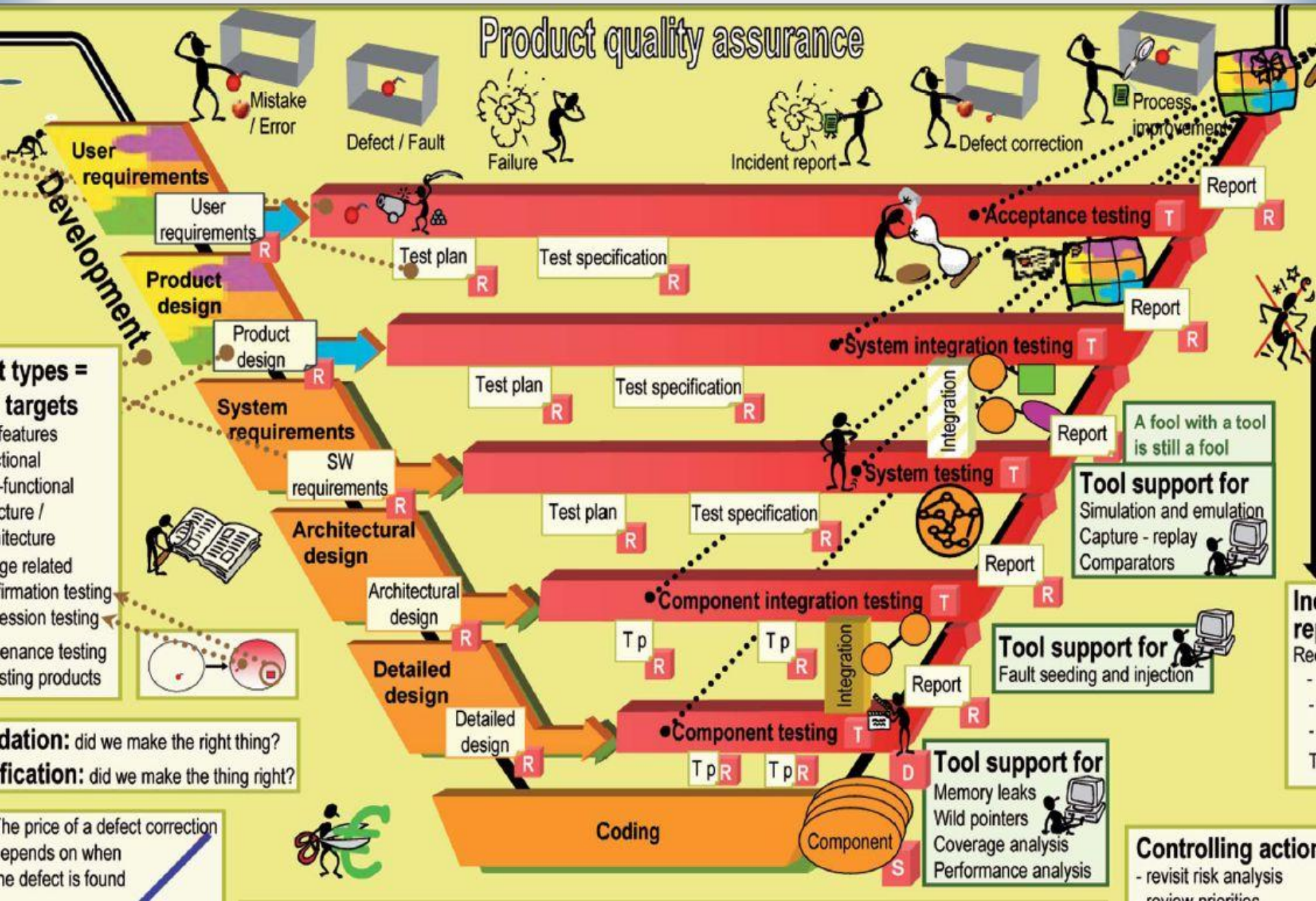
Виды и методы тестирования на разных стадиях разработки ПО

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

Взаимосвязь разработки и тестирования (V-диаграмма)



Software Testing at a Glance – or two

WWW.DELTAAXIOM.COM

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

Start here

Business goals | Business requirements | Product vision

Organizational management

Process improvement

Project management

Configuration management

Release!

