

Тестирование ПО

Цели и задачи тестирования

Кулаков Кирилл Александрович

Информация о курсе

- Лекции 2 часа в неделю
- Контроль успеваемости
 - лабораторные работы
 - зачет
- Помощь
 - Сайт курса
(<http://cs.petrSU.ru/~kulakov/courses/testing>)
 - График консультаций кафедры ИМО (215 ауд.)
 - Электронная почта (kulakov@cs.petrSU.ru)

Литература

- В.П. Котляров Основы тестирования программного обеспечения <http://www.intuit.ru/department/se/testing/>
- С. Канер, Д.Фолк Тестирование ПО
- Э. Дастин, Д. Рэшка, Д. Пол "Автоматизированное тестирование программного обеспечения"
- Р. Калбертсон, К. Браун, Г. Кобб "Быстрое тестирование"
- Д. Макгрегор, Д. Сайкс "Тестирование объектно-ориентированного программного обеспечения"
- Л. Тамре "Введение в тестирование программного обеспечения"
- Р. Савин "Тестирование Дот Ком, или пособие по жесткому обращению с багами в интернет-стартапах"
- Э. Хант, Д. Томас "Программист-прагматик. Путь от подмастерья к мастеру"
- ...

Определения

- **Программное обеспечение (ПО)** - множество развивающихся во времени логических предписаний, с помощью которых некоторый коллектив людей управляет и использует многопроцессорную и распределенную систему вычислительных устройств.
 - Логические предписания – это не только сами программы, но и различная документация и т. д. (система отношений)
 - Современное ПО предназначено, как правило, для одновременной работы со многими пользователями, которые могут быть значительно удалены друг от друга в физическом пространстве (распределенность)
 - Задачи решаемые современным ПО, часто требуют различных вычислительных ресурсов (ресурсоемкость)
 - ПО развивается во времени – исправляются ошибки, добавляются новые функции, выпускаются новые версии, меняется его аппаратная база (обновляемость)

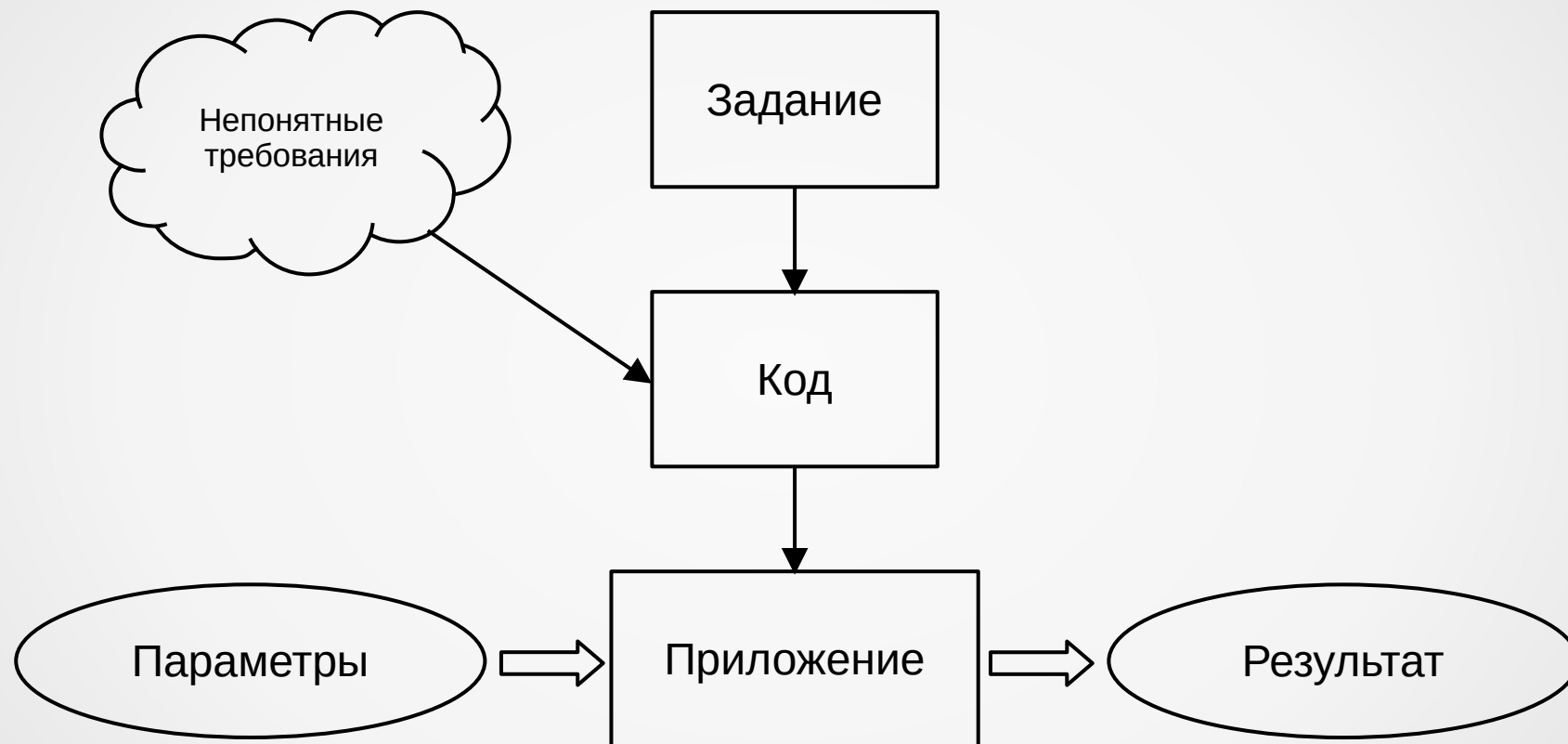
Виды программ

- Автономное (standalone) консольное ПО
- Автономное графическое ПО
- Серверное ПО
 - Инфраструктурное ПО
 - Клиентское ПО
- Сервисы/демоны/процессы
- Мобильное ПО
- Встраиваемое (embedded) ПО

