# Технологии разработки программного обеспечения

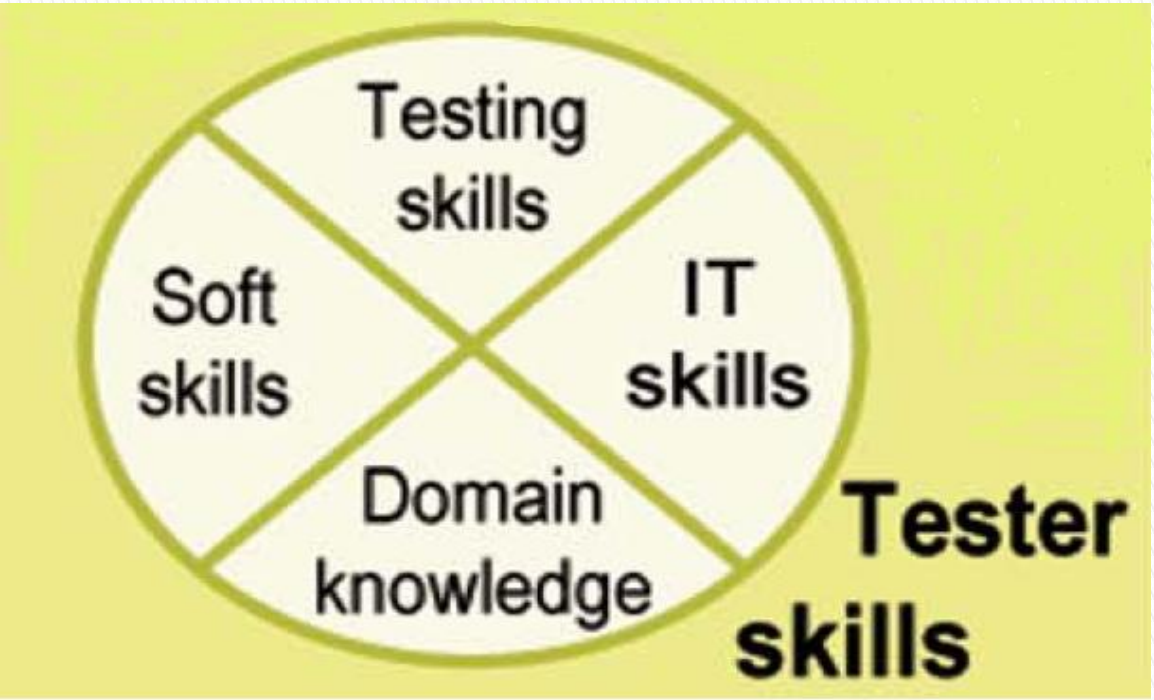
по основной образовательной программе
подготовки бакалавров по направлению
09.03.02 «Информационные системы и технологии»
(квалификация – «бакалавр»)

(3 курс, 6 семестр)
форма обучения: очная

Лекция 8

# Тема: Тестирование программного обеспечения

# Что должен знать тестировщик ПО




Тестирование программ можно использовать для того, чтобы показать наличие ошибок, и никогда — для того чтобы показать их отсутствие!

Эдсгер Дейкстра

# Литература


1. С. Канер, Д. Фолк, Е. Нгуен. Тестирование программного обеспечения. — К.: Диасофт, 2000. — 544 с.
2. Р. Калбертсон, К. Браун, Г. Кобб. Быстрое тестирование. — М: Вильямс, 2002.
3. С. Макконнелл. Совершенный код. — СПб: «Питер», 2005. — 896 с.
4. Г. Майерс. Искусство тестирования программ. — М.: «Финансы и статистика», 1982. — 176 с.
5. Л. Тамре. Введение в тестирование программного обеспечения — М.: «Вильямс», 2003. — 368 с.
6. Г. Майерс. Надежность программного обеспечения. — М.: «Мир», 1980. — 360 с.
7. Б. Бейзер. Тестирование черного ящика. — СПб: «Питер», 2005. — 318 с.
8. Э. Брауде. Технология разработки программного обеспечения. — СПб: «Питер», 2004. — 655 с.
9. С. Орлов. Технологии разработки программного обеспечения. — СПб: «Питер», 2003. — 480 с.

