

Виды и методы тестирования на разных стадиях разработки ПО

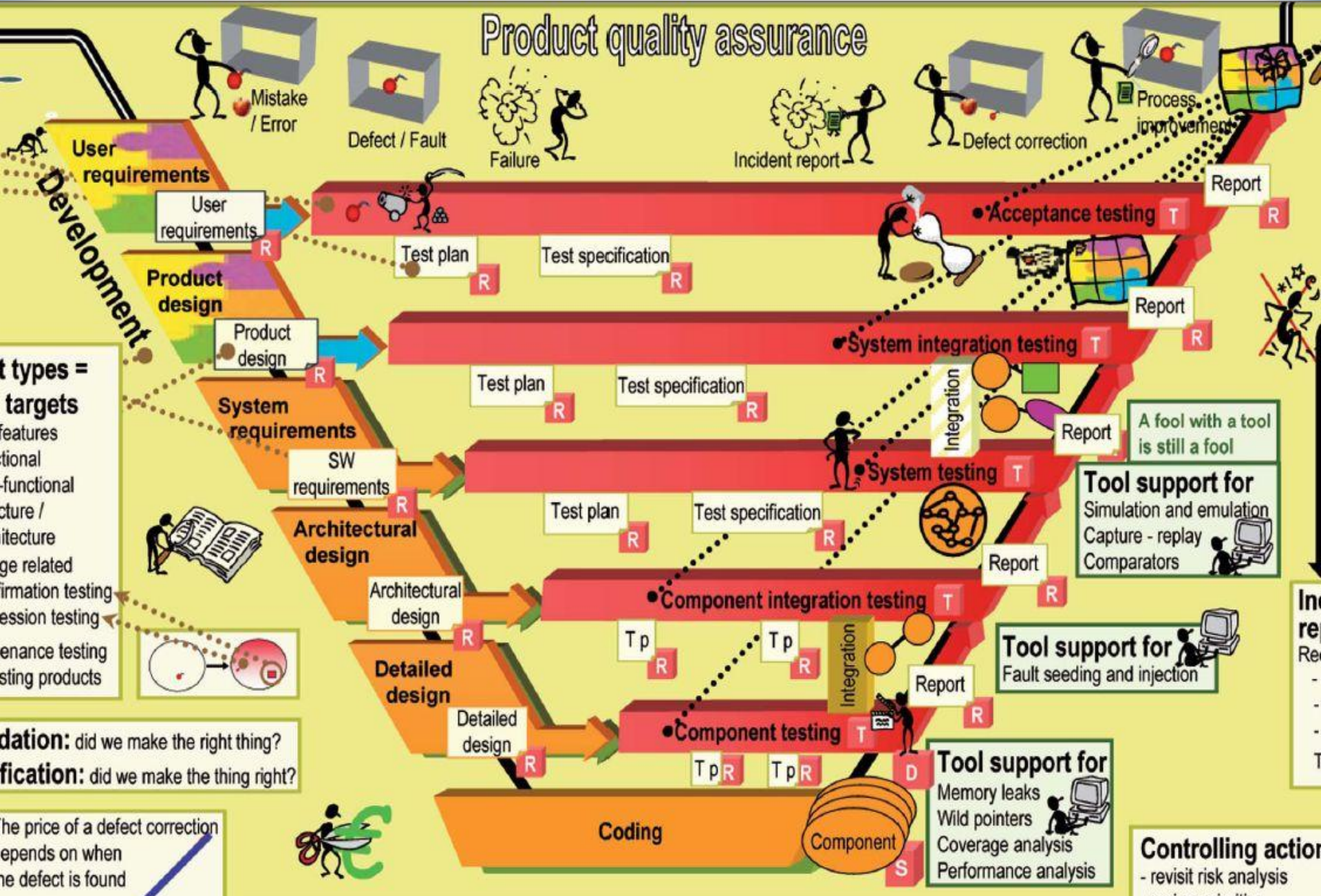
Уровни и виды тестирования

- Модульное тестирование (component testing)
- Интеграционное тестирование (integration testing)
- Системное тестирование (system testing)
- Приемочное тестирование (acceptance testing) – польз-ли

- smoke testing
- регрессионное тестирование

См. с. 144 Савина.