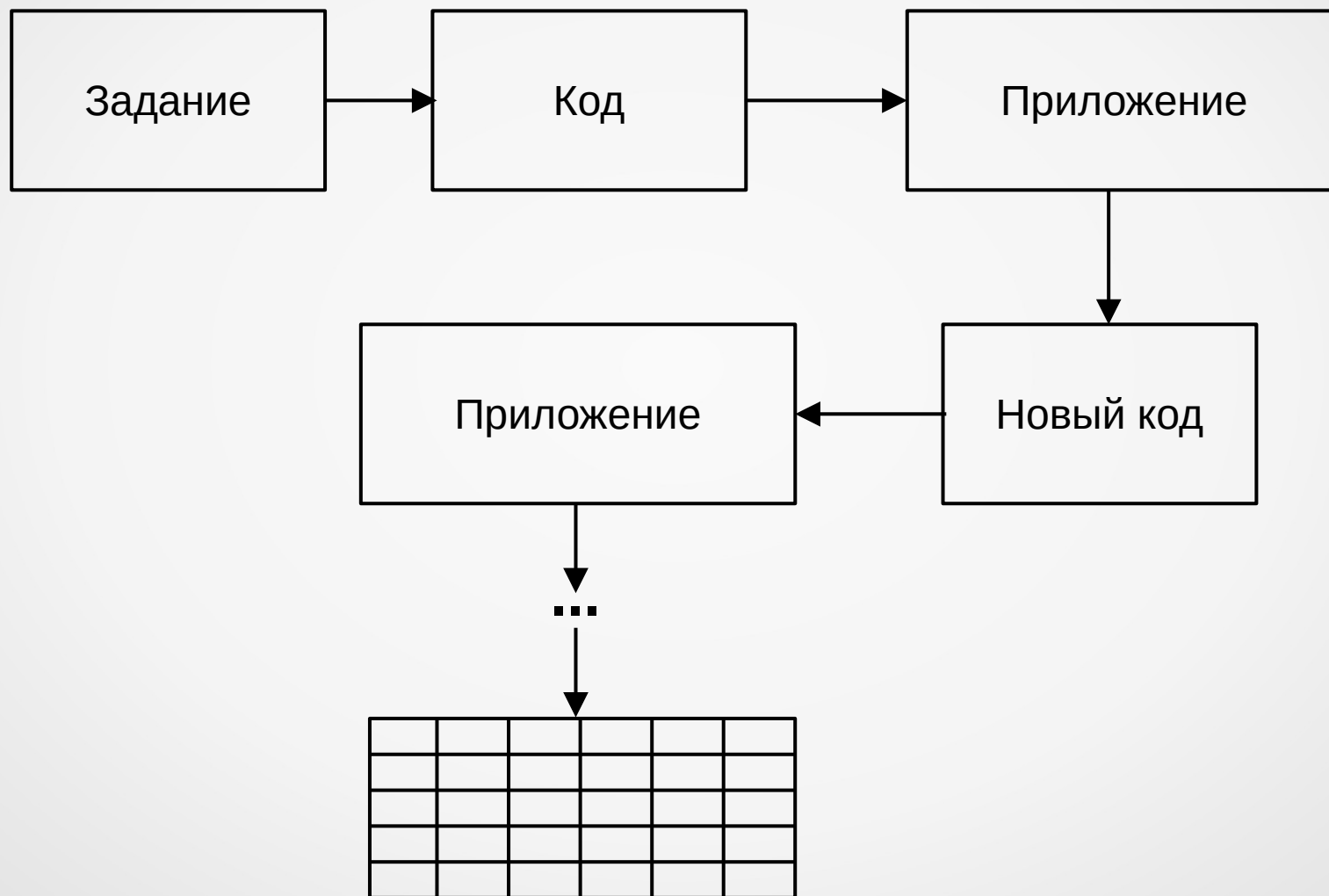
Свойства ПО

- **Сложность** программных объектов, которая существенно зависит от их размеров.
- **Согласованность** – ПО основывается не на объективных посылках, а должно быть согласовано с большим количеством интерфейсов, с которыми впоследствии оно должно взаимодействовать.
- **Изменяемость** – ПО легко изменить и, как следствие, требования к нему постоянно меняются в процессе разработки.
- **Незримость** – ПО невозможно увидеть, оно виртуально. Поэтому, например, трудно воспользоваться технологиями, основанными на предварительном создании чертежей, успешно используемыми в других промышленных областях

Процесс создания простой программы



Более сложный вариант



Программа с развитием



Обеспечение взаимодействия

- Сложные программы создаются командами разработчиков
- Необходима организация взаимодействия людей
 - На уровне задач (кто что будет делать)
 - На уровне работ (кто как делает)
 - На уровне результатов (как оно должно быть)
- Необходима организация взаимодействия программ
 - Объекты данных
 - Процессы
 - Сценарии

Обеспечение взаимодействия

- **Законодательство** (международное, РФ)
 - Пример: ФЗ №63-ФЗ от 06.04.2011 (ред. От 30.12.2015) «Об электронной подписи»
- **Стандарты** (международные, отраслевые)
 - ЕСПД (ГОСТ 19) Единая система программной документации
 - ГОСТ 34. Автоматизированные системы
 - Стандарты IEEE, например, 829-1998 «Стандарт для тестовой документации тестирования программного обеспечения»

Обеспечение взаимодействия

- **Руководства (Guidelines):**
 - Рекомендации по кодированию
 - Рекомендации по интерфейсу пользователя
 - Рекомендации по проектированию
 - Рекомендации по тестированию
 - ...
- Ожидания пользователя
 - «Если кнопочка нарисована она должна нажиматься.»

Обеспечение взаимодействия

- Взаимодействие между программами
 - **Программные интерфейсы (API):** соглашение между разработчиками о подключении программ друг к другу.
 - Пример: математическая библиотека (`#include <math.h>`, ...)
 - **Протоколы передачи данных:** описание процессов передачи и структур данных.
 - Пример: HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SMTP), Remote Desktop Protocol (RDP), Remote Procedure Call (RPC)

Использование готовых решений

- Современная разработка ПО не начинается с нуля
- Если взять два соседних проекта, то у них будет что-то общее
 - Пример: чтение файла специфичной структуры
- Объединение повторяющихся кусков кода в **функции**
- Объединение функций в **библиотеки** (модули)
- Использование «решений» прошлых разработок
 - **Шаблоны** проектных решений
 - **Шаблоны** кода

Использование готовых решений

- Идеальный результат: берем блоки готовых решений, замазываем их клеем из кода и получаем построенную программу
- **Программные каркасы** (фреймворки): скелет будущего приложения куда надо вставить кирпичики кода
 - Уже готовые решения по стандартным операциям
 - Соответствие стандартам, рекомендациям и т. п.
 - Наличие документации