# Литература


11. И. Винниченко. Автоматизация процессов тестирования. — СПб: «Питер», 2005. — 203 с.
12. К. Бек. Экстремальное программирование. — СПб: «Питер», 2002.
13. К. Ауэр, Р. Миллер. Экстремальное программирование. — СПб: «Питер», 2003. — 368 с.
14. Д. Бентли. Жемчужины программирования. — СПб: «Питер», 2002. — 272 с.
15. С. Бобровский. Технологии Пентагона на службе российских программистов. — СПб: «Питер», 2003. — 222 с.
16. А. Якобсон, Г. Буч, Д. Рамбо. Унифицированный процесс разработки программного обеспечения. — СПб: «Питер», 2002. — 496 с.
17. Р. Мартин. Чистый код: создание, анализ и рефакторинг. — СПб: «Питер», 2010. — 464 с.

# Ресурсы

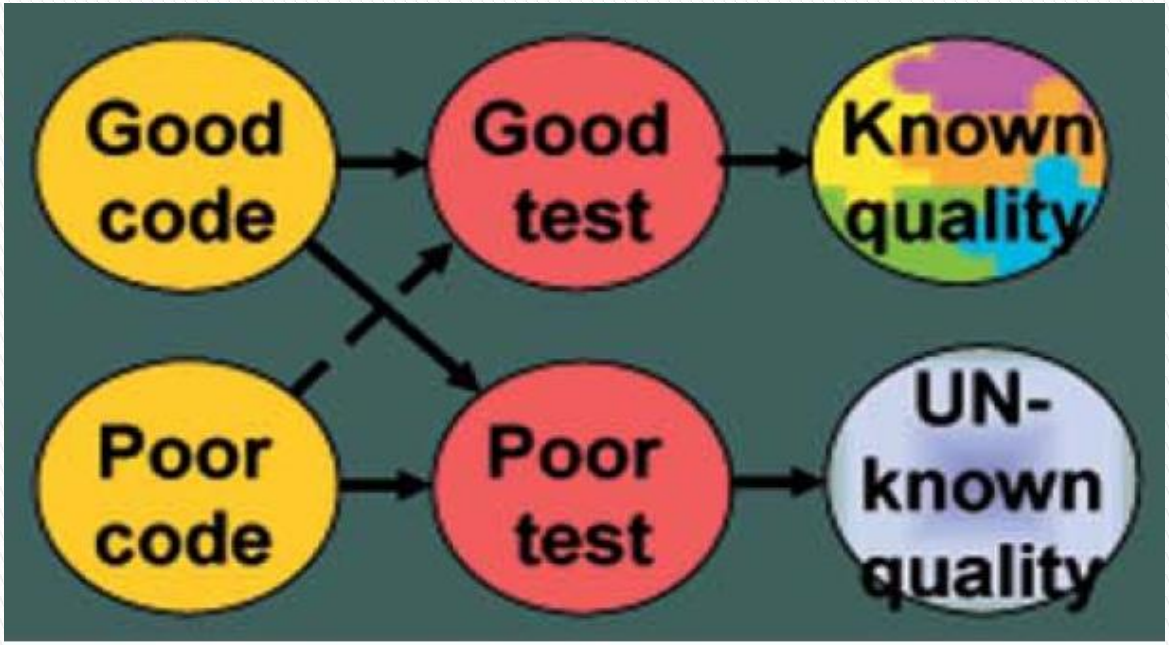
- sorlik.ru/swebok-ru/
  (SWEBOK - Software Engineering Body of Knowledge)
- software-testing.ru – библиотека, статьи, …
- wiki.agiledev.ru/doku.php – гибкая разработка и тестирование
- ru.wikipedia.org – Тестирование ПО, ISO 9126
- www.intuit.ru/catalog/se/testing - курсы лекций
- www.cmcons.com/map - карта сайта. Смотреть:
  - Термины тестирования ПО; Термины, относящиеся к качеству
  - Метрики кода; Тест Джоэла, …
- http://www.osp.ru/os/2008/07/5478839/ (Б.Майер, 7 принципов тестирования ПО)

