Test documentation and Test case design

Iana Mourza
QA Lead/Release Lead
VMware, Inc.

2008

Testing in Software Development

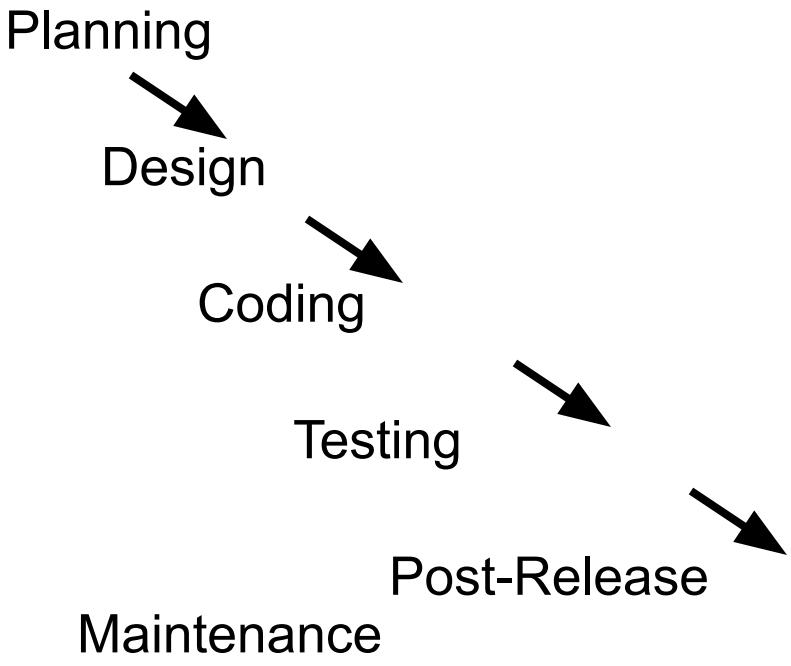
- **Testing**

  = process of searching for software errors

- **How and when do we start?**

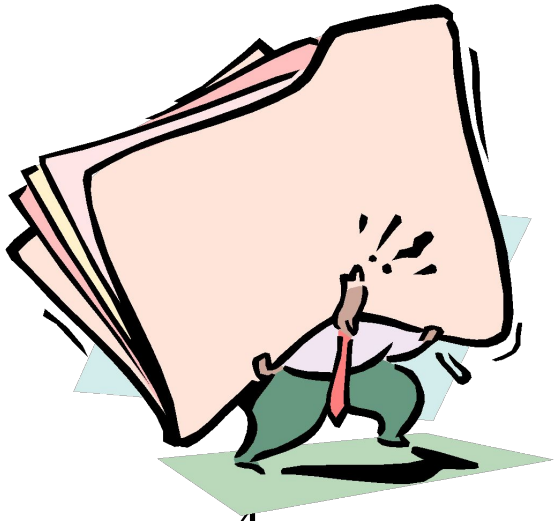
Software Development

- Software Development Life Cycle:



Software documentation

- PRD (Product Requirement Document)
- FS (Functional Specification)
- UI Spec (User Interface Specification)
- Test Plan
- Test Case
- Test Suite
- Traceability matrix
- Risk Analysis matrix

