 - 12
- 12- 21
- 21- 60
- 60-100

Примеры «чисел»

- перечисления (enumeration)
- символы (character)
- количество (разрешённых установок, записей в БД, строк в файле, цветов, …)
- длина (строки, имени файла, пути, текста в файле, слова, абзаца, …)
- размер/объём (файла, памяти, экрана, окна, шрифта, пакета, …)
- номер (версии), время (интервал), скорость (ввода данных, перемещения мыши), …

Анализ граничных значений

Анализ граничных значений представляет собой технологию проектирования тестов, которая является дополнением разбиения на классы эквивалентности.

На каждую границу создайте 3 тест кейса - один, проверяющий значение границы, второй на значение ниже границы и третий на значение выше границы.

Экспериментально было доказано, что дефекты имеют тенденцию концентрироваться на границе области ввода, а не в ее центре. Не особенно ясно, почему так получается, это всего лишь установленный факт.

Граничные значения

- 0 - 2
- 2 - 12
- 12 - 21
- 21- 60
- 60 - 100

- 0 , 2
- 2, 12
- 12, 21
- 21, 60
- 60, 100

Виды границ

Логические Произвольные Зависящие от реализации

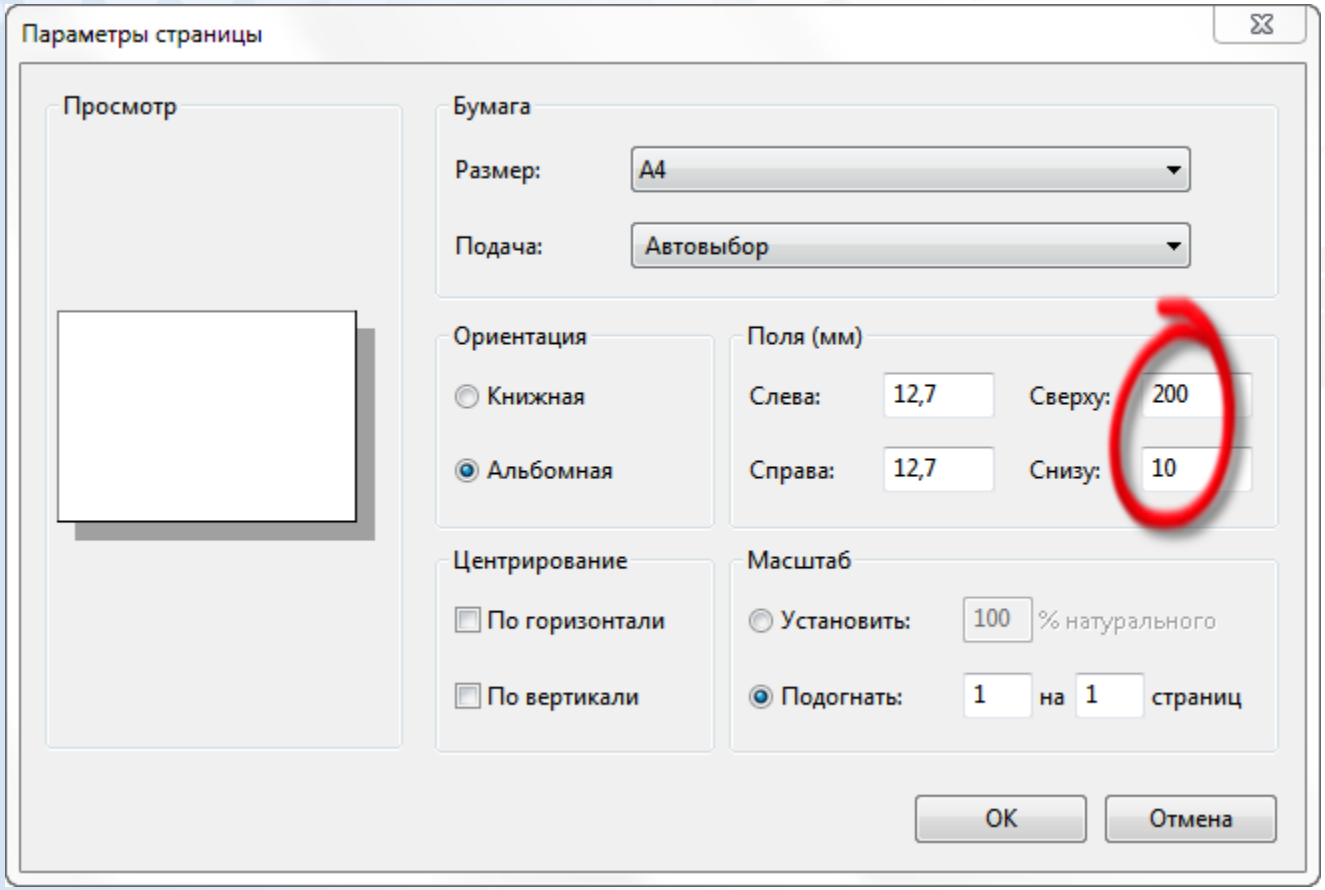
Неверные данные

Набор тестовых данных

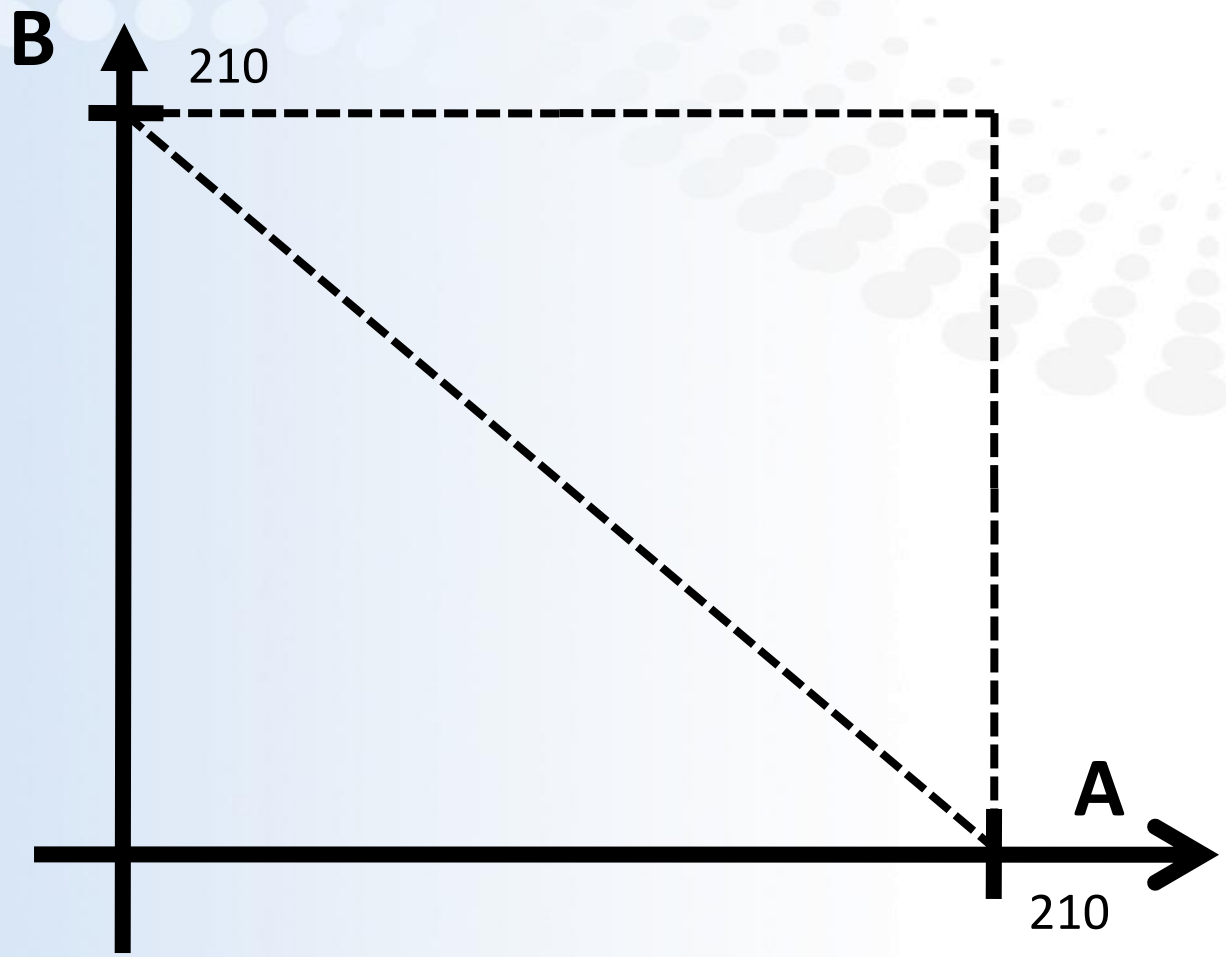
0 - 2	0; 1 min 2	>100;
2 - 12	2, 3, min 12	<0;
12 - 21	12,13, min 21	
21 - 60	21,22, min 60	
60 - 100	60,61,min 100, 100	

Пустое значение, неправильная дата

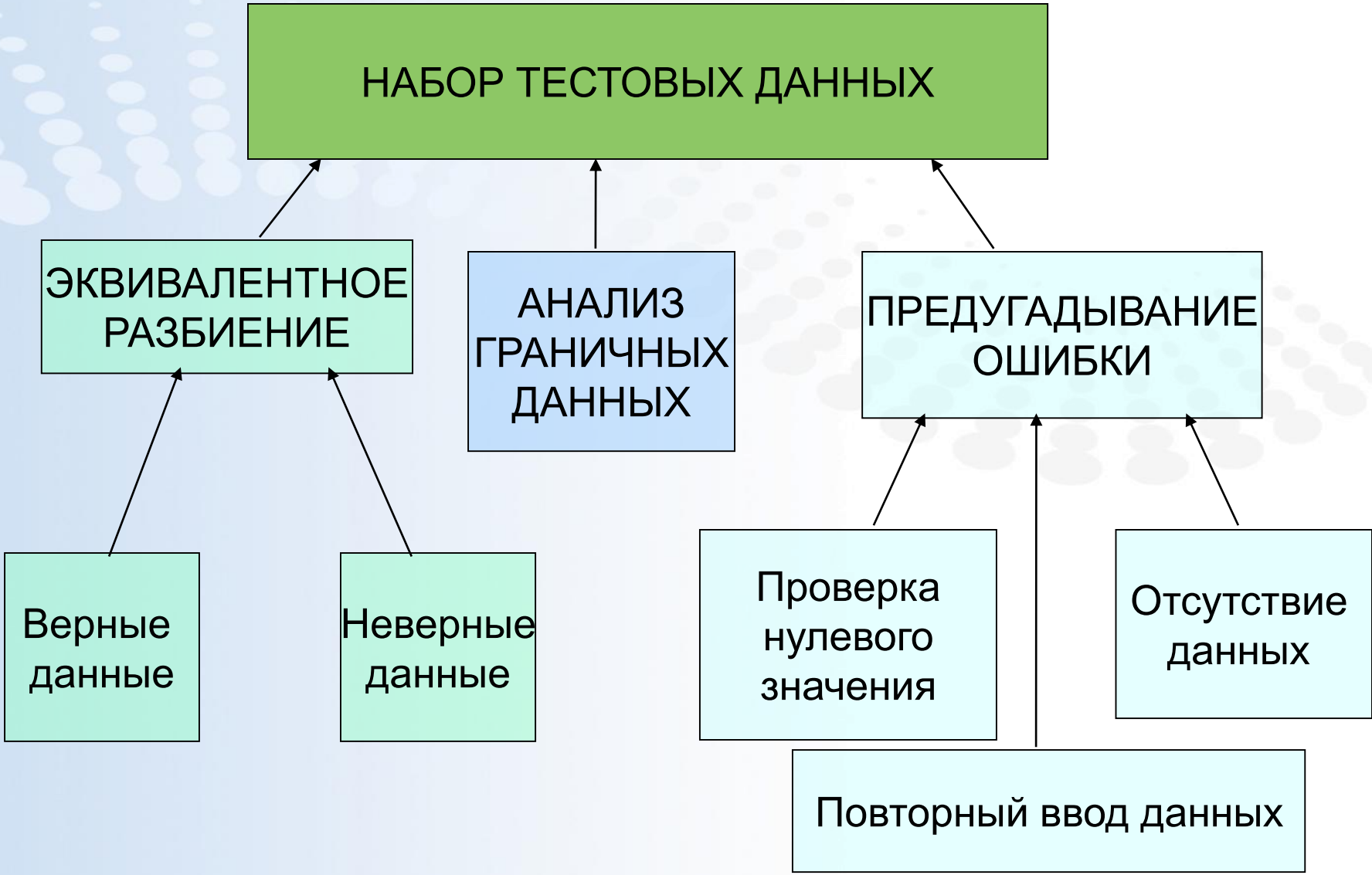
Зависимости между параметрами



Покрытие выходных данных



	Классы корректных данных	**Классы некорректных данных**	**Граничные и специальные значения**
A	0≤A≤MAX		A=0 A=MAX
B	_”_”_	_”_”_	_”_”_
A+B	A+B≤MAX	MAX≤A+B	A+B=MAX