Использование готовых решений

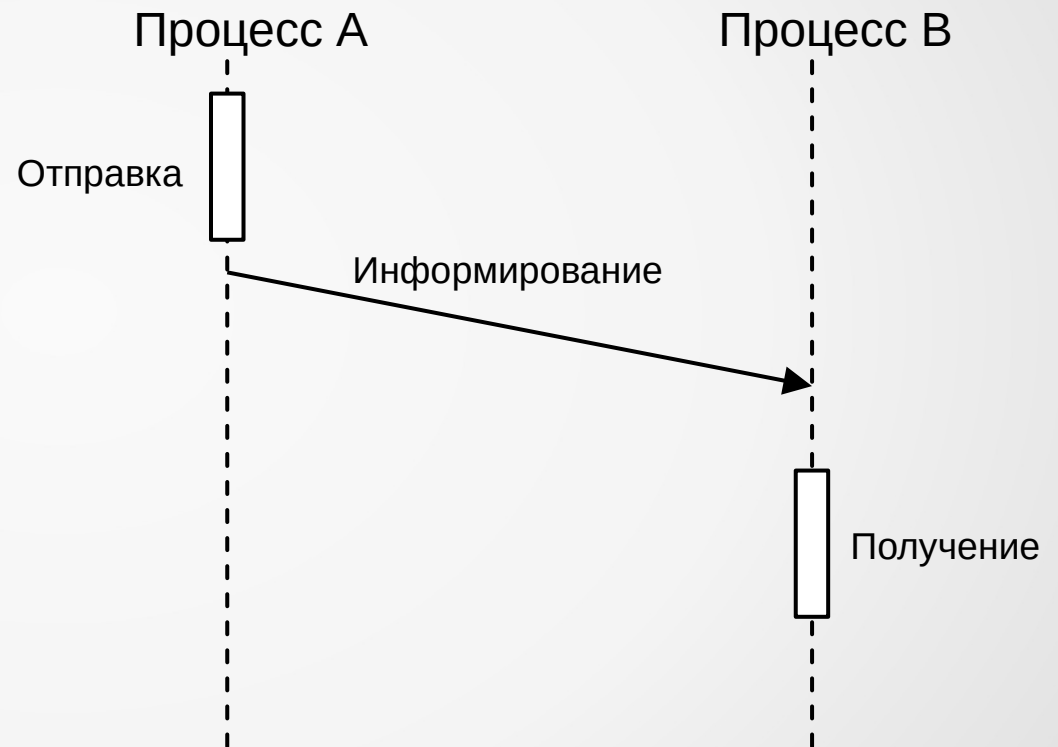
- Программные каркасы позволяют:
 - Минимизировать объем работ
 - Получить работающее приложение «за 5 минут»
 - Повторно использовать свои наработки
 - Расширять функционал с помощью дополнительных библиотек
 - Соответствовать стандартам и рекомендациям
- **Программные каркасы не решат за вас задачу, они помогут быстрее сделать решение!**

Процесс во времени

- Пример часто встречающейся проблемы: работа приложения во времени
 - Выполнение длительных алгоритмов
 - Прерывания во время выполнения
- Результат: нелинейная работа кода
- **Синхронное программирование** — последовательное выполнение операций с ожиданием результата (блокировка)
- **Асинхронное программирование** — результат работы операции доступен не сразу же, а через некоторое время

Процесс во времени

- Пример: передача массива данных между процессами
 - Процесс А отправляет массив
 - Процесс В получает массив
- Как отправить массив без больших затрат ресурсов?
- Как узнать что массив отправлен?
- Как получить массив?



Процесс во времени

- Код процесса A:

```
sendData(arrayData).onSuccess(  
    function () {notifyProcessB();}  
)
```

- Код процесса B

```
waitData().onReady(  
    function (array arrayData) {m_data = arrayData;}  
)
```

Процесс во времени

- Разработчик/программист должен всегда представлять в какой момент времени какой кусок кода будет выполняться и какие данные ему будут доступны
- Иначе он столкнется с различными проблемами, например:
 - Проблема узкого места bottleneck
 - Проблемы взаимных блокировок
- Решение: продумать (спроектировать) приложение до его реализации