# Объекты тестирования

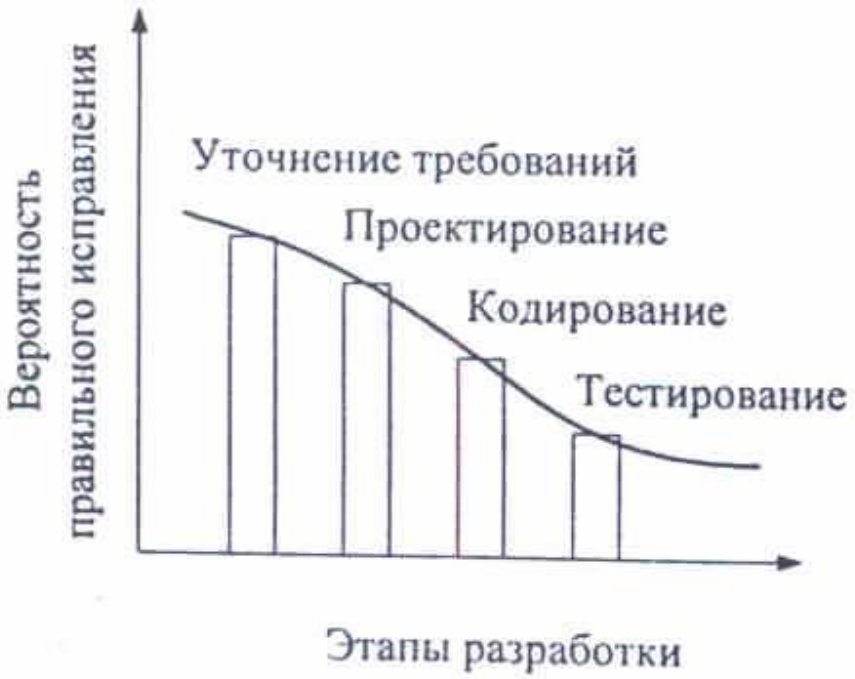
Тестировать можно все:

- работу программы
- качество ее кода и понятность комментариев
- быстродействие
- устойчивость под большой нагрузкой
- расход ресурсов (памяти, диска, потери этих ресурсов)
- взаимодействие с другими программами
- стабильность работы
- возможность работы на других платформах
- удобство интерфейса
- документацию к программе (смысловые и грамматические ошибки, понятность и полноту)
- работу через сеть, работу аппаратного обеспечения и т. п.

# Важность тестирования



# Стоимость ошибки





Зависимость вероятности правильного исправления (*а*) и его стоимости от этапа разработки (*б*)

4.06.1996 через 40 сек. после запуска ракеты-носителя Ariane 5 произошёл автоподрыв 50-метровой ракеты (оборудование стоило полмиллиарда долларов, не говоря об “упущенной выгоде”).

Причина - некорректный перенос из ПО Ariane 4 в ПО Ariane 5 спецификации программного модуля, выполнявшего преобразование из double в WORD.

Ракета Ariane 4 успешно запускалась более 100 раз.

# Основная терминология

- ***Тестирование*** – процесс выявления фактов расхождений с требованиями (ошибок).
- ***Отладка*** (debug, debugging) – процесс поиска, локализации и исправления ошибок в программе [IEEE Std.610-12.1990]

  Как правило, на ***фазе тестирования*** осуществляется и исправление идентифицированных ошибок, включающее:
  - локализацию ошибок
  - нахождение причин ошибок
  - корректировку программы.

Судить о правильности результатов выполнения программы можно только сравнивая спецификацию функции с результатами ее вычисления.

Основная *проблема тестирования* - определение достаточности множества тестов для истинности вывода о правильности реализации программы, а также нахождения множества тестов, обладающего этим свойством.

# Формирование тестовых наборов

- ***Структурный подход*** *базируется на том, что известка структура тестируемого* программного обеспечения, в том числе его алгоритмы («стеклянный ящик»). В этом случае тесты строят так, чтобы проверить правильность реализации заданной логики в коде программы.

- ***Функциональный подход*** *основывается на том, что структура программного обеспечения не* известна («черный ящик»). В этом случае тесты строят, опираясь на функциональные спецификации. Этот подход называют также подходом, управляемым данными, так как при его использовании тесты строят на базе различных способов декомпозиции множества данных.