Предугадывание ошибки

- *Внутренние одинарные кавычки (Embedded Single Quote)* – У большинства SQL баз данных возникают проблемы при наличии одинарных кавычек в запросе (например, Jones's car).
- *Обязательный ввод (Required Data Entry)* – В спецификации вашего приложения должны быть четко определены поля требующие обязательного ввода данных.
- *Типы данных полей (Field Type Test)* – В спецификации вашего приложения должны быть четко определены типы данных для каждого из полей (поля даты/времени, числовые поля, поля для ввода номера телефона или почтового кода и т.д.)
- *Размер поля (Field Size Test)* – В спецификации вашего приложения должно быть четко определено максимально допустимое количество символов в каждом из полей.
- *Числовые граничные значения (Numeric Bounds Test)* – Числовые поля вашего приложения могут иметь ограничения допустимых числовых значений. Эти ограничения могут быть указаны в спецификации вашего приложения либо вытекать из логики работы программы.

- *Числовые ограничения (Numeric Limits Test)* – Большинство баз данных и языков программирования определяют числовые значения как переменные с некоторым типом (например, integer или long integer), которые, в свою очередь, имеют ограничения допустимых числовых значений (например, значения integer должны находиться в диапазоне от -32768 до 32767, а long integer от -2147483648 до 2147483647).
- *Граничные значения даты (Date Bounds Test)* – Очень часто в приложениях существуют логические ограничения для полей содержащих дату и время. Например, если вы проверяете поле содержащее дату рождения пользователя, то вполне логичным будет запрет ввода еще не наступившей даты (т.е. даты в будущем), либо ограничение на ввод даты отличающейся от сегодняшней более, чем на 150 лет.
- *Валидность даты (Date Validity)* – Поля даты всегда должны иметь проверку валидности введенных значений (например, 31-06-2012 – не валидная дата). Также, не забывайте и о проверке дат в високосном году (годы кратные 4м и кратные 100 и 400 одновременно - високосные).

Пример

Доменное имя должно содержать от двух до 63 символов, начинаться и заканчиваться буквой латинского алфавита или цифрой.
Промежуточными символами могут быть буквы латинского алфавита, цифры или дефис.
Доменное имя не может содержать дефисы одновременно в 3-й и 4-й позициях.

Правила регистрации доменных имен в домене RU, http://www.cctld.ru/ru/docs/RU-2.php

Модуль 6

- Модульное, интеграционное и системное тестирование.
- Покрытие кода.

Модульное тестирование

Модульное или компонентное (unit testing, component testing) —это тестирование программы на уровне отдельно взятых модулей, функций или классов.

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 .

На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов, типа работы с условиями и счетчиками циклов, а также с использованием локальных переменных и ресурсов.

Модульное тестирование или **юнит-тестирование** (*unit testing*) — процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Цель модульного тестирования

— изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

– удостовериться в соответствии требованиям каждого отдельного модуля системы перед тем, как будет произведена его интеграция в состав системы.

Модульное тестирование

Модуль – это компонент минимального размера, который может быть независимо протестирован в ходе верификации программной системы.

- модуль – это часть программного кода, выполняющая одну функцию с точки зрения функциональных требований;
- модуль – это программный модуль, т.е. минимальный компилируемый элемент программной системы;
- модуль – это задача в списке задач проекта (с точки зрения его менеджера);
- модуль – это один класс или их множество с единым интерфейсом.

Драйвер — это модуль ПО или средство тестирования, которое заменяет модуль, обеспечивающий управление и/или вызов тестируемого модуля.

Заглушка – минимальная или специализированная реализация программного компонента. Использующаяся для подмены компонента от которого зависит разработка или тестирование другого компонента

Особенности дизайна

1. Тесты должны быть простыми, короткими, лёгкими в сопровождении
2. Тесты должны быть изолированы друг от друга
3. Тесты должны иметь минимальное пересечение
4. Тесты должны работать с интерфейсом

Интеграционное тестирование

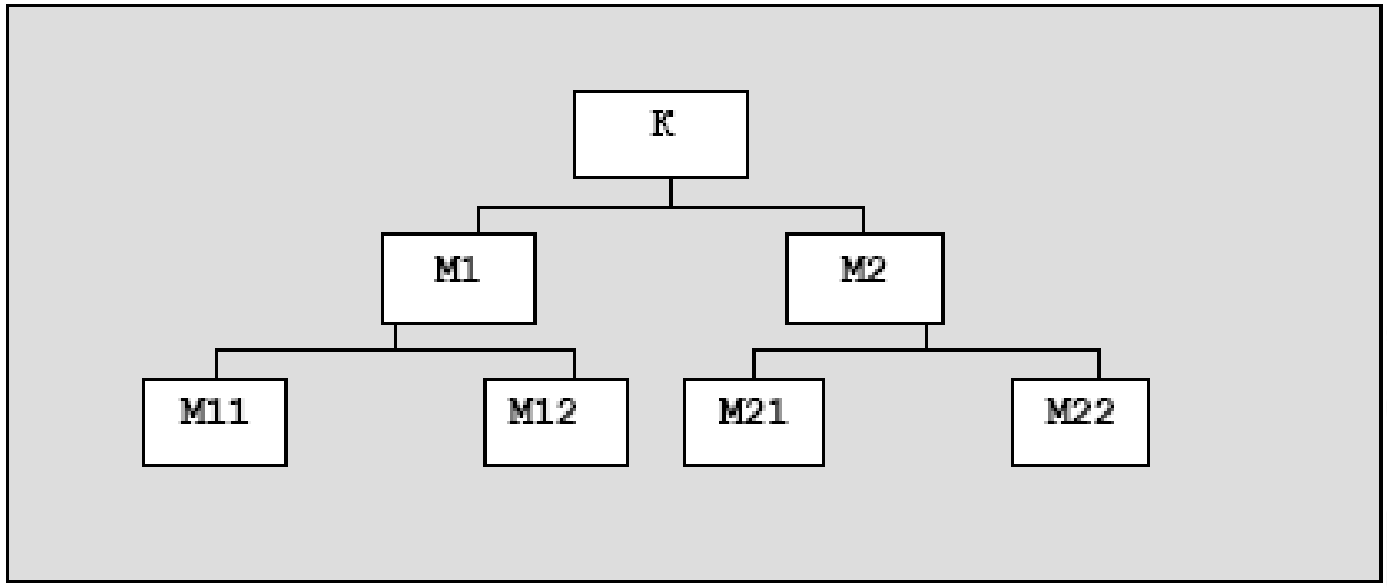
Интеграционное (integration testing) — это проверка наличия/отсутствия проблем в интерфейсах и взаимодействии между интегрируемыми компонентами и/или системами .

Основная задача интеграционного тестирования – поиск дефектов, связанных с ошибками в реализации и интерпретации интерфейсного взаимодействия между модулями.

Выделяют два способа интеграционного тестирования: *монолитный* (метод большого взрыва) и *пошаговый* (инкрементальный).

- Монолитный. Объединяются все элементы. Хотя для этого метода не требуются ни заглушки, ни драйверы, с его помощью значительно сложнее установить ошибки в интерфейсе.
- Пошаговый (помодульный) характеризуется наращиванием комплекса программ с **пошаговым тестированием** собираемого комплекса.
  - "Сверху вниз" и соответствующее ему нисходящее тестирование.
  - "Снизу вверх" и соответственно восходящее тестирование.

Пошаговое тестирование



Сверху вниз (нисходящее). Высокоуровневые подпрограммы становятся драйверами для остальных элементов. В этом методе используются заглушки, имитирующие поведение низкоуровневых элементов.

К; (заглушки)

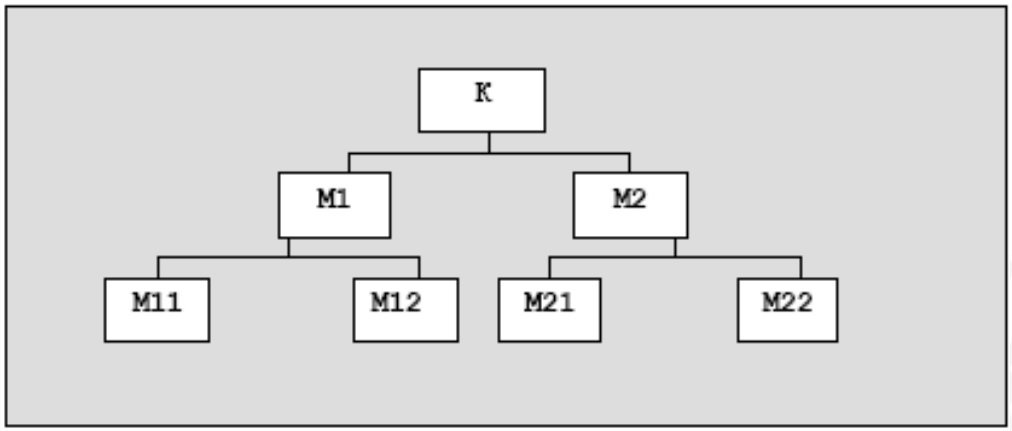
К->М1 (заглушки); К->М2 (заглушки);

К->М1->М11; К->М1->М12 ; К->М2->М12; К->М2->М22

Недостатки нисходящего тестирования

- Проблема разработки достаточно "интеллектуальных" заглушек, т.е. заглушек, способных к использованию при моделировании различных режимов работы комплекса, необходимых для тестирования.
- Сложность организации и разработки среды для реализации исполнения модулей в нужной последовательности.
- Параллельная разработка модулей верхних и нижних уровней приводит к не всегда эффективной реализации модулей из-за подстройки (специализации) еще не тестированных модулей нижних уровней к уже оттестированным модулям верхних уровней.

Восходящее тестирование



Снизу вверх. Тесты и элементы объединяются снизу вверх. Для такого метода необходимо использовать драйверы, чтобы вызывать тестируемые элементы.

М1 (драйвер)	М2 (драйвер)	М1->К
М11->М1	М21->М2	М2->К
М12 ->М1	М22 ->М2	

Недостатки восходящего тестирования

- Запаздывание проверки концептуальных особенностей тестируемого комплекса
- Необходимость в разработке и использовании драйверов

Преимущества	*Недостатки*
Нисходящее тестирование	
1. Имеет преимущества, если ошибки, главным образом, в верхней части программы.	1. Необходимо разрабатывать модули-заглушки, которые часто оказываются сложнее, чем кажется вначале.
2. Раннее формирование структуры программы позволяет провести ее демонстрацию пользователю и служит моральным стимулом.	2. Может оказаться трудным или невозможным создать тестовые условия.
	3. Сложнее оценка результатов тестирования.
	4. Стимулируется незавершение тестирования некоторых модулей.
Восходящее тестирование	
1. Имеет преимущества, если ошибки, главным образом, в модуле нижнего уровня.	1. Программа как единое целое не существует до тех пор, пока не добавлен последний модуль.
2. Легче создавать тестовые примеры.	
3. Проще оценка результатов.	