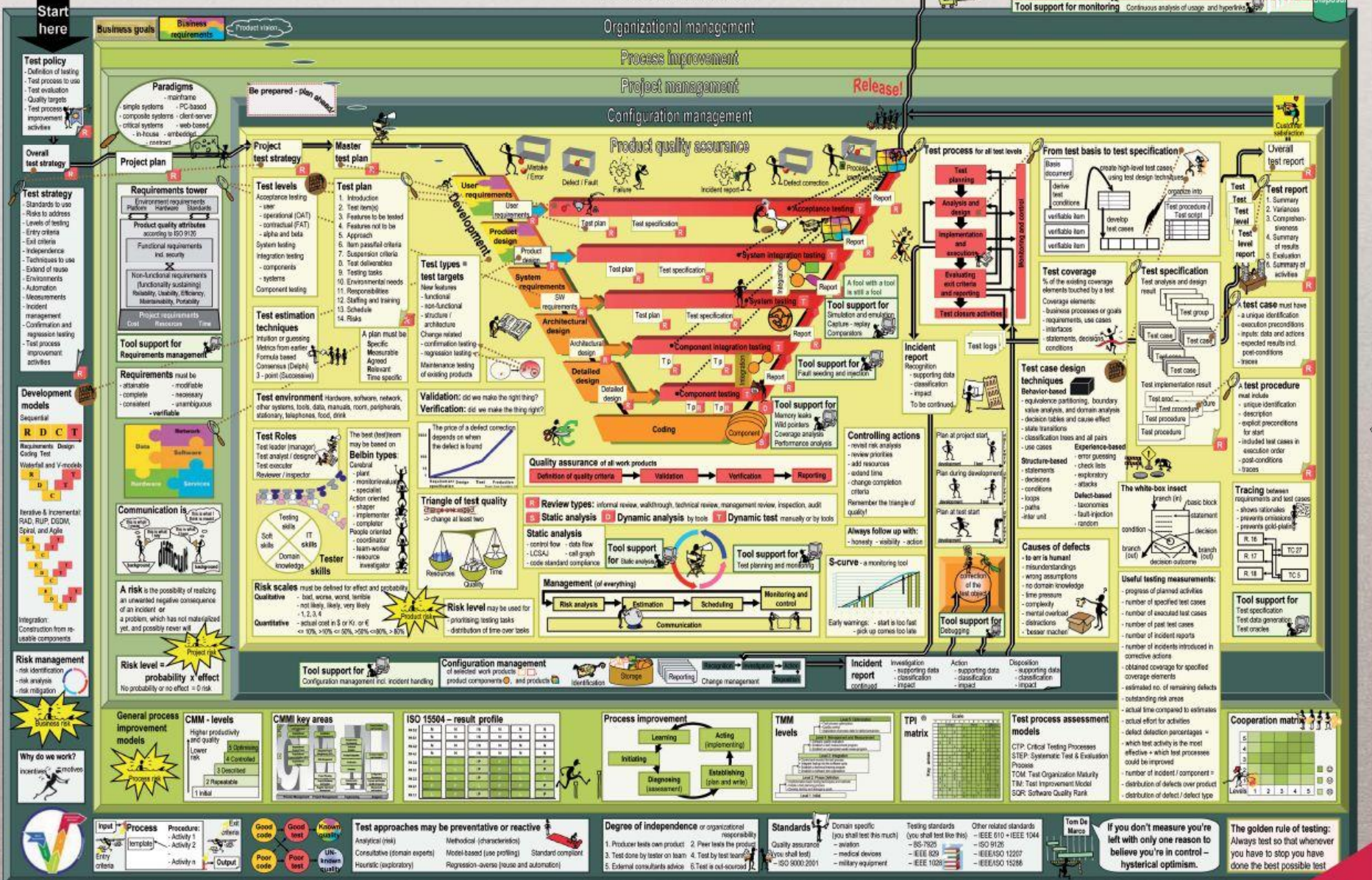
Взаимосвязь разработки и тестирования (V-диаграмма)