# Определения тестирования по стандарту

- Процесс *выполнения* ПО системы или компонента при заданных условиях с анализом или записью результатов и оценкой некоторых свойств тестируемого объекта.

  *The process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.*
- Процесс *анализа* ПО с целью фиксации различий между существующим состоянием ПО и требуемым (что свидетельствует о проявлении ошибки) и оценки свойств тестируемого ПО.

  *The process of analyzing a software item to detect the differences between existing and required conditions (that is, bugs) and to evaluate features of software items [IEEE Std.610-12.1990].*
- Контролируемое выполнение программы на конечном множестве тестовых данных и анализ результатов этого выполнения для поиска ошибок [IEEE Std 829-1983].

# Ручной контроль программного обеспечения.

## Статическое и динамическое тестирование

- ***Статическое*** тестирование выявляет неверные конструкции или неверные отношения объектов программы (ошибки формального задания) формальными методами анализа без выполнения тестируемой программы:
  - С помощью специальных инструментов контроля кода
  - Обзоры (Reviews)
  - Инспекции (Inspections)
  - Сквозные просмотры (Walkthroughs)
  - Аудиты (Audits)
  - Тестирование требований, спецификаций, документации.
- ***Динамическое*** тестирование осуществляет выявление ошибок на выполняющейся программе.

Тестирование заканчивается, когда выполнилось или "прошло" (pass) успешно достаточное количество тестов в соответствии с выбранным критерием тестирования.

# Структурное тестирование

*Структурное тестирование называют также тестированием по «маршрутам», так как в этом* случае тестовые наборы формируют путем анализа маршрутов, предусмотренных алгоритмом. Под ***маршрутами*** *при этом понимают последовательности операторов программы, которые* выполняются при конкретном варианте исходных данных.

Структурный подход к тестированию имеет ряд недостатков. Так тестовые наборы, построенные по данной стратегии:

- не обнаруживают пропущенных маршрутов;
- не обнаруживают ошибок, зависящих от обрабатываемых данных, например, в операторе if (a - b) < eps - пропуск функции абсолютного значения abs проявится только, если a<b;
- не дают гарантии, что программа правильна, например, если вместо сортировки по
- убыванию реализована сортировка по возрастанию.

# Критерии

Формирование тестовых наборов для тестирования маршрутов может осуществляться по нескольким критериям:

- покрытие операторов;
- покрытие решений (переходов);
- покрытие условий;
- покрытие решений/условий;
- комбинаторное покрытие условий.

# Функциональное тестирование

Одним из способов проверки программ является тестирование с управлением по данным или по принципу «черного ящика».

При функциональном тестировании различают следующие методы формирования тестовых наборов:

- эквивалентное разбиение;
- анализ граничных значений;
- анализ причинно-следственных связей;
- предположение об ошибке.

**Эквивалентное разбиение**

- Разработку тестов методом эквивалентного разбиения осуществляют в два этапа: на первом выделяют классы эквивалентности, а на втором - формируют тесты.

# Правила выделения классов эквивалентности

- если некоторый параметр x может принимать значения в интервале [1, 999], то выделяют один правильный класс 1 ≤ x ≤ 999 и два неправильных: x < I их>999;
- если входное условие определяет диапазон значений порядкового типа, например, «в автомобиле могут ехать от одного до шести человек», то определяется один правильный класс эквивалентности и два неправильных: ни одного и более шести человек;
- если входное условие описывает множество входных значений и есть основания полагать, что каждое значение программист трактует особо, например, «типы графических файлов: bmp, jpeg, vsd», то определяют правильный класс эквивалентности для каждого значения и один неправильный класс, например, txt;
- если входное условие описывает ситуацию «должно быть», например, «первым символом идентификатора должна быть буква», то определяется один правильный класс эквивалентности (первый символ -буква) и один неправильный (первый символ - не буква);
- если есть основание считать, что различные элементы класса эквивалентности трактуются программой неодинаково, то данный класс разбивается на меньшие классы эквивалентности.

# Функциональное тестирование

**Анализ граничных значений**

***Граничные значения*** *- это значения на границах классов* эквивалентности входных значений или около них. Анализ показывает, что в этих местах резко увеличивается возможность обнаружения ошибок.