Deployment
Maintenance testing
Tool support for monitoring — Continuous analysis of usage and hyperlinks
Time
Disposal

Test policy
- Definition of testing
- Test process to use
- Test evaluation
- Quality targets
- Test process improvement activities

Overall test strategy

Test strategy
- Standards to use
- Risks to address
- Levels of testing
- Entry criteria
- Exit criteria
- Independence
- Techniques to use
- Extend of reuse
- Environments
- Automation
- Measurements
- Incident management
- Confirmation and regression testing
- Test process improvement activities

Development models
Sequential: R D C T (Requirements, Design, Coding, Test)

Waterfall and V-models

Iterative & incremental: RAD, RUP, DSDM, Spiral, and Agile

Integration: Construction from re-usable components

Risk management
- risk identification
- risk analysis
- risk mitigation

Business risk

Why do we work?
incentives — motives

Paradigms
- mainframe
- simple systems — PC-based
- composite systems — client-server
- critical systems — web-based
- in-house — embedded
- contract

Be prepared - plan ahead!

Project plan

Requirements tower
Environment requirements: Platform, Hardware, Standards

Product quality attributes according to ISO 9126

Functional requirements: incl. security

Non-functional requirements (functionality sustaining): Reliability, Usability, Efficiency, Maintainability, Portability

Project requirements: Cost, Resources, Time

Tool support for Requirements management

Requirements must be
- attainable — modifiable
- complete — necessary
- consistent — unambiguous
- verifiable

Communication is

A risk is the possibility of realizing an unwanted negative consequence of an incident or a problem, which has not materialized yet, and possibly never will

Project risk

Risk level = probability x effect
No probability or no effect = 0 risk

Project test strategy

Master test plan

Test levels
Acceptance testing
- user
- operational (OAT)
- contractual (FAT)
- alpha and beta

System testing
Integration testing
- components
- systems

Component testing

Test plan
1. Introduction
2. Test item(s)
3. Features to be tested
4. Features not to be
5. Approach
6. Item passfail criteria
7. Suspension criteria
8. Test deliverables
9. Testing tasks
10. Environmental needs
11. Responsibilities
12. Staffing and training
13. Schedule
14. Risks

A plan must be: Specific, Measurable, Agreed, Relevant, Time specific

Test estimation techniques
Intuition or guessing
Metrics from earlier
Formula based
Consensus (Delphi)
3 - point (Successive)

Test environment
Hardware, software, network, other systems, tools, data, manuals, room, peripherals, stationary, telephones, food, drink

Test Roles
Test leader (manager)
Test analyst / designer
Test executer
Reviewer / inspector

The best (test)team may be based on **Belbin types:**
Cerebral
- plant
- monitorevaluator
- specialist

Action oriented
- shaper
- implementer
- completer

People oriented
- coordinator
- team-worker
- resource investigator

Tester skills: Testing skills, IT skills, Soft skills, Domain knowledge

Risk scales must be defined for effect and probability
Qualitative - bad, worse, worst, terrible - not likely, likely, very likely - 1, 2, 3, 4
Quantitative - actual cost in $ or Kr. or € - <= 10%, >10% <= 50%, >50% <=80%, > 80%

Product risks

Risk level may be used for
- prioritising testing tasks
- distribution of time over tasks

Product quality assurance

Development

User requirements → Acceptance testing
Product design → System integration testing
System requirements → System testing
SW requirements
Architectural design → Component integration testing
Detailed design → Component testing
Coding

Mistake / Error — Defect / Fault — Failure — Incident report — Defect correction — Process improvement

Test plan — Test specification — Report — Integration

Test types = test targets
New features
- functional
- non-functional
- structure / architecture

Change related
- confirmation testing
- regression testing

Maintenance testing of existing products

Validation: did we make the right thing?
Verification: did we make the thing right?

The price of a defect correction depends on when the defect is found

Triangle of test quality
Change one aspect
-> change at least two
Resources — Quality — Time

Quality assurance of all work products
Definition of quality criteria → Validation → Verification → Reporting

R Review types: informal review, walkthrough, technical review, management review, inspection, audit
S Static analysis — D Dynamic analysis by tools — T Dynamic test manually or by tools

Static analysis
- control flow — data flow
- LCSAJ — call graph
- code standard compliance

Tool support for Static analysis

Tool support for Test planning and monitoring

Management (of everything)
Risk analysis → Estimation → Scheduling → Monitoring and control
Communication

Tool support for
Simulation and emulation
Capture - replay
Comparators

A fool with a tool is still a fool

Tool support for
Fault seeding and injection

Tool support for
Memory leaks
Wild pointers
Coverage analysis
Performance analysis

Controlling actions
- reveal risk analysis
- review priorities
- add resources
- extend time
- change completion criteria

Remember the triangle of quality!