- Монолитное тестирование требует больших затрат труда.
- Монолитный способ применяется чтобы ускорить сроки сдачи программы.
- Монолитное тестирование предоставляет большие возможности распараллеливания работ особенно на начальной фазе тестирования.
- При монолитном - меньше расход машинного времени.
- При пошаговом тестировании раньше обнаруживаются ошибки в межмодульных связях.
- При пошаговом тестировании ошибки в межмодульных связях обнаруживаются легче.

Системное тестирование

Цели системного тестирования:

- определить, функционирует ли окончательный продукт соответствующим образом,
- решить, готова ли конечная система к выпуску.

На этом уровне также тестируются интерфейсы к внешним приложениям, аппаратному обеспечению, операционной среде и т.д.

При тестировании на уровне системы среда тестирования должна быть как можно ближе к среде развертывания.

Чтобы систематическим образом перебирать существенно отличающиеся друг от друга ситуации, используют *критерии полноты тестирования*

Чаще всего для определения критерия полноты некоторые из возможных тестовых ситуаций рассматривают как эквивалентные и определяют количество классов неэквивалентных тестовых ситуаций, встретившихся или «*покрытых*» во время тестирования. Такие критерии полноты называются *критериями тестового покрытия*

При этом определяется и числовая *метрика тестового покрытия* — доля покрытых классов ситуаций среди всех возможных.

Понятие покрытия. Уровни покрытия.

Покрытие кода — мера, используемая при тестировании ПО. Она показывает процент, насколько исходный код программы был протестирован.

- *покрытие операторов (строк)* — каждая ли строка исходного кода была выполнена и протестирована;
- *покрытие условий (ветвей)* — каждая ли точка решения (вычисления истинно ли или ложно выражение) была выполнена и протестирована;
- *Покрытие путей* — все ли возможные пути через заданную часть кода были выполнены и протестированы;
- *покрытие функций* — каждая ли функция программы была выполнена;
- *покрытие вход/выход* — все ли вызовы функций и возвраты из них были выполнены.

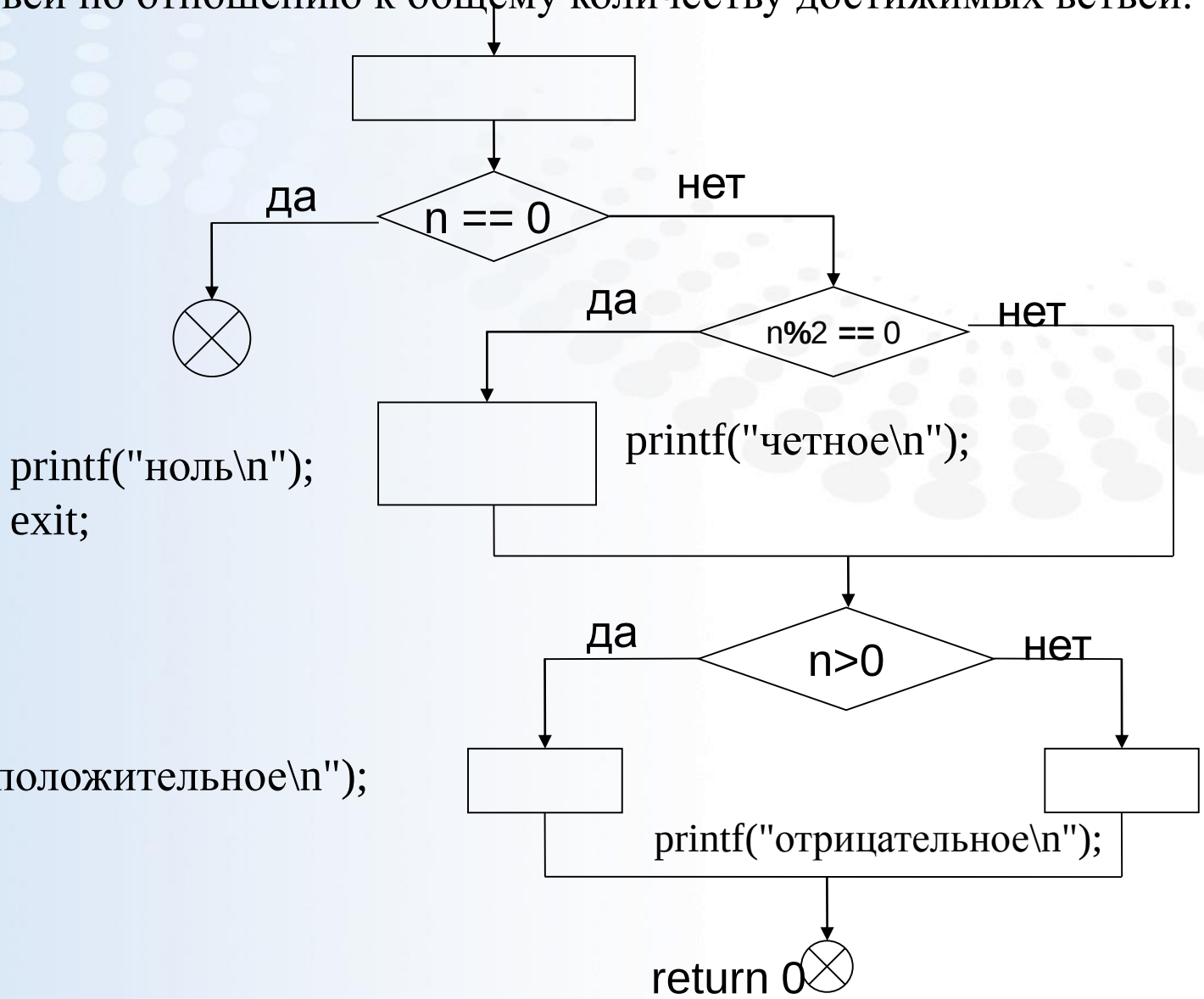
Покрытие операторов. Функция, которая должна по значению целого числа печатать характеристику — ноль это, четное или нечетное число, положительное или отрицательное. (0;2;-2)

```
 int chet(int n)
{
if (n==0)
{
printf(«ноль\n");
exit;
}
if(n%2==0)
printf(«четное\n");

if (n>0)
printf(«положительное\n");
else
printf(«отрицательное\n");

return 0;
}
```

Покрытие ветвей. Метрика покрытия ветвей — это доля покрытых в ходе теста ветвей по отношению к общему количеству достижимых ветвей.



0, 1, 2, –2

```
int chet(int n)
{
if (n==0)
{
printf(“ноль\n”);
exit;
}
if(n%2==0)
printf(“четное\n”);
else
printf(“нечетное\n”);

if ((n<0) && (n%2==0))
printf(“отрицательное\n”);
else
printf(“положительное\n”);

return 0;
}
```

Набор тестовых данных 0, 1, 2, –2, по-прежнему покрывающий 100% ветвей, не обнаруживает такую ошибку.

Чтобы выявить ее, необходимо использовать тесты, покрывающие комбинации условий, встречающихся в операторах ветвления.
Комбинации условий определяется следующим образом.
Выделим **условия всех ветвлений** в коде программы, определяемых условными операторами, операторами цикла или операторами выбора.
В нашем примере такими условиями будут: (n == 0), (n%2 == 0) и (n < 0). Составим комбинации из этих условий и их отрицаний
Из трех условий и их отрицаний можно составить 8 комбинаций

	n==0	n%2 == 0	n < 0
1	1	0	0
2	1	1	0
3	1	1	1
4	0	1	1
5	0	0	1
6	0	0	0
7	1	0	1
8	0	1	0

Не все эти комбинации будут выполнимы.
Так, не может быть одновременно выполнено (n == 0) и (n < 0),
или (n == 0) и не(n%2 == 0).

Номер	n == 0	n%2 == 0	n < 0
1	1	1	0
2	0	0	0
3	0	0	1
4	0	1	0
5	0	1	1

Комбинация значений элементарных условий покрывается тестом, если во время его выполнения эти условия в некоторый момент имеют в точности эти значения.

Метрика покрытия комбинаций условий определяется как доля покрытых текстами комбинаций значений элементарных условий, участвующих в условиях ветвлений программы, по отношению к общему количеству выполнимых комбинаций значений элементарных условий.

В приведенном выше примере набор тестовых данных 0, 1, 2, –2 не покрывает комбинацию условий с номером 3 в таблице. Покрыв эту ситуацию, например, с помощью значения параметра, равного –1, мы обнаружим внесенную в программу ошибку.

Метрика покрытия комбинаций условий строго сильнее метрики покрытия путей (ветвей).

Сложности:
В общем случае определить выполнимость комбинации значений произвольных формул оказывается достаточно сложно, поскольку для этого нужно знать смысл используемых в условиях переменных и функций.

Если в программе используется n операторов ветвления, условие каждого из них элементарно и между этими условиями нет никаких связей, то возникает всего лишь 2n ветвей, но 2^n возможных комбинаций условий.

Метрика покрытия условий и ветвей

Считается она следующим образом. Для каждого элементарного условия определяется, сколько значений оно может принимать при всех возможных сценариях выполнения программы.
Определяется общее число возможных значений элементарных условий, в которое каждое обычное условие вносит 2 значения, а постоянное условие — 1.
Метрика покрытия условий и ветвей вычисляется как отношение общего количества значений, которые все элементарные условия принимали в ходе теста, сложенного к количеством выполненных ветвей к сумме общего возможного числу возможных значений условий и числа достижимых ветвей

Модифицированное покрытия условий и ветвей (MC/DC).

Для обеспечения полного покрытия по этому методу необходимо выполнение следующих условий:

- каждое логическое условие должно принимать все возможные значения;
- каждая компонента логического условия должна хотя бы один раз принимать все возможные значения;
- должно быть показано независимое влияние каждой из компонент на значение логического условия, т.е. влияние при фиксированных значениях остальных компонент.

В ошибочном коде, рассмотренном выше, значение условия третьего ветвления (n < 0) && (n%2 == 0) может быть изменено за счет изменения значения любой из двух входящих в него формул.
Поэтому полный по MC/DC тестовый набор должен обеспечивать для этих формул выполнение комбинаций значений <1, 1>, <0, 1>, <1, 0>.

Анализ покрытия

Цель - выявление участков кода, которые не выполняются при выполнении тестовых примеров.

Анализ покрытия выполняется с учетом следующих соглашений:

- анализ должен подтвердить, что полнота покрытия тестами структуры кода соответствует требуемому виду покрытия и заданному минимально допустимому проценту покрытия;
- анализ полноты покрытия тестами структуры кода может быть выполнен с использованием исходного кода;
- анализ должен подтвердить правильность передачи данных и управления между компонентами кода.

Возможные формы отчетов о покрытии

Типичный отчет о покрытии представляет собой список структурных элементов покрываемого программного кода (функций или методов), содержащий для каждого структурного элемента следующую информацию:

1. название функции или метода;
2. тип покрытия (по строкам, по ветвям, MC/DC или иной);
3. количество покрываемых элементов в функции или методе (строк, ветвей, логических условий);
4. степень покрытия функции или метода (в процентах или в абсолютном выражении);
5. список непокрытых элементов (в виде участков непокрытого программного кода с номерами строк).

Модуль 7

- Функциональные виды тестирования
- Тестирование безопасности, тестирование взаимодействия

- Нефункциональные виды тестирования
- Тестирование производительности
- Особенности тестирования ООП

Виды тестирования

В зависимости от преследуемых целей виды тестирования можно условно разделить на следующие виды:

- Функциональные
- Нефункциональные
  - Тестирование производительности
  - Тестирование установки (инсталляционное)
  - Тестирование удобства пользователя
  - Тестирование на отказ и восстановление
  - Конфигурационное тестирование

Функциональное тестирование

Функциональное тестирование — проверка соответствия системы, предъявляемым к ней требованиям, описанным на уровне спецификации поведенческих характеристик. Функциональные требования определяют, что именно делает ПО, какие задачи оно решает.