- если входное условие описывает область значений, то следует построить тесты для границ области и тесты с неправильными входными данными для ситуаций незначительного выхода за границы области, например, если описана область [-1.0, +1.0], то должны быть сгенерированы тесты: -1.0, + 1.0,-1.001 и+1.001;
- если входное условие удовлетворяет дискретному ряду значений, то следует построить тесты для минимального и максимального значений и тесты, содержащие значения большие и меньшие этих двух значений, например, если входной файл может содержать от 1 до 255 записей, то следует проверить О, 1, 255 и 256 записей;
- если существуют ограничения выходных значений, то целесообразно аналогично тестировать и их: конечно не всегда можно получить результат вне выходной области, но тем не менее стоит рассмотреть эту возможность;
- если некоторое входное или выходное значение программы является упорядоченным множеством, например, это последовательный файл, линейный список или таблица, то следует сосредоточить внимание на первом и последнем элементах этого множества.

# Функциональное тестирование

**Анализ причинно-следственных связей**

- позволяет системно выбирать высокорезультативные тесты. Метод использует алгебру логики и оперирует понятиями «причина» и «следствие».
- ***Причиной*** *в данном случае называют отдельное входное* условие или класс эквивалентности.
- ***Следствием*** *- выходное условие или преобразование* системы.
- Идея метода заключается в отнесении всех следствий к причинам, т. е. в уточнении причинно-следственных связей.
- Данный метод дает полезный побочный эффект, позволяя обнаруживать неполноту и неоднозначность исходных спецификаций.

# Предположение об ошибке

- Процедура метода предположения об ошибке в значительной степени основана на интуиции. Основная его идея заключается в том, чтобы перечислить в некотором списке возможные ошибки или ситуации, в которых они могут появиться, а затем на основе этого списка составить тесты. Другими словами, требуется перечислить те особые случаи, которые могут быть не учтены при проектировании.

# Уровни и виды тестирования

- Модульное тестирование (component testing)
- Интеграционное тестирование (integration testing)
- Системное тестирование (system testing)
- Приемочное тестирование (acceptance testing) – пользователи
- smoke testing
- регрессионное тестирование

# Модульное тестирование (Unit testing)

- ***Модульное тестирование*** - 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.
- Модульное тестирование чаще всего проводится по принципу "белого ящика“.
- Модульное тестирование обычно подразумевает создание вокруг каждого модуля определенной среды

# Обнаруживаемые ошибки

- На уровне модульного тестирования проще всего обнаружить дефекты, связанные с алгоритмическими ошибками и ошибками кодирования алгоритмов.
- Ошибки, связанные с неверной трактовкой данных, некорректной реализацией интерфейсов, совместимостью, производительностью и т.п. обычно выявляются на более поздних стадиях тестирования.

# Интеграционное тестирование

- ***Интеграционное тестирование*** (тестирование сборки) - тестирование части системы, состоящей из двух и более модулей.
- Основная задача - поиск дефектов, связанных с ошибками в реализации и интерпретации взаимодействия между модулями.
- Так же, как и модульное тестирование, оперирует интерфейсами модулей и подсистем и требует создания тестового окружения
- Основная разница между модульным и интеграционным тестированием состоит в типах обнаруживаемых дефектов. В частности, на уровне интеграционного тестирования часто применяются методы, связанные с покрытием интерфейсов
- Интеграционное тестирование использует модель "белого ящика" на модульном уровне.

# Методы сборки модулей

- **Монолитный**, характеризующийся одновременным объединением всех модулей в тестируемый комплекс. Для замены неразработанных к моменту тестирования модулей необходимо дополнительно разрабатывать **драйверы** (**test driver**) и/или **заглушки** (**stub**)
- **Инкрементальный**, характеризующийся помодульным наращиванием комплекса программ с **пошаговым тестированием** собираемого комплекса.

В инкрементальном методе выделяют две стратегии добавления модулей:

- "Сверху вниз" (*нисходящее тестирование)*
- "Снизу вверх" (*восходящее тестирование)*
- «Сэндвич»

# Сравнение методов

- *Монолитное тестирование* требует больших трудозатрат, связанных с дополнительной разработкой драйверов и заглушек и со сложностью идентификации ошибок, проявляющихся в пространстве собранного кода.
- Монолитное тестирование предоставляет большие возможности распараллеливания работ, особенно на начальной фазе тестирования.
- *Пошаговое тестирование* связано с меньшей трудоемкостью идентификации ошибок за счет постепенного наращивания объема тестируемого кода и соответственно локализации добавленной области тестируемого кода.