Always follow up with:
- honesty - visibility - action

S-curve - a monitoring tool
Early warnings: - start is too fast - pick up comes too late

Tool support for Debugging

Test process for all test levels
Test planning
Analysis and design
Implementation and execution
Evaluating exit criteria and reporting
Test closure activities

Monitoring and control

Test logs

Incident report
Recognition
- supporting data
- classification
- impact

To be continued...

From test basis to test specification
Basis document → derive test conditions → verifiable item
develop test cases → organize into → Test procedure / Test script

Test coverage
% of the existing coverage elements touched by a test

Coverage elements:
- business processes or goals
- requirements, use cases
- interfaces
- statements, decisions, conditions

Test specification
Test analysis and design result → Test group → Test case → Test procedure

Test implementation result → Test procedure

Test case design techniques
Behavior-based
- equivalence partitioning, boundary value analysis, and domain analysis
- decision tables and cause effect
- state transitions
- classification trees and all pairs
- use cases

Structure-based
- statements
- decisions
- conditions
- loops
- paths
- intar unit

Experience-based
- error guessing
- check lists
- exploratory
- attacks

Defect-based
- taxonomies
- fault-injection
- random

Causes of defects
- to err is human!
- misunderstandings
- wrong assumptions
- no domain knowledge
- time pressure
- complexity
- mental overload
- distractions
- bessar machen

The white-box insect
branch (in) — basic block — statement — decision — branch (out) — decision outcome — condition

Useful testing measurements:
- progress of planned activities
- number of specified test cases
- number of executed test cases
- number of pasit test cases
- number of incident reports
- number of incidents introduced in corrective actions
- obtained coverage for specified coverage elements
- estimated no. of remaining defects
- outstanding risk areas
- actual time compared to estimates
- actual effort for activities
- defect detection percentages =
- which test activity is the most effective + which test processes could be improved
- number of incident / component =
- distribution of defects over product
- distribution of defect / defect type

Overall test report

Test report
1. Summary
2. Variances
3. Comprehensiveness
4. Summary of results
5. Evaluation
6. Summary of activities

A test case must have
- a unique identification
- execution preconditions
- inputs: data and actions
- expected results incl. post-conditions
- traces

A test procedure must include
- unique identification
- description
- explicit preconditions for start
- included test cases in execution order
- post-conditions
- traces

Tracing between requirements and test cases
- shows rationales
- prevents omissions
- prevents gold-plating

R.16 — TC 27
R.17 — TC 5
R.18

Tool support for
Test specification
Test data generation
Test oracles

Tool support for Configuration management incl. incident handling

Configuration management of selected work products: product components, and products
Identification — Storage — Reporting — Recognition — Investigation — Action — Disposition — Change management

Incident report continued
Investigation - supporting data - classification - impact
Action - supporting data - classification - impact
Disposition - supporting data - classification - impact

General process improvement models
Process risk

CMM - levels
Higher productivity and quality
Lower risk
5 Optimizing
4 Controlled
3 Described
2 Repeatable
1 Initial

CMMI key areas

ISO 15504 – result profile

Process improvement
Learning — Acting (implementing) — Establishing (plan and write) — Diagnosing (assessment) — Initiating

TMM levels

TPI matrix

Test process assessment models
CTP: Critical Testing Processes
STEP: Systematic Test & Evaluation Process
TOM: Test Organization Maturity
TIM: Test Improvement Model
SQR: Software Quality Rank

Cooperation matrix
Levels 1 2 3 4 5

Input — Process — Output — Entry criteria — Exit criteria — Procedure: Activity 1, Activity 2, Activity n — template

Good code / Good test / Known quality
Poor code / Poor test / UN-known quality

Test approaches may be preventative or reactive
Analytical (risk)
Consultative (domain experts)
Heuristic (exploratory)
Methodical (characteristics)
Model-based (use profiling)
Regression-averse (reuse and automation)
Standard compliant

Degree of independence or organizational responsability
1. Producer tests own product
2. Peer tests the product
3. Test done by tester on team
4. Test by test team
5. External consultants advice
6. Test is out-sourced

Standards
Quality assurance (you shall test)
- ISO 9000:2001

Domain specific (you shall test like this)
- aviation
- medical devices
- military equipment

Testing standards (you shall test like this)
- BS-7925
- IEEE 829
- IEEE 1028

Other related standards
- IEEE 610 + IEEE 1044
- ISO 9126
- IEEE/ISO 12207
- IEEE/ISO 15288

Tom De Marco

If you don't measure you're left with only one reason to believe you're in control – hysterical optimism.