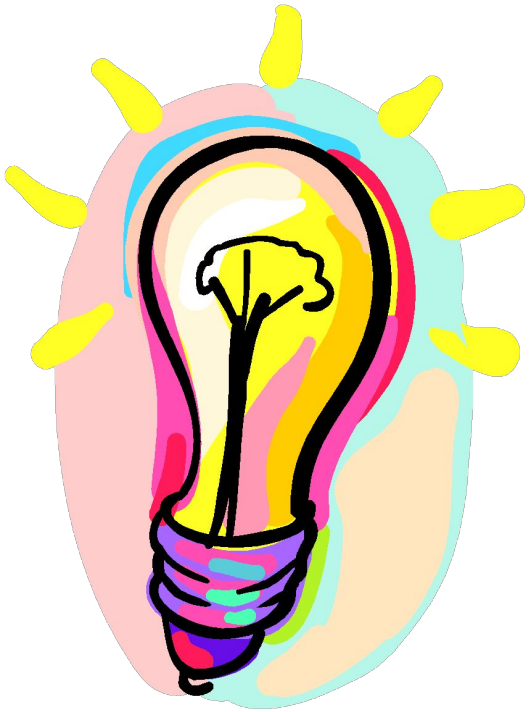


Software documentation

- **PRD (Product Requirement Document)**
  - What: set of software requirements
  - Who: Product Marketing, Sales, Technical Support
  - When: planning stage
  - Why: we need to know what the product is supposed to do
  - QA role:
    - Participate in reviews
    - Analyze for completeness
    - Spot ambiguities
    - Highlight contradictions
    - Provide feedback on features/usability

Software documentation

- **PRD (example)**

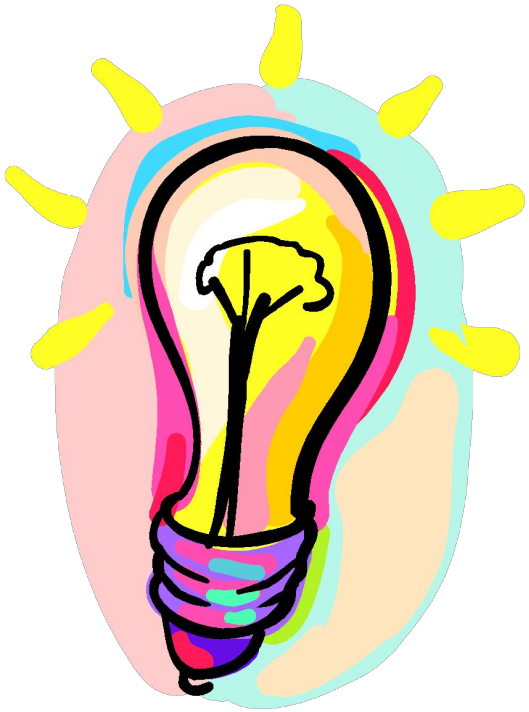


Software documentation

- **FS (Functional Specification)**
  - What: software design document;
  - Who: Engineering, Architects;
  - When: (planning)/design/(coding) stage(s);
  - Why: we need to know how the product will be designed;
  - QA role:
    - Participate in reviews;
    - Analyze for completeness;
    - Spot ambiguities;
    - Highlight contradictions.

Software documentation

- **FS (example)**



Test documentation

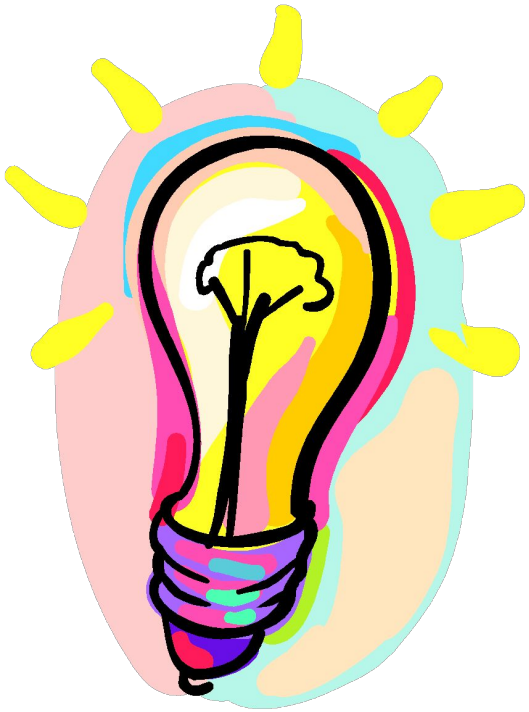
- **Test Plan**
  - What: a document describing the scope, approach, resources and schedule of intended testing activities; identifies test items, the features to be tested, the testing tasks, who will do each task and any risks requiring contingency planning;
  - Who: QA;
  - When: (planning)/design/coding/testing stage(s);

Test documentation

- **Test Plan (cont'd)**
  - Why:
    - Divide responsibilities between teams involved; if more than one QA team is involved (ie, manual / automation, or English / Localization) – responsibilities between QA teams ;
    - Plan for test resources / timelines ;
    - Plan for test coverage;
    - Plan for OS / DB / software deployment and configuration models coverage.
  - QA role:
    - Create and maintain the document;
    - Analyze for completeness;
    - Have it reviewed and signed by Project Team leads/managers.