# 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

# Недостатки восходящего тестирования

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

# Системное тестирование

- Основная *задача системного тестирования* - выявление дефектов, связанных с работой системы в целом:
  - отсутствующая или неверная функциональность
  - неверное использование ресурсов системы
  - непредусмотренные комбинации данных пользовательского уровня
  - несовместимость с окружением
  - непредусмотренные сценарии использования
  - неудобство в применении и тому подобное.
- Системное тестирование производится над проектом в целом с помощью метода «черного ящика».

# Категории тестов системного тестирования

1. Полнота решения функциональных задач.
2. Тестирование целостности (соответствие документации, комплектность).
3. Проверка инсталляции и конфигурации на разных платформах.
4. Оценка производительности.
5. Стрессовое тестирование - на предельных объемах нагрузки входного потока.
6. Корректность использования ресурсов (утечка памяти, возврат ресурсов).
7. Эффективность защиты от искажения данных и некорректных действий.
8. Корректность документации и т.д.

Объемы данных на этом уровне таковы, что обычно более эффективным подходом является полная или частичная *автоматизация тестирования*

# Другой пример разделения на категории:

- ***Функциональное тестирование*** (functional testing)
- ***Тестирование производительности*** (performance testing)
- ***Стрессовое тестирование*** (stress testing)
- ***Нагрузочное тестирование*** (load testing)
  - HP LoadRunner
- ***Тестирование удобства использования*** (usability testing)
- ***Тестирование интерфейса пользователя*** (UI testing)
- ***Тестирование безопасности*** (security testing)
- ***Тестирование локализации*** (localization testing)
- ***Тестирование совместимости*** (compatibility testing)

# Регрессионное тестирование

- Регрессионное тестирование - цикл тестирования, который производится при внесении изменений на фазе системного тестирования или сопровождения продукта.
- Главная проблема регрессионного тестирования - выбор между полным и частичным перетестированием и пополнение тестовых наборов. При частичном перетестировании контролируются только те части проекта, которые связаны с измененными компонентами.

# Исправление дефекта

- Получив отчет об ошибке, программист анализирует исходный код, находит ошибку, исправляет ее и модульно или интеграционно тестирует результат.
- В свою очередь тестировщик, проверяя внесенные программистом изменения, должен:
  - Проверить и утвердить исправление ошибки. Для этого необходимо выполнить указанный в отчете тест, с помощью которого была найдена ошибка.
  - Попробовать воспроизвести ошибку каким-нибудь другим способом.
  - Протестировать последствия исправлений. Возможно, что внесенные исправления привнесли ошибку (наведенную ошибку) в код, который до этого исправно работал.

# Комбинирование уровней тестирования

- В каждом конкретном проекте должны быть определены задачи, ресурсы и технологии для каждого уровня тестирования.
- Задача тестировщиков и менеджеров - оптимально распределить ресурсы между тремя уровнями тестирования так, чтобы каждый из возможных типов дефектов был «адресован» (в наборе тестов должны иметься тесты, направленные на выявление дефектов этого типа).
- Например, перенесение усилий на поиск фиксированного типа дефектов из области системного в область модульного тестирования может существенно снизить сложность и стоимость всего процесса тестирования.

| | Модульное | Интеграционное | Системное |
|---|---|---|---|
| **Типы дефектов** | Локальные дефекты | Интерфейсные дефекты | Отсутствующая функциональность, ошибки совместимости, документации, переносимости, проблемы производительности, инсталляции и т.п. |
| **Необходимость в системе тестирования** | Да | Да | Нет* |
| **Цена разработки системы тестирования** | Низкая | Низкая до умеренной | Умеренная до высокой или неприемлемой |
| **Цена процесса тестирования** | Низкая | Низкая | Высокая |

# Приемочное тестирование

***Приемочное тестирование*** *(Acceptance testing)* - тестирование готового продукта конечными пользователями в реальном окружении. Приемочные тесты разрабатываются пользователями (обычно в виде сценариев).