- Функциональное тестирование
- Тестирование безопасности
- Тестирование взаимодействия

Устойчивость системы

- ***Устойчивость (robustness)*** – уровень, до которого компонент или система может функционировать корректно при наличии некорректных входных данных или функционирования в стрессовых условиях. [IEEE 610]
- Устойчивость к ошибкам – способность системы или компонента продолжать нормально функционировать, несмотря на присутствие неправильных входных данных. [IEEE 610]

Тестирование безопасности

Защищенность - способность ПП защищать информацию и данные так, чтобы неавторизованные субъекты или процессы не смогли читать или модифицировать их, а авторизованным пользователям и процессам не было отказано в доступе к ним.

Цель - проверка безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Основные принципы

Конфиденциальность. Предотвращение несанкционированного доступа к информации или программ.

Целостность. Предотвращение повреждения, искажения или изменения информации или программ.

Проверка подлинности (аутентификация). Удостоверение субъектов (пользователей или процессов) в сети перед предоставлением доступа к данным, а также удостоверение объектов перед их использованием.

Проверка полномочий (авторизация). Обеспечение доступа к данным только тем субъектам, которые были надлежащим образом авторизованы и имеют соответствующие права (на чтение, запись, изменение и т.д.).

Доступность. Обеспечение необходимой работоспособности и доступности данных и программ.

Подотчётность (в некоторых источниках — доказательство причастности, контроль). Установление связи между пользователем (или объектом) и действием (например, в случае электронных платёжных систем — подтверждение факта, что отправитель послал, а адресат получил некоторую сумму денег, а также опровержение этого факта, если транзакция, по каким-то причинам не состоялась).

Тестирование взаимодействия

Тестирование взаимодействия (**Interoperability Testing**) – это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости и интеграционное тестирование.

Тестирование производительности

Тестирование производительности —тестирование, которое проводится с целью определения, как быстро работает система или её часть под определенной нагрузкой.

Тестирование производительности

=

тестирование эффективности

+

тестирование надёжности

Пользовательский интерфейс:

0.1 секунды – приложение реагирует мгновенно

1 секунда – пользователь замечает задержку

10 секунд – предельное значение времени, в течение которого внимание пользователя сосредоточено на приложении

- Эффективность
  - Временная эффективность

    атрибуты ПО, относящиеся к временам отклика и обработки и к скорости выполнения его функций
  - Использование ресурсов

    атрибуты ПО, относящиеся к объему используемых ресурсов и продолжительности такого использования при выполнении функций
- Надёжность
  - Зрелость, завершенность (обратна к частоте отказов)
  - Устойчивость к отказам
  - Способность к восстановлению работоспособности при отказах

В тестировании производительности различают следующие направления:

- нагрузочное (load)
- стрессовое (stress)
- тестирование стабильности (endurance or soak or stability)
- объемное (volume)

Цели

1. Определение характеристик производительности системы
   (= эффективности + надёжности)
2. Проверка соответствия этих характеристик требованиям
3. Сравнение различных версий или конфигураций системы
4. Выявление «узких мест»

1. Анализ требований, дизайна, реализации
2. Дизайн тестов
3. Калибровка и отладка тестов
4. Выполнение тестов на стенде, измерения
5. Анализ результатов
6. * Предпродуктивное тестирование
7. * Постпродуктивное тестирование
8. * Мониторинг в режиме продуктивной эксплуатации

Анализ дизайна, реализации

- Определение способов воздействия на систему и регистрации реакций
  - синхронные и асинхронные запросы
  - параллельная обработка запросов
  - очереди запросов
  - журналирование
  - интерфейсы мониторинга
  - интерфейсы с другими системами
- Прогнозирование возможных проблем

Дизайн тестов

- Классификация нагрузки
  - Одиночные запросы
  - Запросы идут «звеньями»
  - Непрерывный поток или волны
  - Массированная атака
  - Разные виды запросов
  - Скорость движения, время нахождения в «очереди на обслуживание»

- Моделирование нагрузки

  - количеством виртуальных пользователей

  - с какой интенсивностью, будет выполняться каждый сценарий

- Разработка тестовых сценариев

  – запись заготовок сценариев (если возможно)

  – написание сценариев или доводка заготовок

  – параметризация сценариев

  – корреляция действий в рамках сценария

- Разработка профилей нагрузки

Профиль – совокупность *сценариев*, тестовых данных, и описание способа запуска сценариев

  - какие сценарии запускать
  - параллельный и последовательный запуск
  - количество экземпляров сценариев, изменяется в зависимости от времени
  - тестовые данные, изменяются в зависимости от времени
  - общая продолжительность
  - асинхронные и синхронные действия
  - распределение промежутков между действиями в сценариях

Калибровка и отладка тестов

Калибровка – запуск на «минимальном» профиле с целью убедиться, что:

- тестируемая система функционирует
- тесты работают
- мониторы измеряют

Разновидности ТП

- Монотонно возрастающая нагрузка
  - «тестирование масштабируемости»
- Постоянная средняя нагрузка
  - «традиционное ТП»
- Постоянная предельная нагрузка
  - «тестирование живучести»
- Постоянная запредельная нагрузка
  - «тестирование устойчивости»
- Всплески запредельных нагрузок
  - «стрессовое тестирование»

Тестирование производительности

Тестирование производительности (Performance testing) Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит: измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций определение количества пользователей, одновременно работающих с приложением определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)

Нагрузочное тестирование

Нагрузочное тестирование обычно проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой.

Этой нагрузкой может быть, например, ожидаемое количество одновременно работающих пользователей приложения, совершающих заданное число транзакций за интервал времени.

В случае наблюдения за базой данных, серверных приложений, сетью и т.д., этот тип тестирования может также идентифицировать некоторые узкие места приложения.

Стрессовое тестирование

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса.

Всплески запредельных нагрузок:

- Короткий всплеск
- Средний всплеск (до насыщения очередей)
- Длительный всплеск
- Резкий или плавный всплеск

Объемное тестирование

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит: измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций может производиться определение количества пользователей, одновременно работающих с приложением .

Монотонно возрастающая нагрузка

- Цель:
  - поиск «пресыщения»
  - определение предельных возможностей системы
  - определение «узких мест»
- Варианты:
  - линейное возрастание
  - логарифмическое возрастание
  - непрерывное или «уступами»

Тестирование стабильности

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Времена выполнения операций могут играть в данном виде тестирования второстепенную роль.

Тестирование инсталляции

Включает тестирование

- сценариев первичной инсталляции системы,
- сценариев повторной инсталляции,
- деинсталляции,
- инсталляции в условиях наличия ошибок в инсталлируемом пакете, в окружении или в сценарии.

Тестирование пользовательской документации

Включает проверку

- полноты и понятности описания правил и особенностей использования продукта,
- наличие описания всех сценариев и функциональности,
- синтаксис и грамматику языка,
- работоспособность примеров.

Модуль 8

- Регрессионное тестирование
- Тестирование пользовательского интерфейса
- Тестирование удобства пользования
- Тестирования web-приложений

Регрессионное тестирование

- Регрессионное тестирование – это любой вид тестирования, который преследует цель обнаружение регресса системы.
- Регрессионное тестирование – это тестирование, предназначенное для повторной проверки свойств приложения или продукта с целью убедиться в том, что после внесения изменений или добавления новых возможностей приложение по-прежнему работает.

Когда проводить? При любых изменениях:

- Изменения в коде
- Изменение настроек, конфигурационных файлов
- При эксплуатации - накапливается «мусор» (в оперативной памяти, в БД…)

Повторение, но не регрессионное

- Приёмочное тестирование
- Конфигурационное тестирование
  - тестирование в разных окружениях
  - тестирование с разными настройками
- Тесты с элементами «стохастичности»
- Мутационное тестирование

Поводы для повторного выполнения тестов

Рациональные поводы:

- **Тесты могут снова сработать (перезарядка)**
- «Плавающие» дефекты
- Недоверие к предыдущему запуску
- Увеличение пула тестовых данных
- **Сравнение характеристик (например, производительность)**

«Иррациональные» поводы:

- Это дёшево
- Ошибка появляется снова и снова
- Очень (!) важно, чтобы ошибки не было
- Цель – не тестирование

Повторное выполнение тестов

- «Парадокс пестицида»
- Метафора «минного поля»

Полный набор

«Иррациональный» критерий полноты:

- Все тесты, которые когда-либо были придуманы и выполнены

Рациональные критерии полноты:

- Обеспечивающий покрытие требований
- Обеспечивающий покрытие кода
- Обеспечивающий отлов любых мутаций

Распространённые «практики»

- Простое накопление тестов
  - результат – «гора мусора»
- Создание тестов для каждого дефекта, запроса на изменение, патча
  - результат – «лоскутное одеяло»

Как с этим бороться?

- Продумывать архитектуру тестового набора
- Минимизировать тестовый набор
  - выбрасывать лишние тесты
  - рефакторинг тестов
- Делать разные наборы тестов в соответствии с разными критериями полноты

Минимизация тестового набора

- Лишний тест – не увеличивающий покрытие по какому-либо критерию

- Рефакторинг тестов
  - похожие тесты можно «склеить» (пример – одинаковые действия, но разные проверки)
  - крупные тесты можно «раздробить», лишние куски выкинуть
  - мелкие тесты можно «склеить»

Разные наборы тестов

- Тесты для изменившихся частей
- Тесты для частей, зависящих от изменившихся частей

- Тесты для критичной функциональности
- Тесты для основной функциональности
- Тесты определённого типа
- Наборы с иными заданными характеристиками

Регрессионное тестирование

- **Недавнее** (Recent)
- **Основное** (Core)
- **Рисковое** (Risky)
- **Чувствительное к конфигурации** (Configuration sensitive)
- **Исправленное** (Repaired)
- **Хроническое** (Chronic)

НОРЧИХ

НОРЧИХ

Недавнее

Что было недавно добавлено в приложение? Недавние изменения – от очевидных до едва заметных – возможная причина появления дефектов. Не забудьте проанализировать запросы на изменение требований, сообщения об ошибках, изменения модели данных и относящуюся к делу внутреннюю переписку.

Основное

Определите области приложения, которые обязаны работать «в любом случае», и вы получите ключевые функциональные возможности. Вспомните значения слова «регресс»: движение назад, упадок, ухудшение. Это то, что должно предотвращать регрессионное тестирование.

Рисковое

Вспомнить области повышенного риска, имеющиеся в приложении – будь то новые функциональные возможности или старые.

Чувствительное к конфигурации

Код, который зависит от параметров окружения, т.е. чувствительный к конфигурации, более уязвим.

Исправленное

Некоторые функции никак не удается реализовать или поправить с первой попытки, их постоянно сопровождает шлейф дефектов, для устранения которых приходится выпускать несколько внутренних релизов.

Хроническое

Если что-то может сломаться, оно обязательно сломается. Нет ли у продукта областей, в которых часто возникают проблемы?

Тестирование пользовательского интерфейса (UI)

Интерфейс пользователя (UI) - это часть программы, которая находится на виду у пользователя и призвана обеспечивать отображение данных, управление или диалог с пользователем.

На процесс проектирования пользовательского интерфейса набольшее влияние оказывают субъективные представления проектировщика о понятности, удобстве и красоте.

- Какая информация необходима пользователю для решения задачи?
- Какую информацию пользователь может игнорировать (не учитывать)?
- Совместно с пользователем разделить всю информацию на сигнальную, отображаемую, редактируемую, поисковую и результирующую.
- Какие решения пользователю необходимо принимать в процессе работы с программой?
- Может ли пользователь совершать несколько различных действий (решать несколько задач) одновременно?
- Какие типовые операции использует пользователь при решении задачи?
- Что произойдет, если пользователь будет действовать не по предписанному Вами алгоритму, пропуская те или иные шаги или обходя их?

Критерии качественного интерфейса

- Лучше тот интерфейс, при котором время выполнения задачи меньше;
- Лучше тот интерфейс, в котором число непроизвольных ошибок пользователя меньше;
- Неоднозначность в понимании интерфейса должна быть минимальна (это способствует самообучению пользователей и делает их поведение предсказуемым);
- Необходима высокая стандартизация интерфейса (она облегчает обучение пользователей);
- Объем вводимой пользователем информации должен стремиться к минимуму (одни и те же данные не должны вводиться несколько раз);
- Простота и визуальная привлекательность (удобство использования не менее важно, чем функциональность).