Программная инженерия

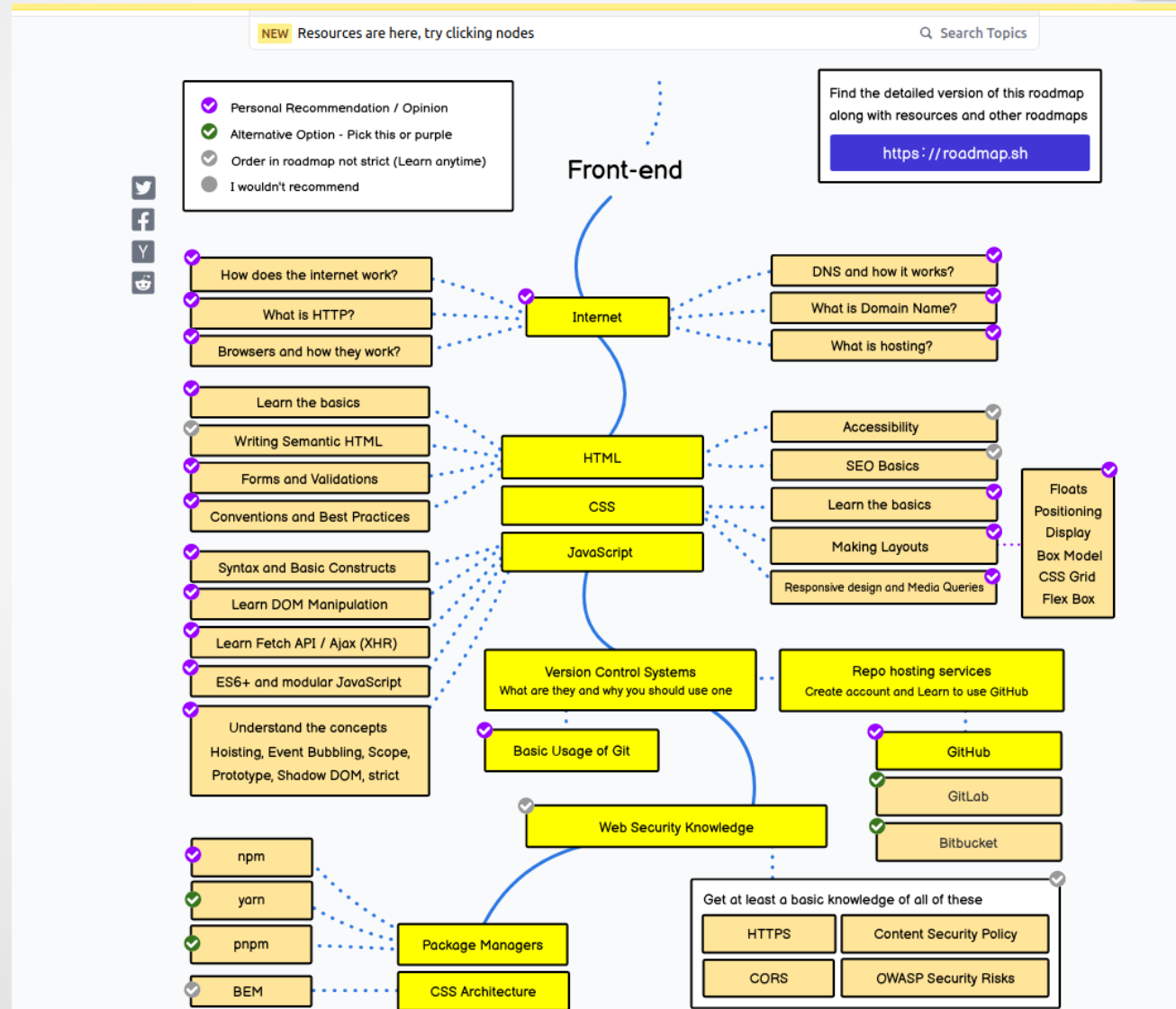
- **Программная инженерия** — раздел компьютерных наук (англ. computer sciences), изучающий методы и средства построения компьютерных программ как продукта теоретической и инженерной деятельности разработчиков или их коллективов.
- **Основные критерии ПИ:** продуктивность, индустрия и качество.

Программная инженерия

- Программная инженерия основывается на математических дисциплинах:
 - теория алгоритмов нормальные алгоритмы, вычислимые функции, машина Тьюринга, граф-схемы, модели алгоритмов;
 - математическая логика формальный вывод утверждений;
 - теория управления принципы, методы и общие законы планирования и управления в сложных системах;
 - теория доказательств математическая теория вывода по аксиомам и утверждениям, теория верификации программ;
 - теория множеств формальное представление совокупностей объектов из предметной области.