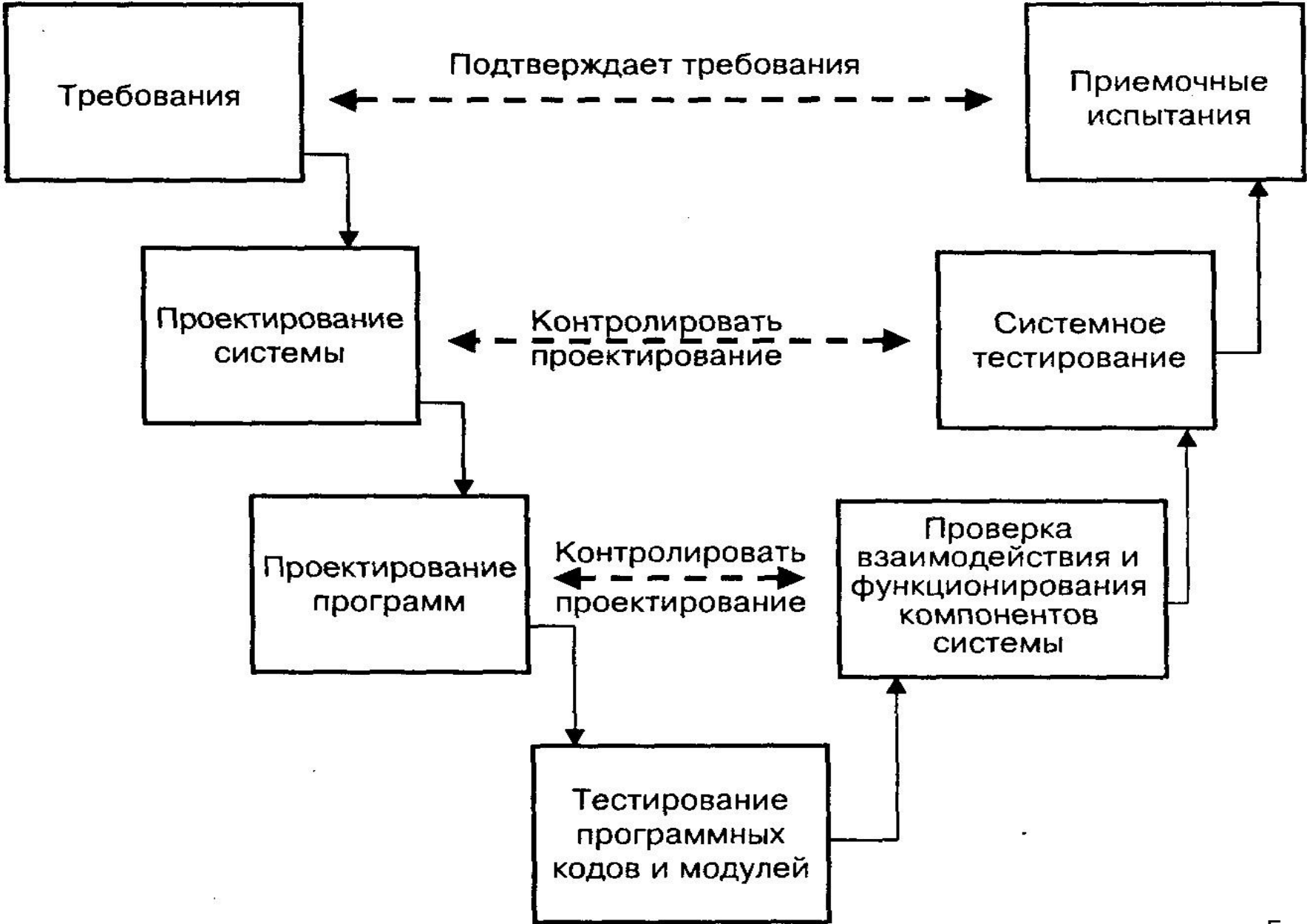
Software Testing at a Glance – or two

WWW.DELTAAXIOM.COM

The purpose of test:
 - Provides information to assure the quality of the product (finding defects)
 - decisions (quantifying risks)
 - process (finding root causes)



Взаимосвязь разработки и тестирования (V-диаграмма)



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.
- (белый и черный ящик)

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 протестированных модулей нижних уровней к уже протестированным модулям верхних уровней

Недостатки восходящего тестирования

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

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**Интеграцион
ное****Системное****Типы дефектов**Локальные
дефектыИнтерфейсные
дефектыОтсутствующая
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
умереннойУмеренная до
высокой или
неприемлемой**Цена процесса
тестирования**