Тестирование графического интерфейса пользователя

Виды тестирования графического интерфейса пользователя:

- Тестирование на соответствие стандартам графических интерфейсов;
- Тестирование с различными разрешениями экрана;
- Тестирование в ограниченных условиях, например, в условиях нехватки памяти;
- Совместимость с различными Интернет-браузерами;
- Тестирование локализованных версий: точность перевода, проверка длины названий элементов интерфейса и т.д.;
- Тестирование графического интерфейса пользователя на целевых устройствах (для КПК-совместимых приложений).

Функциональное тестирование GUI

- Анализ требований к пользовательскому интерфейсу;
- Разработка тест-требований и тест-планов для проверки пользовательского интерфейса;
- Выполнение тестовых примеров и сбор информации о выполнении тестов;
- Определение полноты покрытия пользовательского интерфейса требованиями;
- Составление отчетов о проблемах в случае несовпадения поведения системы и требований либо в случае отсутствия требований на отдельные интерфейсные элементы.

Требования к пользовательскому интерфейсу могут быть разбиты на две группы:

- требования к внешнему виду пользовательского интерфейса и формам взаимодействия с пользователем;

- требования по доступу к внутренней функциональности системы при помощи пользовательского интерфейса.

Требования к размещению элементов управления на экранных формах

Требования могут определять общие принципы размещения элементов пользовательского интерфейса или требования к размещению конкретных элементов.

Пример:

Каждое окно приложения должно быть разбито на три части: строка меню, рабочая область и статусная строка. Строка меню должна быть горизонтальной и прижатой к верхней части окна, статусная строка должна быть горизонтальной и прижатой к нижней части окна, рабочая область должна находиться между строкой меню и статусной строкой и занимать всю оставшуюся площадь окна.

Примером требований по размещению конкретного элемента может служить следующее:
Кнопка "Начать передачу" должна находиться непосредственно под строкой меню в левой части рабочей области окна.

При тестировании необходимо определить:
- сохраняется ли расположение элемента при изменении размера окна,
- сохраняется ли расположение элемента при использовании элемента (в данном случае - при нажатии).

Тестирование удобства пользовательского интерфейса (Usability Testing)

Тестирование удобства пользования - это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. [ISO 9126]

Характеристики

- Наблюдаемость состояния системы.
- Соотнесение с реальным миром.
- Пользовательское управление и свобода действий.
- Целостность и стандарты.
- Помощь пользователям в распознавании, диагностике и устранении ошибок.
- Предотвращение ошибок.
- Распознавание, а не вспоминание.
- Гибкость и эффективность использования.
- Эстетичный и минимально необходимый дизайн.
- Помощь и документация.

Оценка уровня удобства

- **Производительность, эффективность** (efficiency) - сколько времени и шагов понадобится пользователю для завершения основных задач приложения, например, размещение новости, регистрации, покупка и т.д.? (*меньше - лучше*)
- **Правильность** (accuracy) - сколько ошибок сделал пользователь во время работы с приложением? (*меньше - лучше*)

- **Активизация в памяти** (recall) – как много пользователь помнит о работе приложения после приостановки работы с ним на длительный период времени? (*повторное выполнение операций после перерыва должно проходить быстрее чем у нового пользователя*)
- **Эмоциональная реакция** (emotional response) – как пользователь себя чувствует после завершения задачи - растерян, испытал стресс? Порекомендует ли пользователь систему своим друзьям? (*положительная реакция - лучше*)

Уровни тестирования

- По отношению к готовому продукту, посредством тестирования черного ящика.
- К интерфейсам приложения (API), используемым при разработке - тестирование белого ящика (white box testing).
  - Проверяется удобство использования внутренних объектов, классов, методов и переменных.
  - Рассматривается удобство изменения, расширения системы и интеграции ее с другими модулями или системами.

Заблуждения о тестировании удобства пользования

- Тестирование пользовательского интерфейса = Тестирование удобства пользования
- Тестирование удобства пользования можно провести без участия эксперта

В Usability-тестировании существует метод, построенный на интервьюировании пользователей по заранее составленному сценарию, и направленный на выяснение того, как пользователи используют продукт.

Смысл метода состоит в наблюдении за тем, с какими трудностями сталкиваются пользователи, работая с продуктом.

Целью тестирования является улучшение интерфейсов так, чтобы целевая аудитория продукта наиболее эффективно взаимодействовала с финальной версией продукта.

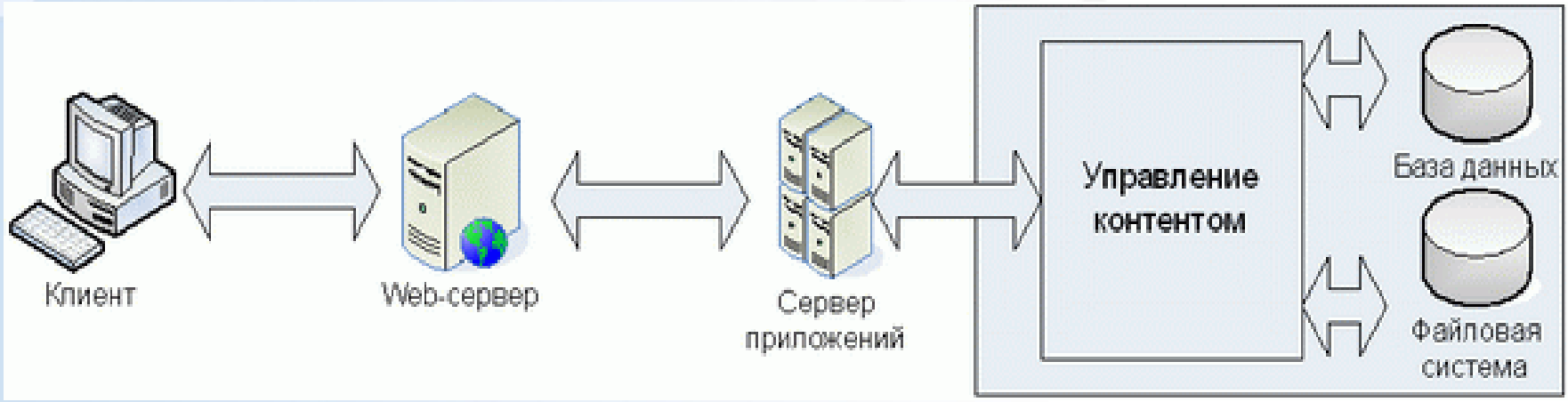
- Однозначно понятные
- Сформулированы полностью
- При этом достаточно короткие
- Не содержащие в себе подсказок
- Содержащие точку начала выполнения задания
- Такими, чтобы респондент должен был сначала решить нужно ли в данное время выполнять это действие

Тестирование web-приложений

Web-приложениями будем называть любые приложения, предоставляющие Web-интерфейс

Web-приложения в первую очередь характеризуются тем, что их пользовательский интерфейс имеет стандартизированную архитектуру, в которой:

1) для взаимодействия с пользователем используется Web-браузер;
2) взаимодействие с пользователем четко разделяется на этапы, в течение которых браузер работает с одним описанием интерфейса;
3) эти этапы разделяются однозначно выделяемыми обращениями от браузера к приложению;
4) для описания интерфейса применяется стандартное представление ((HTML);
5) коммуникации между браузером и приложением осуществляются по стандартному протоколу.



Фундаментальные суждения о тестировании Web – приложений

- Когда мы видим ошибку со стороны клиента, то мы видим симптом ошибки, но не ее саму.

- Ошибки бывают зависимыми от среды и могут не возникать в различных средах.

- Ошибки могут быть в коде или в конфигурации.

- Ошибки могут постоянно находиться на любом из нескольких уровней.

Подходы к тестированию Web-приложений

- Функциональное тестирование
- Тестирование пользовательского интерфейса
- Тестирование удобства использования;
- Нагрузочное тестирование
- Проверка ссылок и HTML-кода
- Тестирование безопасности

Проверка ссылок и HTML-кода

- **Проверка ссылок** – актуальна для внутренних ссылок (в случае больших и разветвленных порталов) и для внешних – если это, к примеру, каталог сайтов, или страница «Ссылки»

- **Проверка HTML-кода страниц** – проверка корректности HTML-кода, в том числе на соответствие стандартам

Тестирование безопасности

Тестированию подвергается не только сам конкретный сайт или веб-приложение, а весь сервер полностью

Программа «прикидывается» реальным пользователем-взломщиком и пытается применить к серверу все известные ей методы атаки и проверяет все уязвимости

Результатом работы будет отчет о найденных уязвимостях и рекомендации по их устранению

Нагрузочное тестирование

- Измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций;
- Определение количества пользователей, одновременно работающих с приложением;
- Определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций);
- Исследование производительности на высоких, предельных, стрессовых нагрузках

Проверка HTML-кода

Для такого рода тестирования написано множество утилит, проверяющих весь сайт на соответствие стандартам (а некоторые валидаторы могут в автоматическом режиме исправлять найденные недочеты, например, пропущенные закрывающие теги и т.д.).
Часто такие средства встраивают в Веб-редакторы, существуют браузеры с встроенными валидаторами.

Модуль 9

- Жизненный цикл ПО
- Методологии разработки ПО
- Риски

Жизненный цикл разработки ПО

Жизненный цикл (ЖЦ) ПО - это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Пять основных этапов жизненного цикла программного продукта.

- Планирование
- Проектирование
- Кодирование и написание документации
- Тестирование и исправление недостатков
- Сопровождение (после выпуска) и усовершенствование

Жизненный цикл ПО

Фазы жизненного цикла:

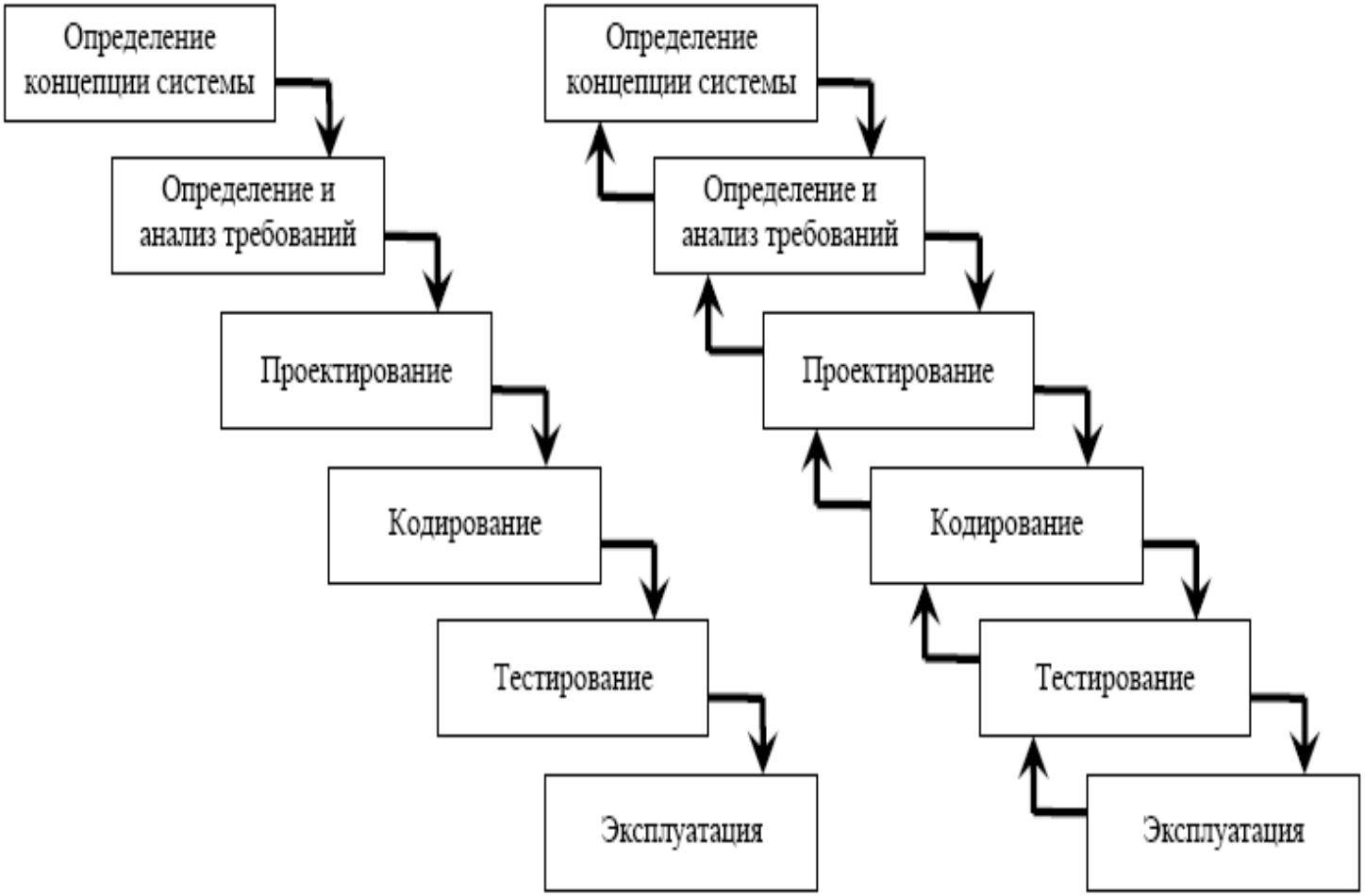
- концепция (инициация, идентификация, отбор)
- определение (анализ)
- выполнение (практическая реализация или внедрение, производство и развертывание, проектирование или конструирование, сдача в эксплуатацию, инсталляция, тестирование и т.п.)
- закрытие (завершение, включая оценивание после завершения)

В общем случае, *жизненный цикл определяется моделью* и *описывается в форме методологии(метода)*.