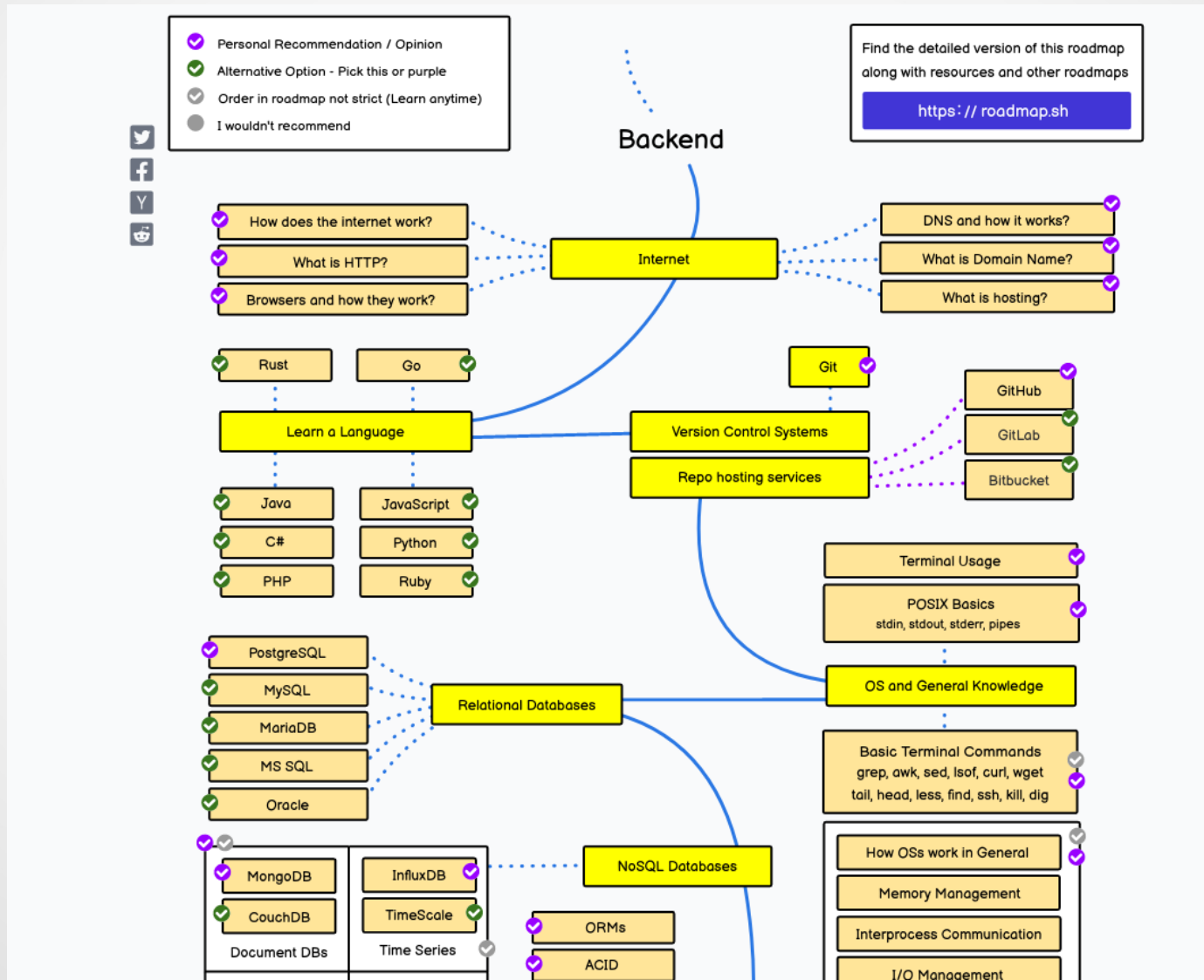
Дерево знаний

- <https://roadmap.sh/frontend>



Дерево знаний

- <https://roadmap.sh/backend>



Проблемы написания сложных программ

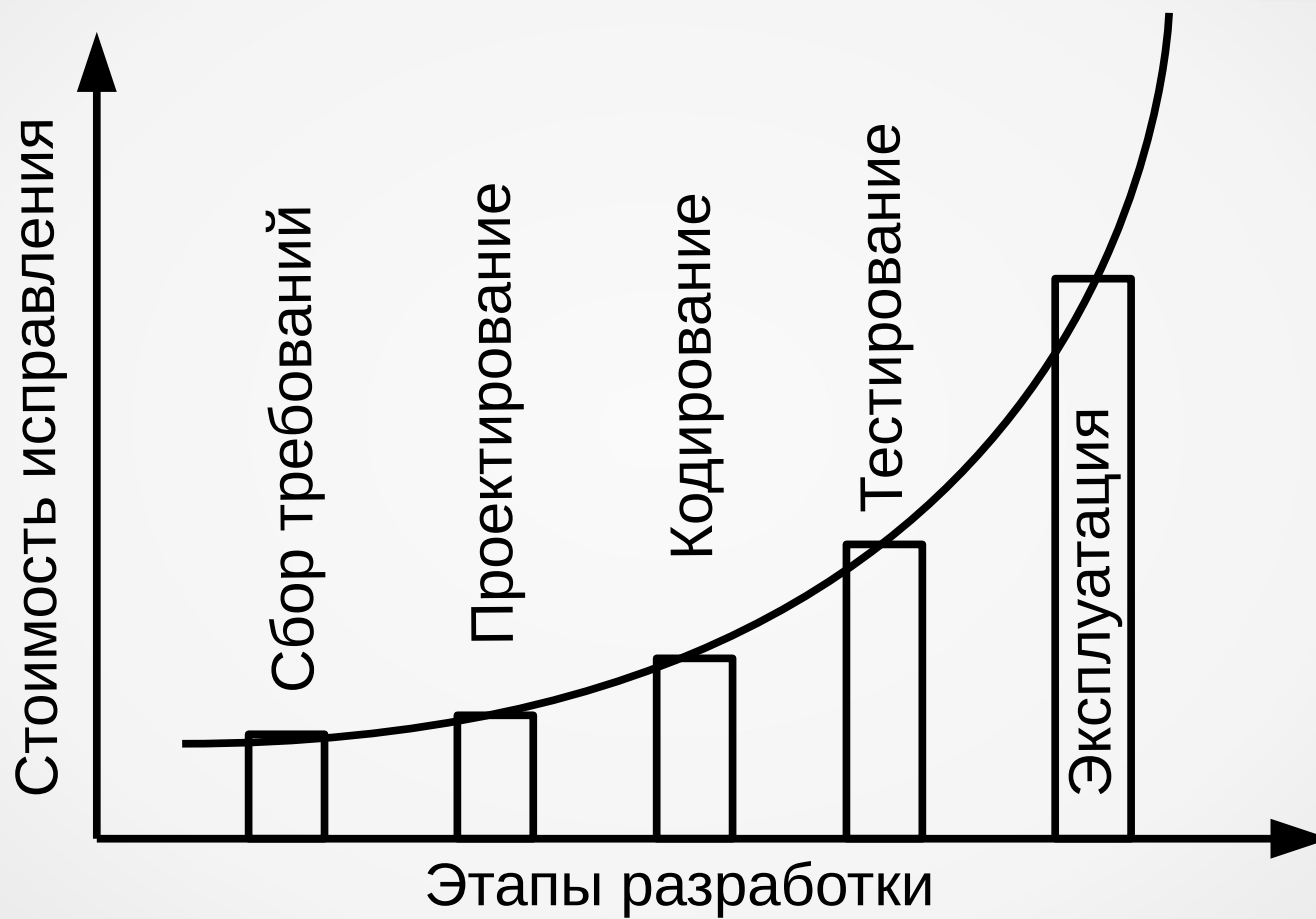
- Техническое задание
 - Слишком сложное
 - Очень большое
- Сроки реализации
- Потребление ресурсов (время, память, сеть, экран, ...)
- Объем работ
 - 1 программист в среднем пишет 500 строк хорошего кода в день
- Взаимодействие внутри команды

- Необходимость перехода от программирования как искусства к программированию как индустрии в связи с усложнением программного обеспечения.

Проблемы написания сложных программ

- Цели программ бывают разные
 - На практике 1 программа = множество целей
- Задача разработчика ПО — приблизиться к цели
 - Реализация качественного ПО
- Как проверить результат работ?
 - Выполнение тестирования и аттестации
- Необходимо предусмотреть возможность проверок кода

Стоимость исправления ошибки



Плохой программист Джон

Плохой программист Джон сделал ошибку в коде, из-за которой каждый пользователь программы был вынужден потратить в среднем 15 минут времени на поиск обхода возникшей проблемы. Пользователей было 10 миллионов. Всего впустую потрачено 150 миллионов минут = 2.5 миллиона часов. Если человек спит 8 часов в сутки, то на сознательную деятельность у него остается 16 часов. То есть Джон уничтожил 156250 человеко-дней \approx 427.8 человеко-лет. Средний мужчина живет 64 года, значит Джон убил примерно 6 целых 68 сотых человека.

Как тебе спится, Джон — серийный программист?