Низкая

Низкая

Высокая

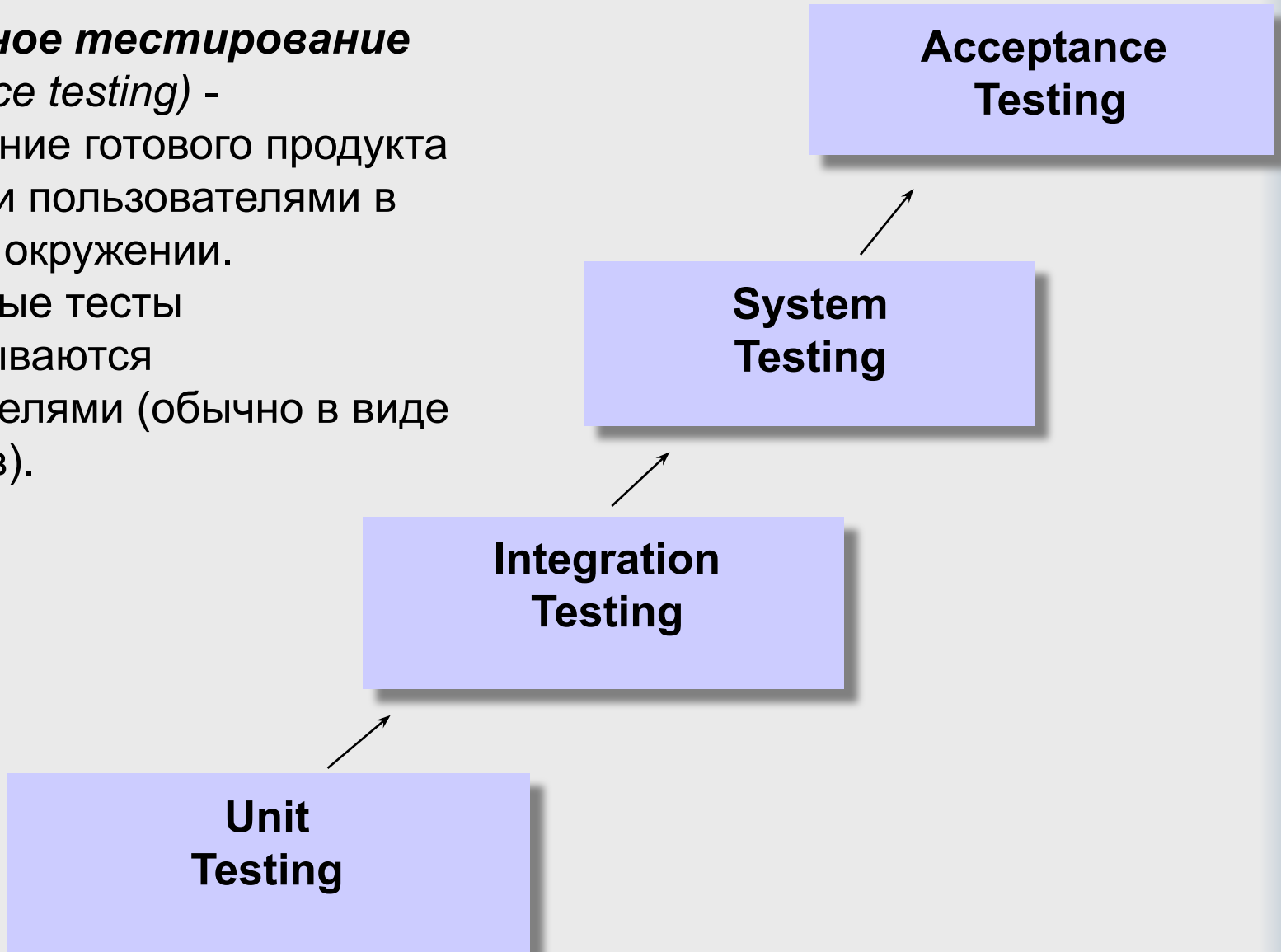
Приемочное тестирование

Приемочное тестирование

(Acceptance testing) -

тестирование готового продукта конечными пользователями в реальном окружении.

Приемочные тесты разрабатываются пользователями (обычно в виде сценариев).