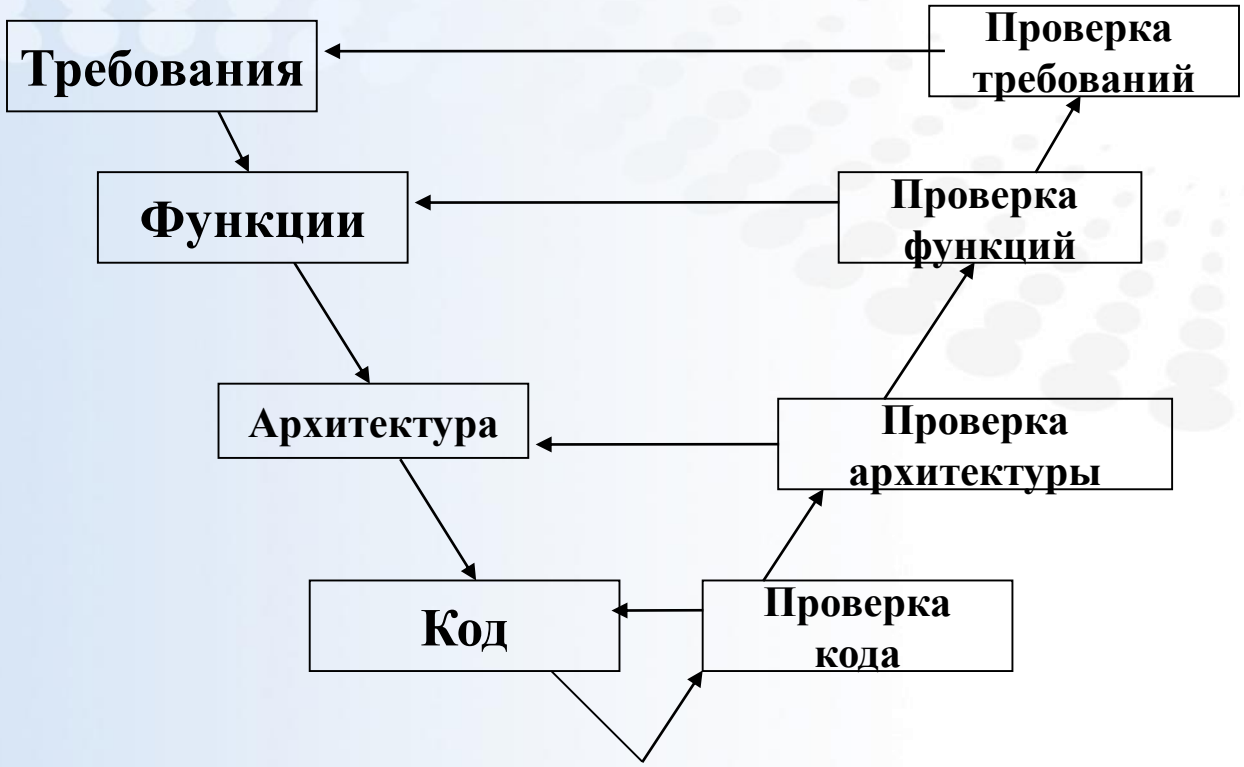
- **Методология (метод)** задает комплекс работ, их детальное содержание и ролевую ответственность специалистов на всех этапах выбранной модели жизненного цикла.

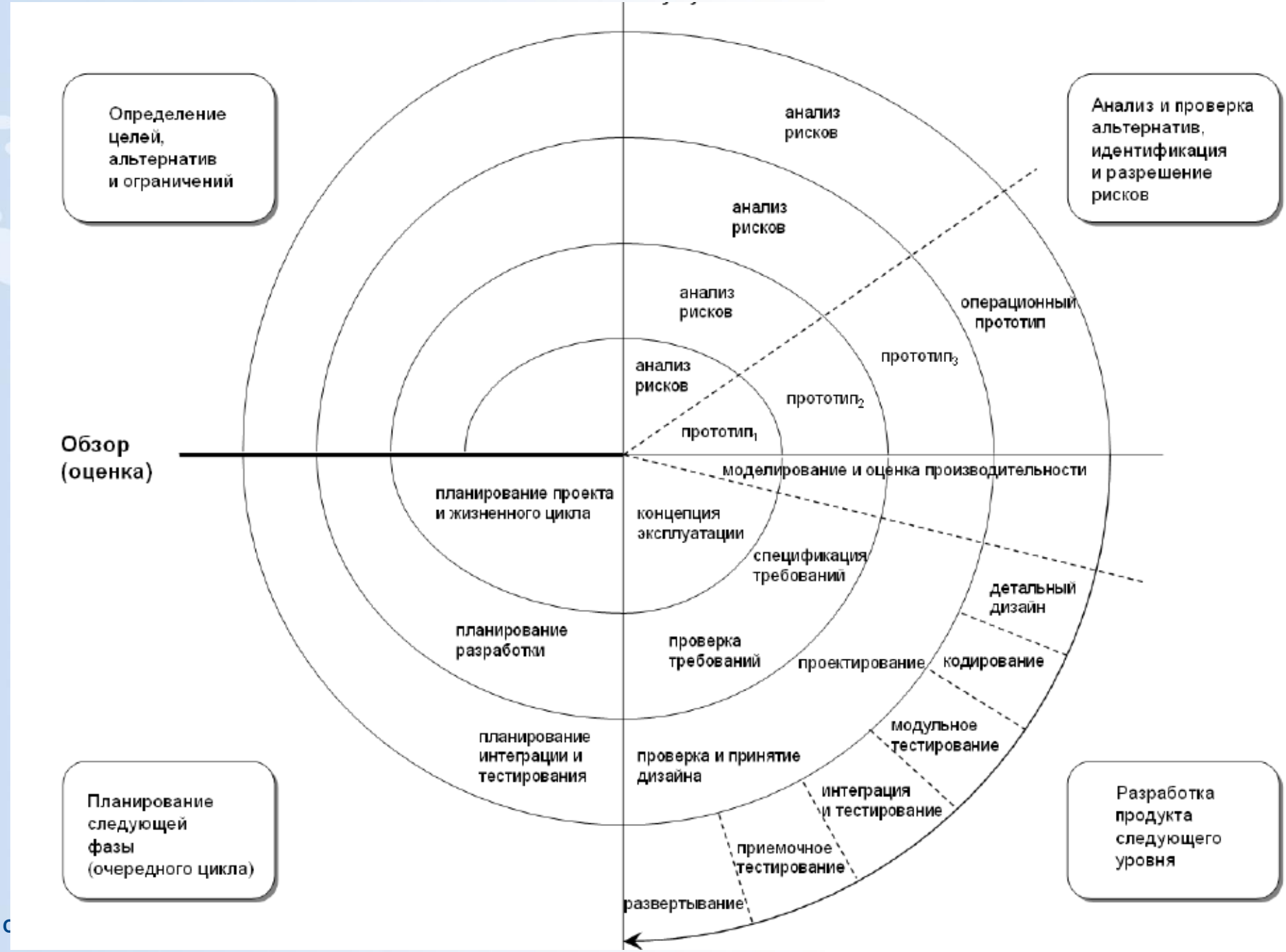
КАСКАДНЫЙ ЖИЗНЕННЫЙ ЦИКЛ

В 1970 году У. У. Ройс (*W. W. Royce*) описал в виде концепции то, что сейчас принято называть «модель водопада». В модели водопада Ройса, следующие фазы шли в таком порядке:



V-ОБРАЗНЫЙ ЖИЗНЕННЫЙ ЦИКЛ




Определение целей, альтернатив и ограничений
Анализ и проверка альтернатив, идентификация и разрешение рисков
Планирование следующей фазы (очередного цикла)
Разработка продукта следующего уровня
Обзор (оценка)
анализ рисков
анализ рисков
анализ рисков
анализ рисков
прототип1
прототип2
прототип3
операционный прототип
планирование проекта и жизненного цикла
концепция эксплуатации
спецификация требований
моделирование и оценка производительности
планирование разработки
проверка требований
проектирование
детальный дизайн
кодирование
планирование интеграции и тестирования
проверка и принятие дизайна
модульное тестирование
интеграция и тестирование
приемочное тестирование
развертывание

Методология MSF

Microsoft Solutions Framework (MSF) - это методология ведения проектов и разработки решений, базирующаяся на принципах работы над продуктами самой фирмы Microsoft и предназначенная для использования в организациях, нуждающихся в концептуальной схеме для построения современных решений

Жизненный цикл процессов в MSF сочетает водопадную и спиральную модели разработки: проект реализуется поэтапно, с наличием соответствующих контрольных точек, а сама последовательность этапов может повторяться по
спирали

Тестирование

Цель: Одобрение выпуска продукта только лишь после того, как все дефекты выявлены и улажены.