The golden rule of testing: Always test so that whenever you have to stop you have done the best possible test

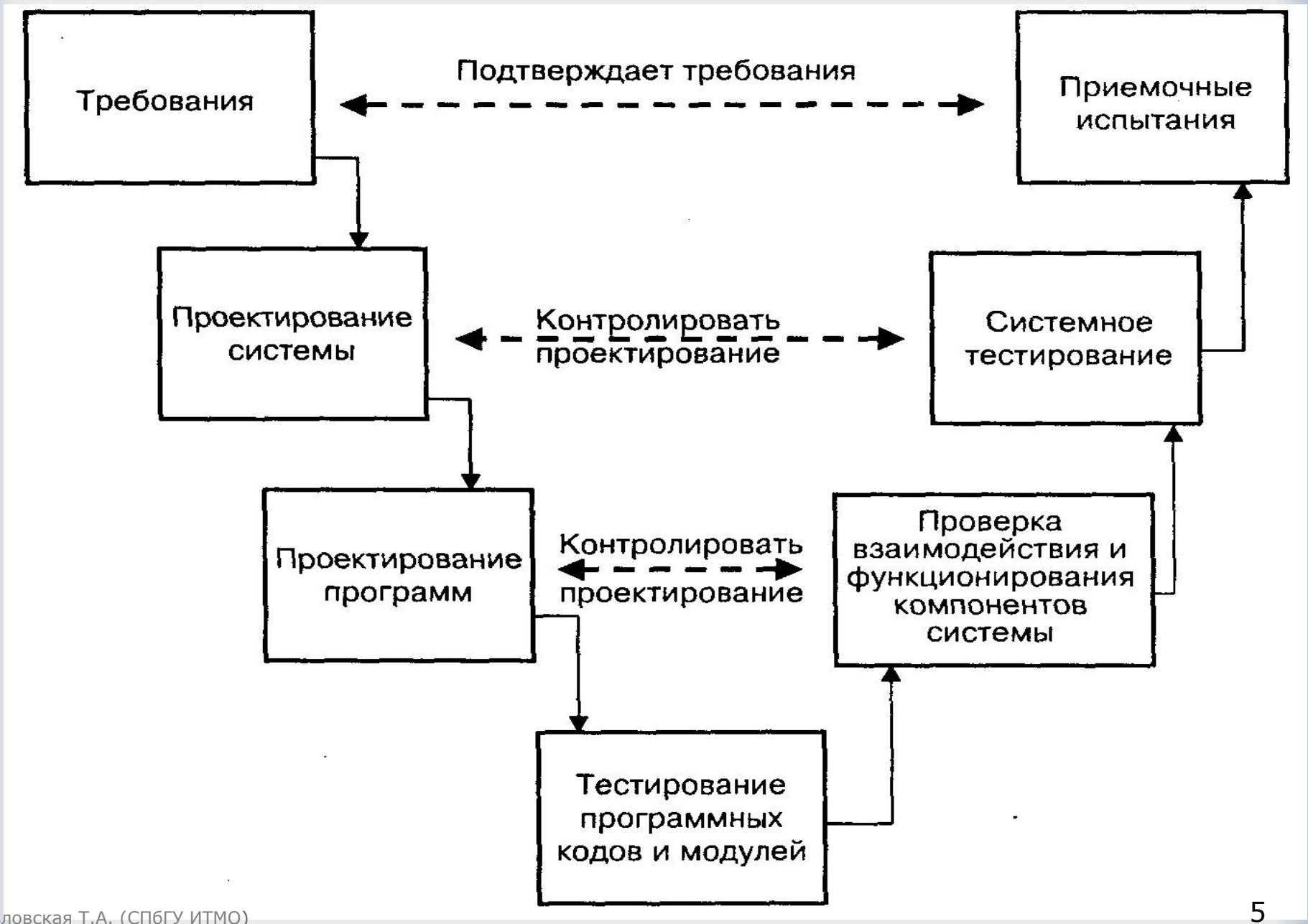
DELTA AXIOM

Test is comparing what is to what should be

DELTA

Graphic advice: SEB-CONSULT, Denmark — Screen Beans®/Clip Art, USA — V. T-UK-0707

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 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

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.