Тестирование — это:

- 1980 - Процесс выполнения программы с намерением найти ошибки
- 1987 - Процесс наблюдения за выполнением программы в специальных условиях и вынесения на этой основе оценки каких-либо ее аспектов
- 1990 - Интеллектуальная дисциплина, имеющая целью получение надежного программного обеспечения без излишних усилий на его проверку
- 1999 - Техническое исследование программы для получения информации о ее качестве с точки зрения определенного круга заинтересованных лиц
- 2004 - Проверка соответствия между реальным поведением программы и ее ожидаемым поведением на конечном наборе тестов, выбранном определенным образом

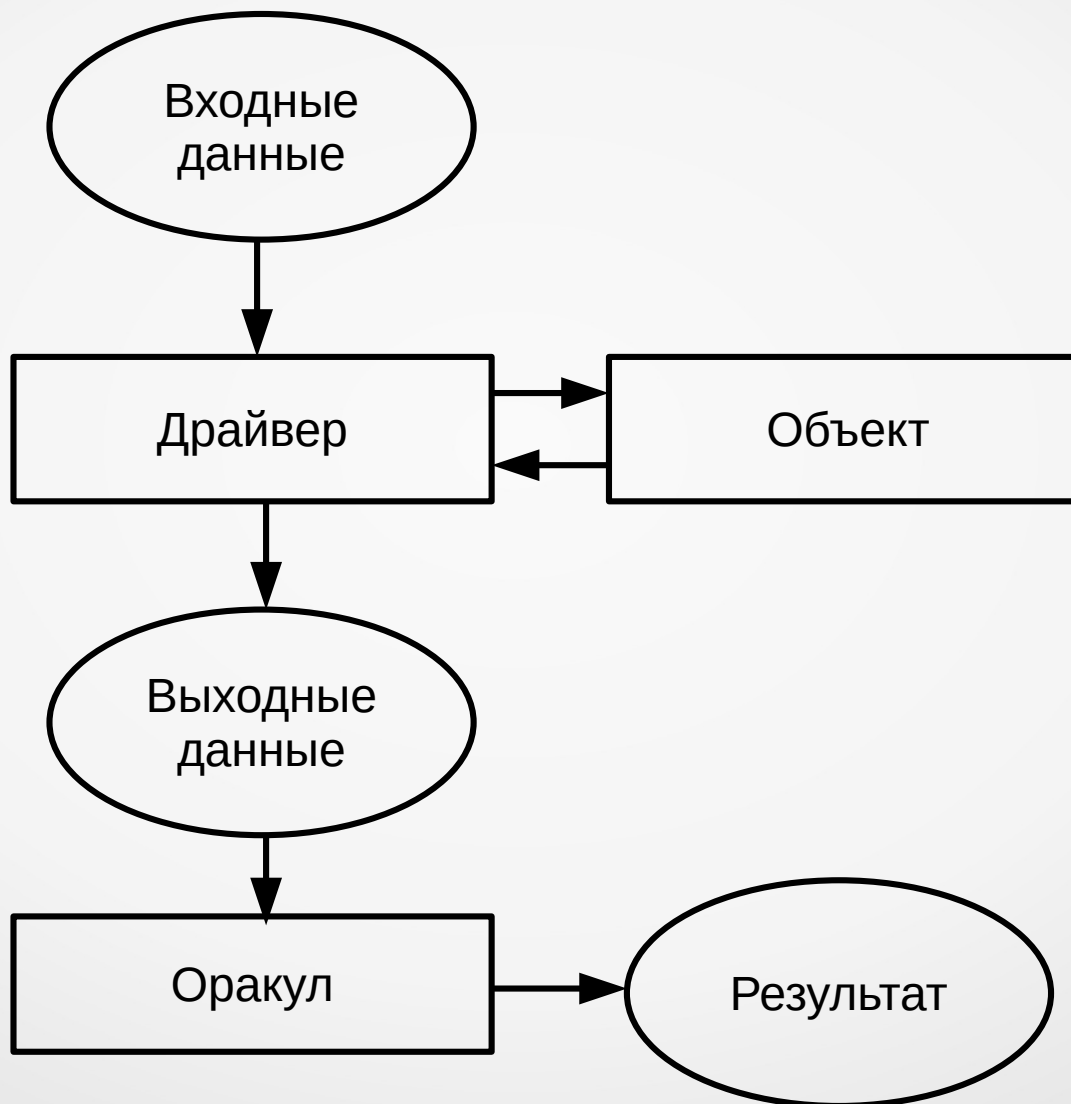
Терминология

- **Тестирование** обеспечивает выявление (констатацию наличия) фактов расхождений с требованиями (ошибок)
 - **Статическое тестирование** выявляет формальными методами анализа без выполнения тестируемой программы неверные конструкции или неверные отношения объектов программы (ошибки формального задания) с помощью специальных инструментов контроля кода
 - **Динамическое тестирование** (собственно тестирование) осуществляет выявление ошибок только на выполняющейся программе с помощью специальных инструментов автоматизации тестирования
- **Отладка** (debug, debugging) – процесс поиска, локализации и исправления ошибок в программе

Отладка

- "Выполнение программы в уме" (deskchecking).
- Вставка операторов протоколирования (печати) промежуточных результатов (logging).
- Пошаговое выполнение программы (single-step running).
- Выполнение с заказанными остановками (breakpoints), анализом трасс (traces) или состояний памяти - дампов (dump).
- Реверсивное (обратное) выполнение (reversible execution).