Область компетенций:

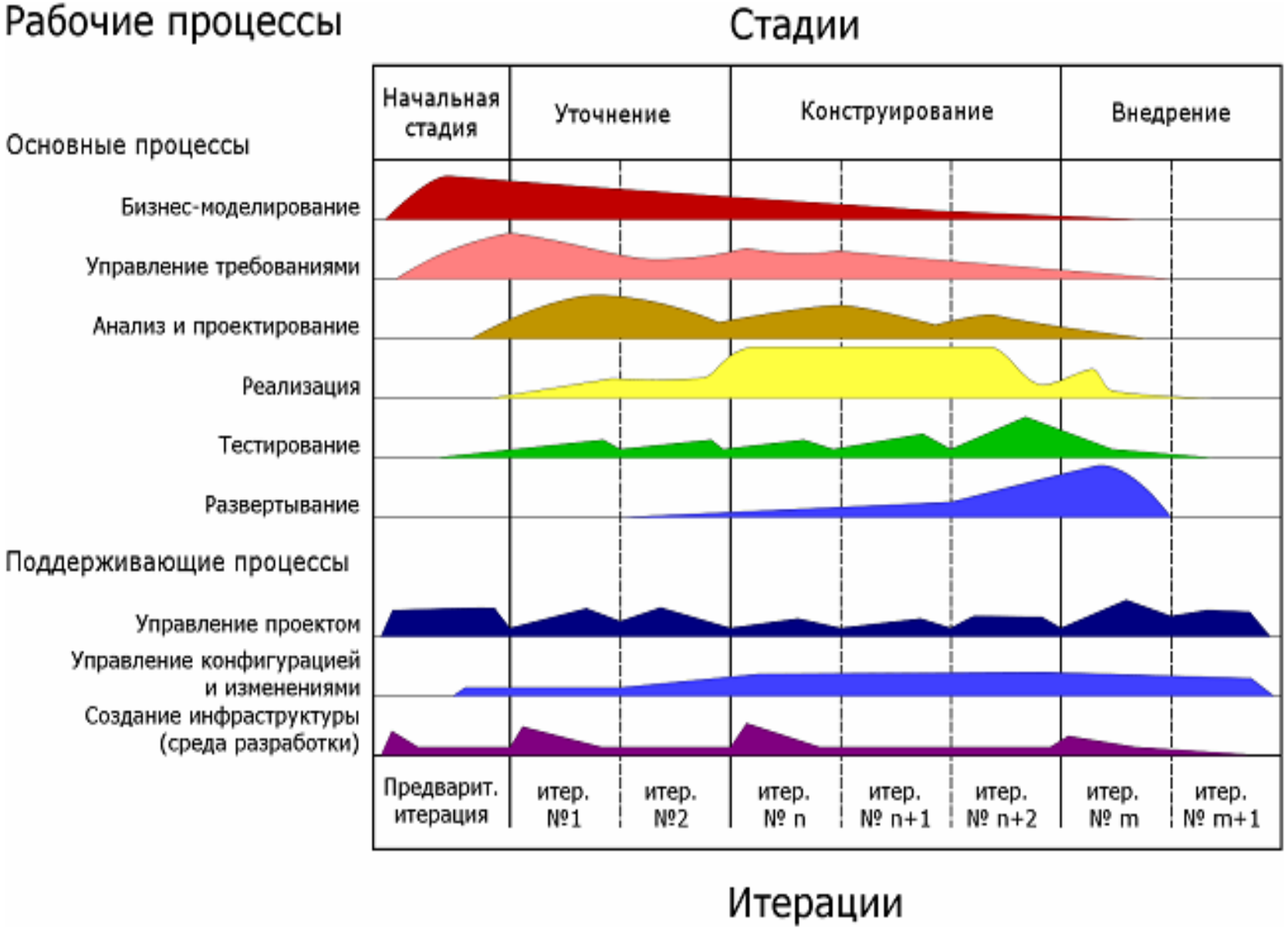
- Планирование тестов;
- Разработка тестов;
- Отчетность по тестам.

Rational Unified Process

RUP был создан в 1996г. корпорацией Rational при участии Гради Буча, Айвара Якобсона и Джима Румбаха.

RUP обеспечивает строгий подход к распределению задач и ответственности внутри организации-разработчика. Основная идея RUP — максимально четко распределить работу каждого участника процесса разработки.

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.



Роль тестировщика

Роль тестировщика состоит в обеспечении разработчиков и руководства объективной информацией о качестве продукта на основе критериев:

- Соответствие стандартам.
- Выявление дефектов.
- Воспринимаемое (ожидаемое) качество.

Миссии тестирования – соответствует целям фазы:

- Оценить соответствие спецификации.
- Подтвердить соответствие стандарту.
- Найти как можно больше дефектов.
- Быстро определить существующие проблемы.
- Определить воспринимаемые риски для качества.

Экстремальное программирование

Методология XP была была создана Кентом Беком (Kent Beck) в 1996 году

Экстремальное программирование – методология быстрой разработки программного обеспечения. Состоит из набора методик и принципов, позволяющих как по отдельности, так и в комплексе, оптимизировать процесс разработки.

Эта упрощенная методика организации производства для небольших и средних по размеру команд специалистов (2-10 программистов), занимающихся разработкой программного продукта в условиях неясных или быстро меняющихся требований.

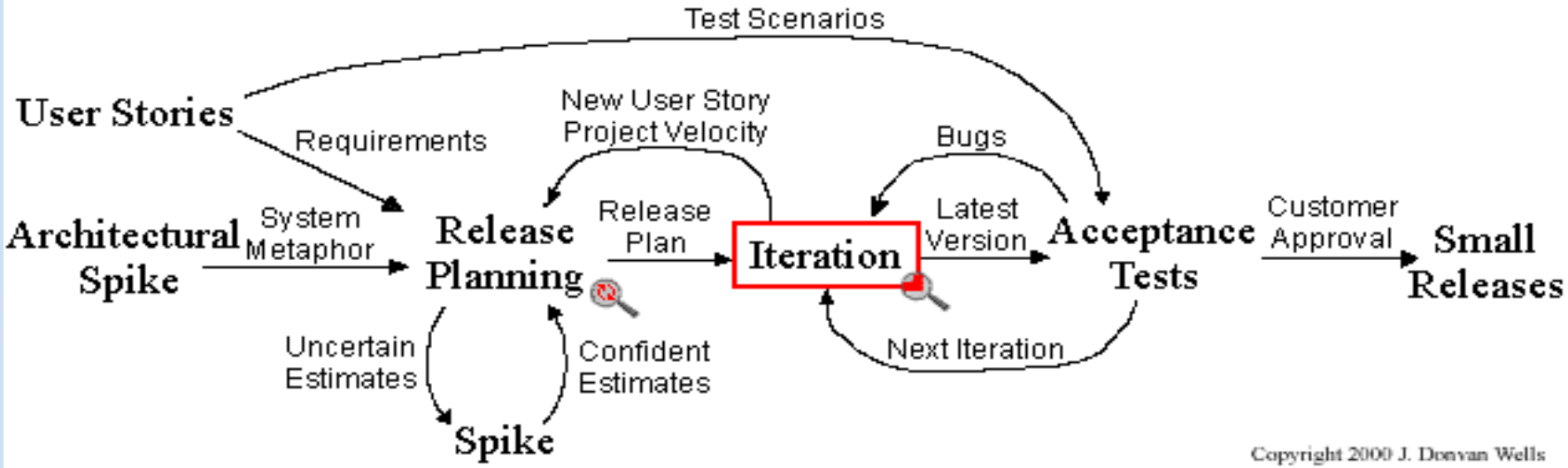
Экстремальное программирование определяет кодирование как ключевую и основополагающую деятельность при работе над программным проектом.

В основе экстремального программирования — очень короткий, постоянно повторяющийся цикл разработки, составляющий одну-три недели.

К концу каждого цикла вы должны иметь полностью рабочий, функциональный и протестированный релиз приложения.

Эти циклы должны быть повторяющимися и бесперебойными на протяжении всего проекта.

Extreme Programming Project



Короткий цикл обратной связи

- **Разработка через тестирование**
- Игра в планирование
- Заказчик всегда рядом
- **Парное программирование**

Непрерывный, а не пакетный процесс

- Непрерывная интеграция
- Рефакторинг
- Частые небольшие релизы
- Понимание, разделяемое всеми
- Простота
- Метафора системы

Коллективное владение кодом или выбранными шаблонами проектирования

- Стандарт кодирования

Социальная защищенность программиста

- 40-часовая рабочая неделя

Парное программирование – метод, при котором два разработчика работают над созданием кода вместе, на одной рабочей машине.

Тесты вперед – написание и реализация прецедентов использования перед написанием кода.

Риски в тестировании ПО

Риск — это существующий или развивающийся фактор процесса, который обладает потенциально негативным воздействием на процесс.

Риск это то, что *может случиться* и привести к негативным последствиям, а проблема, это то, что *уже случилось* и мешает работать.

Пять основных составляющих управления риском:

идентификация риска: первоначальный мозговой штурм по выявлению риска, последующая сортировка и определение какого-либо механизма для обеспечения постоянного действия данного процесса.

анализ воздействия риска: количественная оценка каждого риска в терминах вероятности его наступления и потенциального ущерба.

- *планирование реагирования на риски:* что вы собираетесь делать, если и когда данный риск наступит.

- *ослабление риска:* меры, которые должны быть приняты предварительно, чтобы обеспечить возможность и эффективность проведения запланированных действий, если они потребуются.

- *мониторинг и управление рисками:* отслеживание рисков, выделенных в качестве объектов управления, выявление материализации рисков.

Пример:

Риск - выход из строя разработческого сервера вследствие стихийных бедствий.

Вероятность проявления - низкая

Обоснование: работы ведутся в средней полосе России. Это не сейсмоопасный район, так же по близости нет крупных водоемов, подверженных штормам и цунами и т.п.

Степень влияния на результат - низкая

Обоснование: офис компании находится в железобетонном бункере на глубине 50 метров под землей.

Вывод - можем проигнорировать этот риск.

Действия: нет необходимости в проведении привентивных мероприятий.

Пример:

Риск - не соответствие реализации требованиям.

Вероятность - средная

Обоснование: тестирование ведется без подробной спецификации продукта, такая же ситуация и у разработчиков. Функционал разрабатывается со слов бизнес аналитика.

Степень влияния на результат - высокая

Обоснование: выявить не соответствие может только заказчик, что повлечет необходимость доработки или переделки готовой части продукта.

Вывод - при средней вероятности возникновения расхождений между спецификацией и реализацией степень влияния данного факта очень высока, что безусловно выльется в дополнительные расходы по доработке.

Действия: поставить менеджера в известность.

Где есть неопределенность, там появляется риск.

1. Требования к системе: Что именно должна делать система?

2. Обеспечение стандартов взаимодействия: Как будет система взаимодействовать с людьми-операторами и другими системами того же уровня?

3. Влияние изменяющейся среды: Как во время разработки будут изменяться потребности и цели?

4. Ресурсы: Какие ключевые навыки и знания исполнителей возможно будет (при необходимости) привлечь по мере продвижения работы над проектом?

5. Управление: Хватит ли у руководства таланта, чтобы создать эффективные команды, поддерживать боевой дух, обеспечивать низкую текучесть кадров и координировать сложные комплексы взаимосвязанных задач?

6. Сеть поставок: Будут ли другие участники проекта действовать так, как ожидалось?

7. Политика: Каков может быть результат использования политической силы для навязывания ограничений, несовместимых с успехом проекта?

8. Конфликты: Как различные участники проекта найдут компромисс между своими, зачастую несовместимыми, целями?

9. Инновации: Как уникальные для данного проекта технологии и методы влияют на возможный результат?

10. Масштаб: Как повлияет на осуществление проекта увеличение масштаба работ, если раньше у разработчика не было соответствующего опыта?

Top -10 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. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. “Разрыв” в квалификации специалистов разных областей знаний.

Барри Боэмом [Boehm, 1988]

Описание ролей участников группы тестирования.

Тест-менеджер, менеджер проекта по тестированию - производит управленческий контроль.

Ответственность:

• Обеспечивает техническое направление

• Получает необходимые ресурсы

• Обеспечивает управленческую отчётность

Тест дизайнер - обеспечивает разработку тестовых случаев

Ответственность:

•Разрабатывает план тестирования

•Разрабатывает модель тестирования

•Оценивает эффективность тестирования

Тестировщик, Инженер по тестированию -
выполняет тесты
Ответственность:
•Выполняет тест
•Фиксирует результаты
•Восстанавливает тесты и систему после сбоев
•Документирует запросы на изменение

7 принципов тестирования

- Тестирование показывает наличие дефектов, но не доказывает их полное отсутствие
- Полное (exhaustive) тестирование невозможно. Перебрать все комбинации ко входам можно только в тривиальных случаях. Надо фокусироваться на приоритетах
- Начинать тестирование надо на как можно ранних этапах цикла разработки
- Ошибки физически образуют кластеры (наиболее ошибочны конкретные модули)
- Парадокс пестицидов - ошибки подстраиваются под существующие тесты, поэтому их надо периодически обновлять
- Тестирование зависит от контекста системы
- Отсутствие дефектов не значит выполнение требований и ожиданий пользователя

Главное качество тестировщика

Тестировщик должен быть любопытным, им должно быть интересно, как это работает?

А что случится если…..?

Приложение.

Особенности тестирования приложений, написанных на объектно-ориентированных языках.

Процедурное программирование

Предполагает написание исходного кода в императивном (повелительном) стиле, предписывающем определенную последовательность выполнения команд. Процедура представляет собой законченную последовательность действий или операций, направленных на решение отдельной задачи.

Сложные программные проекты имеют модульно-иерархическое построение, и тестирование модулей является начальным шагом процесса тестирования ПО.