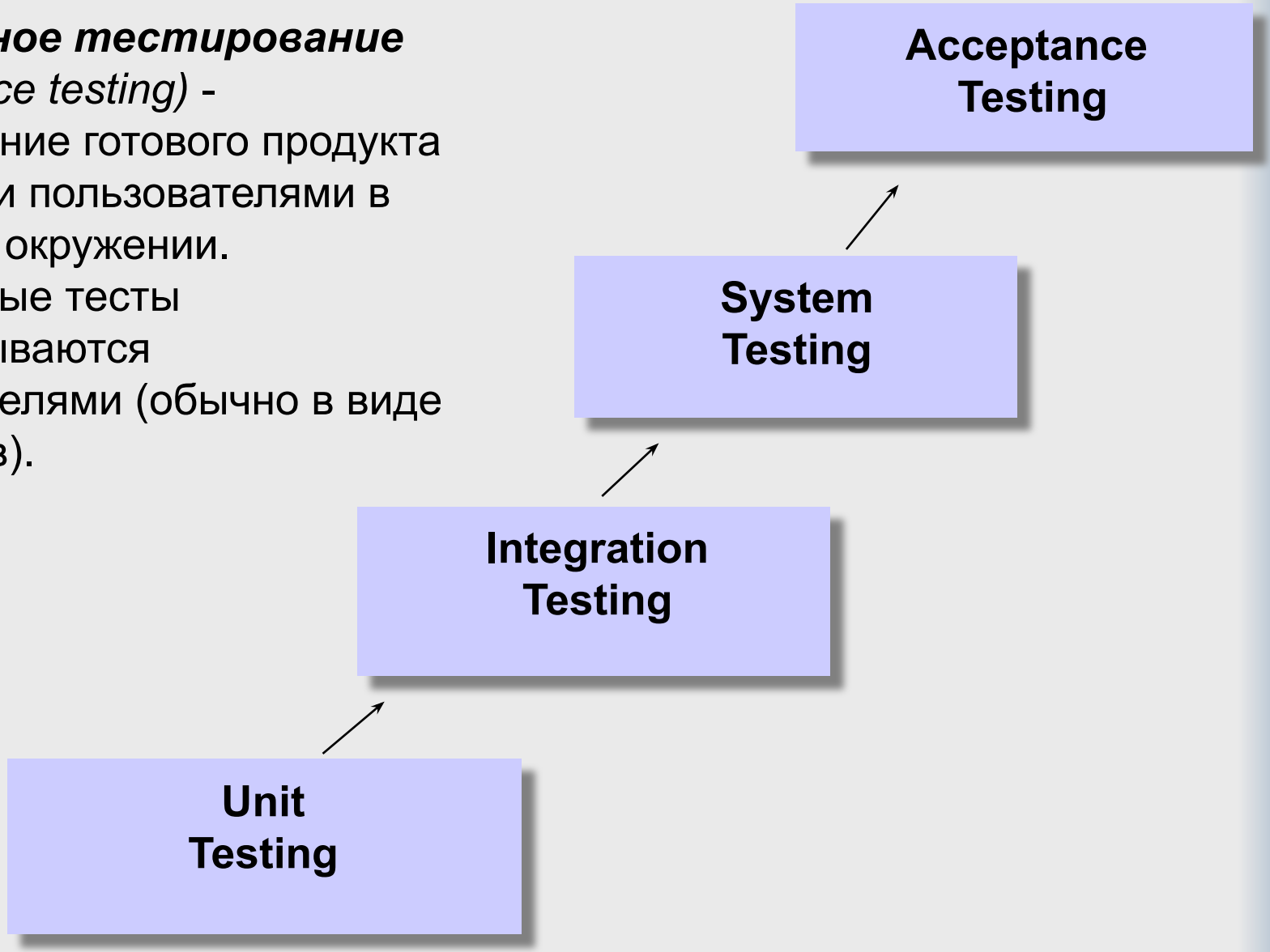
Комбинирование уровней тестирования

- 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).



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 переставновки 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 числе 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! Взращивайте 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).*