Эвристические методы создания тестов

Простейший пример

- Программа выполняет ввод трех целых чисел
 - и выводит сообщение о том, является ли треугольник с такими сторонами неравносторонним, равнобедренным или равносторонним
1. правильный неравносторонний
 2. правильный равносторонний
 3. правильный равнобедренный
 4. по крайней мере 3 теста, представляющих правильные равнобедренные, полученные как перестановки двух разных сторон
 5. длина одной из сторон 0
 6. длина одной из сторон < 0
 7. три положительных числа, сумма двух из сторон равна третьей
 8. по крайней мере 3 теста со всеми тремя перестановками, когда сумма двух сторон равна третьей
 9. три положительных числа, сумма двух из сторон $<$ третьей
 10. по крайней мере 3 теста из категории 9
 11. все стороны = 0
 12. по крайней мере один тест, содержащий нецелые значения
 13. по крайней мере один тест, содержащий неверное кол-во значений
 14. и вообще: указаны ли ожидаемые результаты каждого теста?

Подход к созданию тестов на примере

Программа вводит два числа и выводит их сумму.

- В каждом из чисел 1 или 2 цифры
- Ввод каждого числа завершается Enter
- Ввод каждого числа отображается на экране
- После ввода числа выводится сумма.
- Программа запускается командой ADDER

- Первый тест - базовый
- Проблемы:
- Ввод запрашивается с помощью знака «?»
- - ош-ка пр-я: нет сопровод. инф-и, что вводить
- как остановить
- что за программа
- - ош-ка кодир-я: ответ в стороне от исх. дан

- $99 + 99$ 198
- $-99 + -99$ -198
- $99 + -14$ 85 большое первое может повлиять
на интерпр-ю второго
- $-38 + 99$ 61 - и +
- $56 + 99$
- $9 + 9$
- $0 + 0$
- $0 + 23$
- $-78 + 0$

(каждая цифра встречается 1 раз)

Классы тестов

- Классом можно назвать группу значений, которые программа обрабатывает одним и тем же способом. Граничные значения класса – те входные данные, на которых программа меняет свое поведение
- Не всегда программа меняет свое поведение там, где предполагается
- Границу нужно протестировать с двух сторон

- серия недопустимых значений
- серия проверки редактирования (стрелки, BS, Del)
- граничные условия
 - 100 + 100
 - цифра ли: коды от 48 до 57 (мб опечатка 75).
 - границы / (47) 0 9 : (58)

Фантазии:

- Enter + Enter
- 123456 + 0
- +1 + ___2 (пробелы – до и после числа)
- 1,2 + 5
- a + b
- Ctrl-A + Ctrl-B
- F1 + esc

Характеристики хорошего теста

- существует обоснованная вероятность выявления тестом ошибок
- не избыточен
- тестовый набор дб наилучшим в своей категории
- не дб слишком простым или слишком сложным

Некорректное поведение программы должно проявляться с достаточной очевидностью

Дорогие друзья! Возращивайте и лелейте в себе неисправимый пессимизм в отношении идеи о коде, свободном от багов.

Смотрите на код как на виртуальную вещь, которая в процессе тестирования послужит еще одним доказательством постулата о несовершенстве мира. (Р. Савин)

- Классы эквивалентности
- граничные условия
- тестирование переходов между состояниями
 - все меню и опции (трудно) => все вероятные последовательности действий пользователей
- Условия гонок и другие временные зависимости
 - запуск параллельно многих задач
 - нажатие клавиш не вовремя
 - тестирование производительности
- нагрузочное тестирование
- прогнозирование ошибок (не явл. границами, но могут вызвать сбой; интуиция) – error-guess testing

Виды тестов

- Базовый тест -- smoke test
(простой тестовый пример)
- Инвентаризация
(определить различные категории данных и создать тесты для каждого элемента категории)
- Комбинированные тесты
(скомбинировать различные входные данные)
- Граничные оценки
(оценить поведение программы при граничных значениях данных)
- Ошибочные данные
(оценить отклик системы на ввод неправильных данных)
- Нагрузочные тесты, создание напряжений
(попытаться вывести систему из строя)

Из Савина:

Методы генерирования тестов:

- 1. Черновик-чистовик (*dirty list-white list*);
- 2. Матричная раскладка (*matrices*);
- 3. Блок-схемы (*flowchart*).

Методы отбора тестов:

- 1. Оценка риска (*risk estimate*);
- 2. Эквивалентные классы (*equivalent classes*);
- 3. Пограничные значения (*boundary values*).