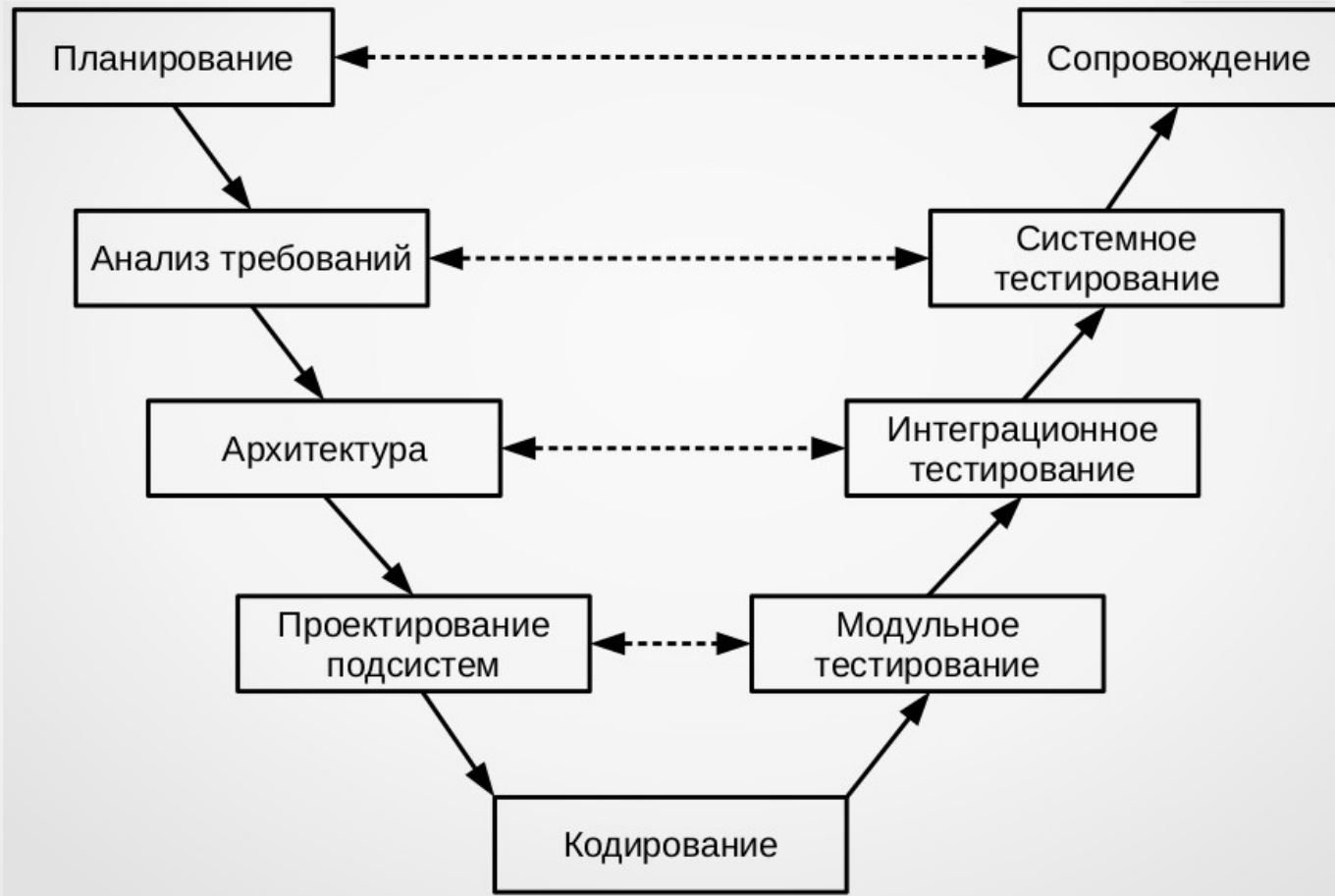
Схема тестирования



Когда тестировать?

- Тестирование нового кода
 - наличие заявленной функциональности
 - отсутствие ошибок
 - возможность интеграции в проект
- Тестирование изменений
 - изменение может быть неправильным
 - изменение может отобразить скрытые ошибки
 - изменение может привести к потере работоспособности
- Тестирование демонстрацией
 - эффект демо: "Вчера все работало, а тут ..."
- Тестирование использованием
 - реальное применение разработанного ПО

Когда тестировать?



Как тестировать?

- Запуск всей программной системы
 - дорогостоящая операция
 - неизвестно где ошибка
- Запуск компоненты системы (приложения)
 - надо много окружения
 - неизвестно где ошибка
- Запуск модуля приложения
 - сам модуль не запустится — нужен драйвер
 - модуль использует внутренние форматы данных

Чем тестировать?

- Каждый тест — отдельная программа
 - как это все запускать?
- Скрипты запуска тестовой программы с параметрами
 - что будет если надо добавить тест?
 - А если надо временно отключить группу тестов
- Система контроля и автоматического запуска тестов
 - много магии
 - Требуется «управление» не только разработкой, но и процессом тестирования

Проблемы тестирования

- Тестирование программы на всех входных значениях невозможно.
- Невозможно тестирование и на всех путях.
- Надо отбирать конечный набор тестов, позволяющий проверить программу на основе наших интуитивных представлений
- В теории алгоритмов доказано, что не существует общего метода для решения этого вопроса, а также вопроса, достигнет ли программа на данном тесте заранее фиксированного оператора.
- **Основная проблема тестирования** — определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.

Этапы тестирования

- 1) Создание тестового набора (test suite) путем ручной разработки или автоматической генерации для конкретной среды тестирования (testing environment)
- 2) Прогон программы на тестах, управляемый тестовым монитором (test monitor, test driver) с получением протокола результатов тестирования (test log).
- 3) Оценка результатов выполнения программы на наборе тестов с целью принятия решения о продолжении или остановке тестирования.

Характеристики хорошего теста

- Цель тестирования выявление ошибок
- Ошибка — отклонение от эталона
- Варианты эталонов:
 - неформальное представление того, «как ПО должно работать»;
 - формальная техническая спецификация;
 - набор тестовых примеров;
 - корректные результаты работы программы;
 - другая (априори корректная) реализация той же исходной спецификации.