Каждый модуль имеет несколько точек входа (при строгом написании кода - одну) и несколько точек выхода (при строгом написании кода - одну). Сложные программные проекты имеют модульно-иерархическое построение, и тестирование модулей является начальным шагом процесса тестирования ПО.

Объектно-ориентированное программирование

Объектно-ориентированное программное обеспечение является событийно управляемым. Передача управления внутри программы осуществляется не только путем явного указания последовательности обращений одних функций программы к другим, но и путем генерации сообщений различным объектам, разбора сообщений соответствующим обработчиком и передача их объектам, для которых данные сообщения предназначены.

Фундаментальные понятия ООП:

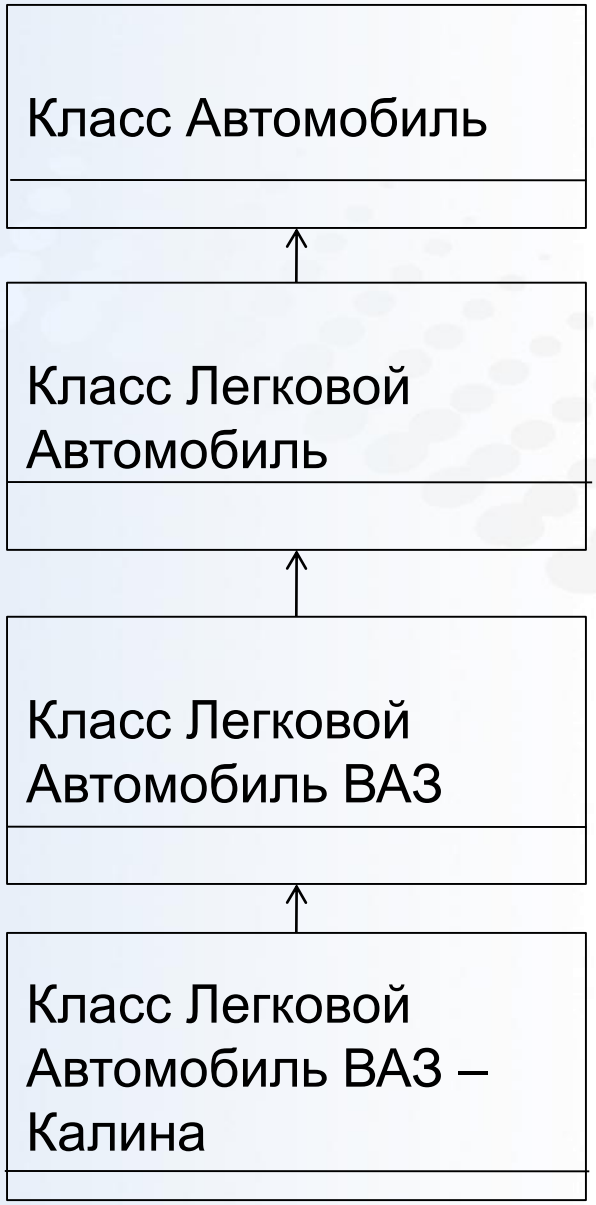
Класс – некая абстрактная совокупность объектов, которые имеют общий набор свойств и обладают одинаковым поведением

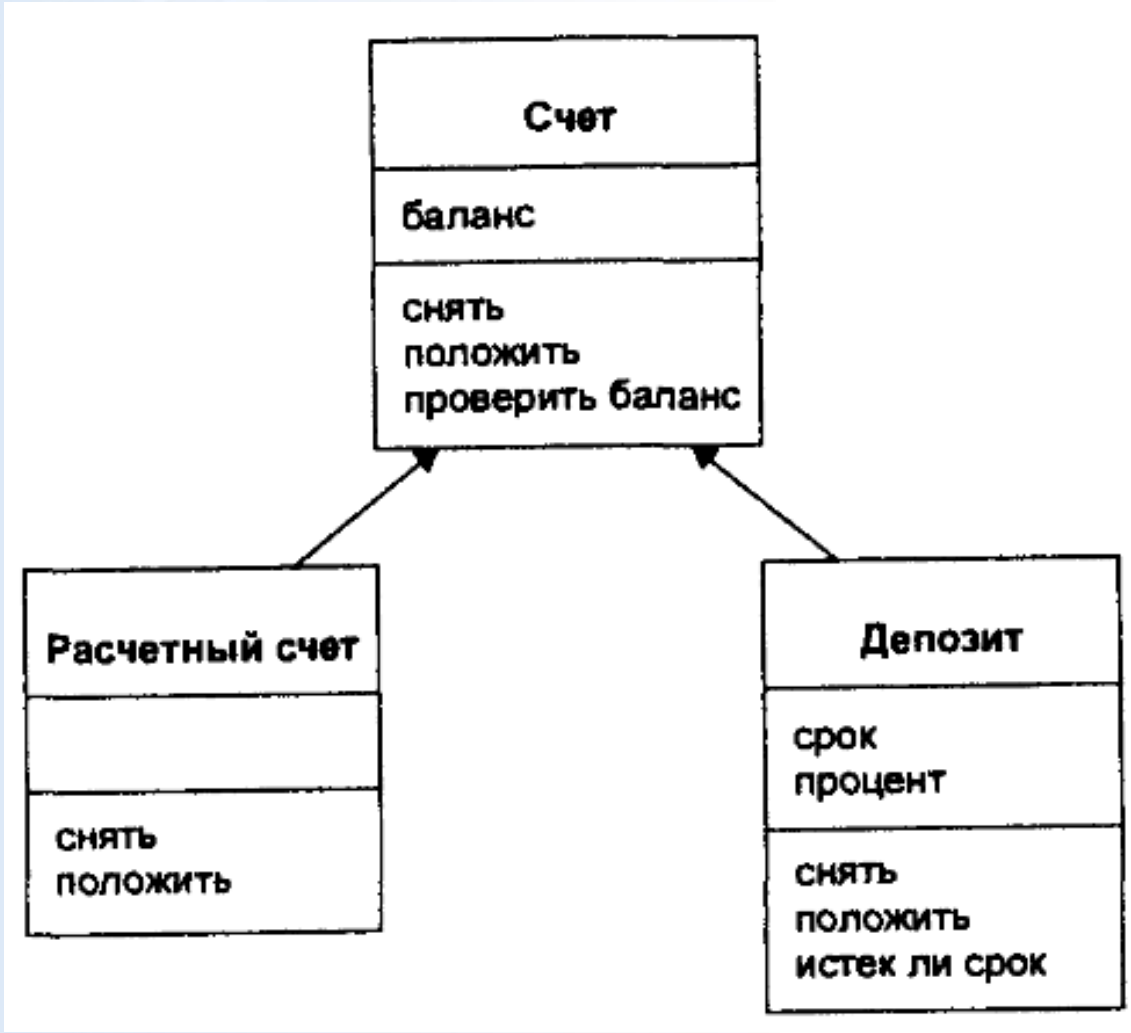
Объект – экземпляр (представитель) соответствующего класса.

Объекты которые не имеют полностью одинаковых свойств или не обладают одинаковым поведением не могут быть отнесены к одному классу.

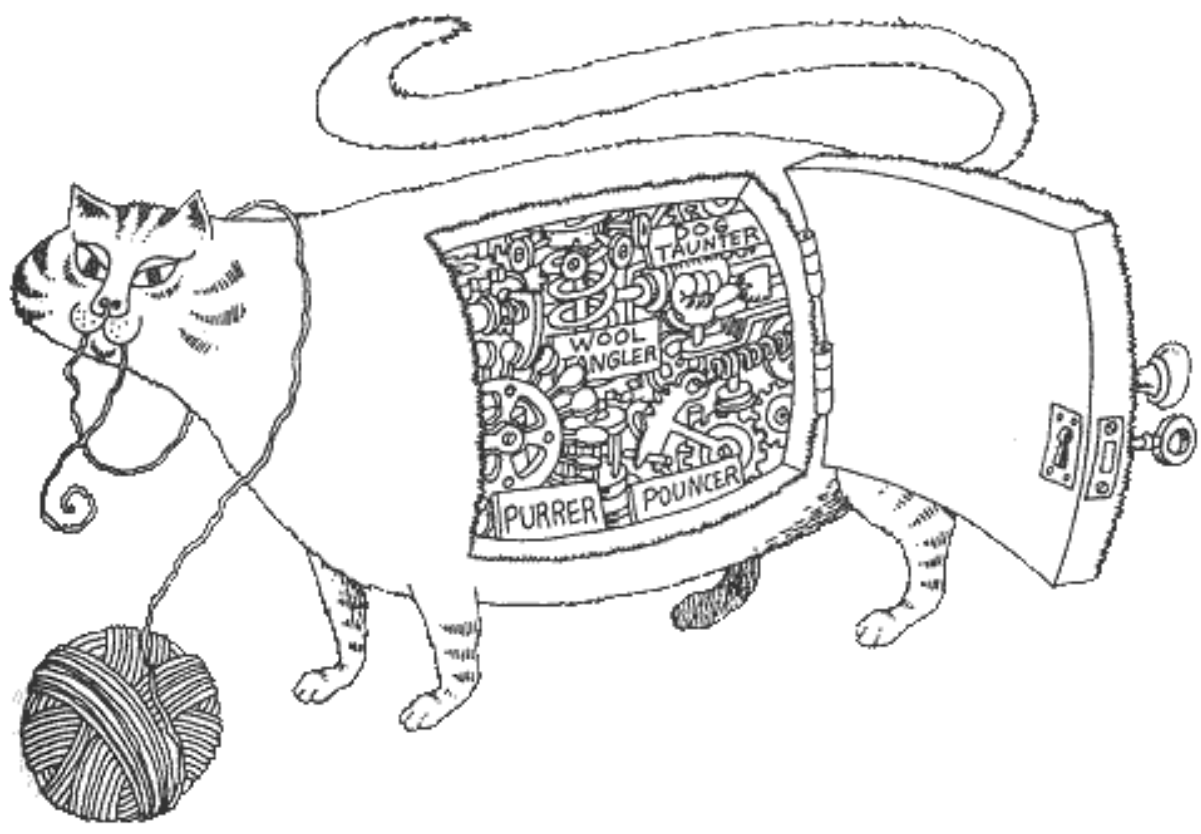
Основные принципы ООП

Наследование – принцип в соответствии с которым знание о более общей категории разрешается применять к более частной категории. При этом, если некоторый более общий или родительский класс обладает фиксированным набором свойств и поведением, то и потомок должен содержать этот же набор свойств и поведение, а так же дополнительные, которые будут характеризовать его уникальность.





Инкапсуляция – этот термин характеризует сокрытие отдельных деталей внутреннего устройства классов от внешних по отношению к нему объектов или пользователей



Полиморфизм означает, что действия, выполняемые одноименными методами, могут отличаться в зависимости от того к какому классу относится тот или иной метод.

Пример: три объекта

1. Автомобиль
2. Электрический свет в комнате
3. Компьютер

Операция «выключить».

1 – прекращение подачи топлива и его остановка

2 – щелчок выключателя, комната погрузится в темноту

3 – запускаются процедуры закрытия программ

Модульное тестирование.

Обычно за тестируемый модуль принимается класс, если система разрабатывается на объектно-ориентированном языке.

При тестировании объектно-ориентированных систем существует ряд особенностей, прежде всего вызванных инкапсуляцией данных и методов в классах.

Процесс тестирования классов как модулей иногда называют компонентным тестированием. В ходе такого тестирования проверяется взаимодействие методов внутри класса и правильность доступа методов к внутренним данным класса.

Специфические дефекты объектно-ориентированного программного обеспечения:

- дефектов инкапсуляции, в результате которых, например, сокрытые данные класса оказывается доступными при помощи соответствующих публичных методов;
- дефектов наследования, при наличии которых схема наследования блокирует важные данные или методы от классов-потомков;
- дефектов полиморфизма, при которых полиморфное поведение класса оказывается распространенным не на все возможные классы;
- дефектов инстанцирования, при которых во вновь создаваемых объектах класса не устанавливаются корректные значения по умолчанию параметров и внутренних данных класса.