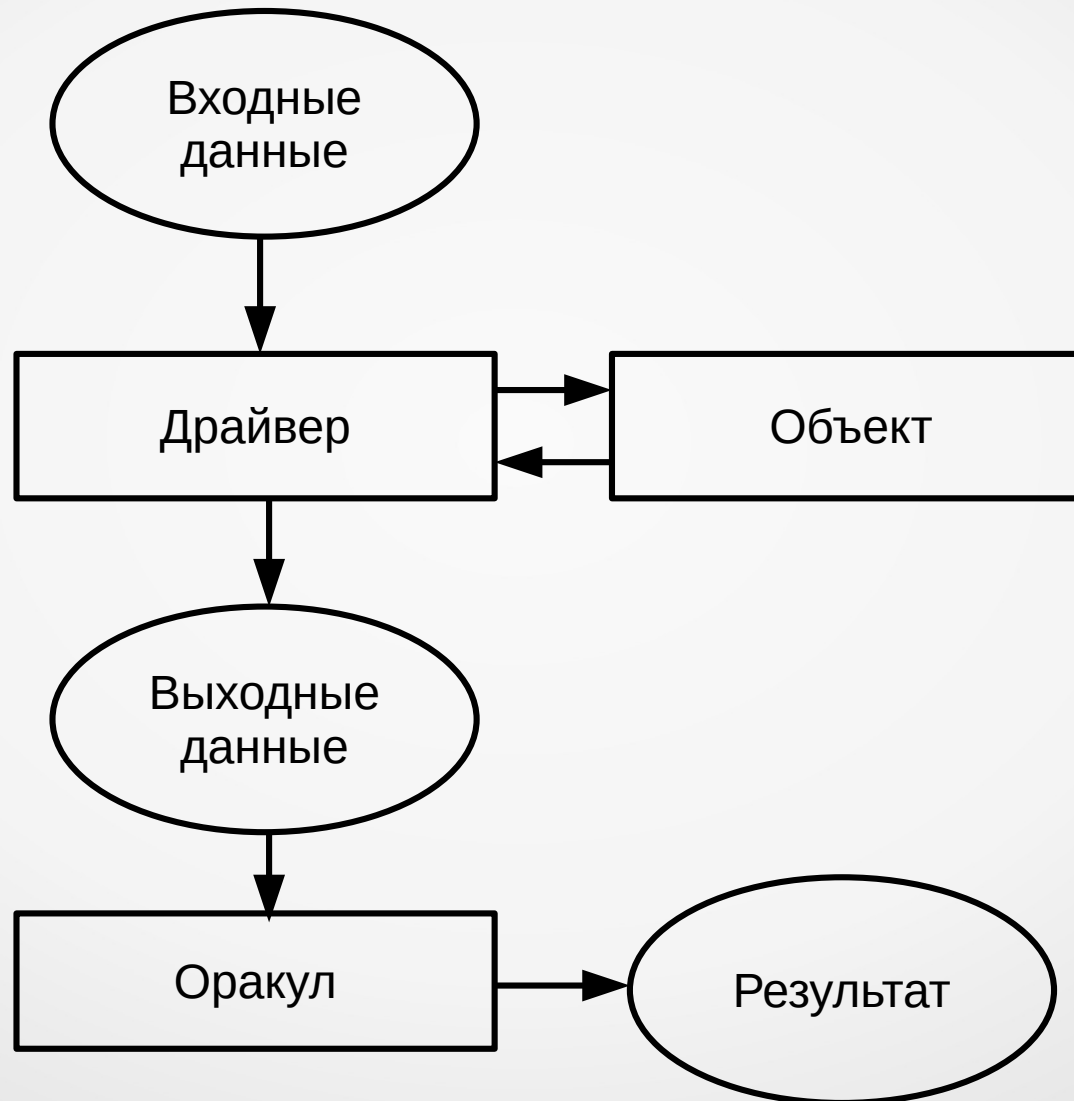
Характеристики хорошего теста

- достижение (Reachability) — тест должен выполнить место в исходном коде, где присутствует программная ошибка;
- повреждение (Corruption) — при выполнении ошибки состояние программы должно испортиться с появлением сбоя;
- распространение (Propagation) — сбой должен распространиться дальше и вызвать неудачу в работе программы.

Модульное тестирование

- Структурное тестирование
- Поиск дефектов
 - алгоритмические ошибки
 - ошибки кодирования алгоритмов
 - выполнение условных и циклических операторов
 - использование переменных и ресурсов
- Использование заглушек, эмуляторов и других вспомогательных инструментов, заменяющих полностью или частично реальные компоненты ПО

Модульное тестирование



Интеграционное тестирование

- Смешанная модель разработки
- Поиск ошибок
 - трактовка данных
 - реализация интерфейса взаимодействия
 - совместимость
- Схема интеграции
 - восходящая (от малого к большому)
 - нисходящая (подключение к основному модулю)
 - смешанная (сборка блоков и подключение к основному модулю)

Интеграционное тестирование

	Восходящее	Нисходящее
Преимущества	<ul style="list-style-type: none">• Возможность ранней проверки корректности низкоуровневого поведения• Не требуется написание заглушек• Просто определить требования ко входам/выходам модулей	<ul style="list-style-type: none">• Возможность ранней проверки корректности высокоуровневого поведения• Модули могут добавляться по одному, независимо друг от друга• Не требуется разработка множества драйверов• Можно разрабатывать систему как в глубину, так и в ширину
Недостатки	<ul style="list-style-type: none">• Отложенная проверка высокоуровневого поведения• Требуется разработка драйверов• При замене драйвера на модуль высокого уровня может произойти "мини-Большой Взрыв" числа найденных ошибок	<ul style="list-style-type: none">• Отложенная проверка низкоуровневого поведения• Требуется разработка "заглушек"• Крайне сложно корректно сформулировать требования ко входам/выходам частичной системы

Системное тестирование

- Функциональное тестирование
- Проверка на соответствие стандартам
- Оценка характеристик качества ПО
 - устойчивость
 - надежность
 - безопасность
 - производительность
- Приемочное тестирование
- Демонстрация
- Тестирование пользователями

Переходы между состояниями

- Переход м/у состояниями в следствие действий пользователя или внешнего окружения
- Переходов может быть очень много!
 - Протестируйте все наиболее вероятные последовательности действий пользователей.
 - Если можно предположить, что действия пользователя в одном режиме могут воздействовать на представление данных или набор предоставляемых программой возможностей в другом режиме, протестируйте эту зависимость.
 - Кроме проведения самых необходимых тестов — из тех, что описаны выше, — стоит поработать с программой в произвольном режиме, случайным образом выбирая путь ее выполнения.
- Построение схем меню и форм

Регрессионное тестирование

- Исправление ошибки зачастую порождает новые ошибки
 - Проверка исправления ошибки
 - Поиск связанных ошибок
 - Проверка остальной части программы
- Использование библиотеки регрессионных тестов
 - Удаление тестов эквивалентных другим тестам библиотеки.
 - Уменьшение количества тестов, объектом которых является уже исправленная ошибка
 - Комбинации тестов
 - Автоматизация тестирования
 - Периодическое выполнение

Анализ чувствительности

- Поиск тестов, вызывающих наибольшие возмущения
 - Общее представление о поведении функции, на основе ее значений для ряда параметров, располагающихся вдоль всей области определения
 - Поиск участков области определения, на которых небольшие изменения аргументов вызывают значительные скачки результирующих значений
- Оценка погрешностей получаемых значений (операции с плавающей запятой)
- Использование теории вероятности для тестирования мат.функций

Нагрузочные испытания

- Нагрузочное испытание — один из видов пограничного тестирования
- Проверка максимально допустимых ресурсов (размер файлов, количество соединений, устройств и т.п.)
- Условия гонок
 - Проверка работы в нагруженном и ненагруженном режиме
 - Воздействия на приложение во время работы
 - Проверка на быстрых и медленных машинах

Пользовательское тестирование

- Пользовательское тестирование
- Альфа-тестирование
- Бета-тестирование
- Произвольное использование (bag bush)
- Тестирование экспертом
- Парное тестирование
- Использование внутри компании (Eat your own dogfood)
- Тестирование локализации

Дерево знаний

- <https://roadmap.sh/qa>