Test documentation

- **Test Plan (example)**



Test documentation

- **Test Case**
  - What: a set of inputs, execution preconditions and expected outcomes developed for a particular objective, such as exercising a particular program path or verifying compliance with a specific requirement;
  - Who: QA;
  - When: (planning)/(design)/coding/testing stage(s);
  - Why:
    - Plan test effort / resources / timelines;
    - Plan / review test coverage;
    - Track test execution progress;
    - Track defects;
    - Track software quality criteria / quality metrics;
    - Unify Pass/Fail criteria across all testers;
    - Planned/systematic testing vs Ad-Hoc.

Test documentation

- **Test Case (cont'd)**
  - Five required elements of a Test Case:
    - ID – unique identifier of a test case;
    - Features to be tested / steps / input values – what you need to do;
    - Expected result / output values – what you are supposed to get from application;
    - Actual result – what you really get from application;
    - Pass / Fail.

Test documentation

- **Test Case (cont'd)**
  - Optional elements of a Test Case:
    - Title – verbal description indicative of testcase objective;
    - Goal / objective – primary verification point of the test case;
    - Project / application ID / title – for TC classification / better tracking;
    - Functional area – for better TC tracking;
    - Bug numbers for Failed test cases – for better error / failure tracking (ISO 9000);
    - Positive / Negative class – for test execution planning;
    - Manual / Automatable / Automated parameter etc – for planning purposes;
    - Test Environment.

Test documentation

- **Test Case (cont'd)**
  - Inputs:
    - Through the UI;
    - From interfacing systems or devices;
    - Files;
    - Databases;
    - State;
    - Environment.
  - Outputs:
    - To UI;
    - To interfacing systems or devices;
    - Files;
    - Databases;
    - State;
    - Response time.

Test documentation

- **Test Case (cont'd)**
  - Format – follow company standards; if no standards – choose the one that works best for you:
    - MS Word document;
    - MS Excel document;
    - Memo-like paragraphs (MS Word, Notepad, Wordpad).
  - Classes:
    - Positive and Negative;
    - Functional, Non-Functional and UI;
    - Implicit verifications and explicit verifications;
    - Systematic testing and ad-hoc;

Test documentation

- **Test Case (exercise)**

Test documentation

- **Test Case (example)**

Test documentation

- **Test Suite**
  - A document specifying a sequence of actions for the execution of multiple test cases;
  - Purpose: to put the test cases into an executable order, although individual test cases may have an internal set of steps or procedures;
  - Is typically manual, if automated, typically referred to as test script (though manual procedures can also be a type of script);
  - Multiple Test Suites need to be organized into some sequence – this defined the order in which the test cases or scripts are to be run, what timing considerations are, who should run them etc.

Test documentation

- **Traceability matrix**
  - What: document tracking each software feature from PRD to FS to Test docs (Test cases, Test suites);
  - Who: Engineers, QA;
  - When: (design)/coding/testing stage(s);
  - Why: we need to make sure each requirement is covered in FS and Test cases;
  - QA role:
    - Analyze for completeness;
    - Make sure each feature is represented;
    - Highlight gaps.

Test documentation

- **Traceability matrix (example)**

PRD Section	FS Section	Test case	Notes
1.1. Validation of user login credentials.	4.1. User login validation.	6.1.4. User login with proper credentials.	
		6.1.5. User login with invalid username.	
		6.1.6. User login with invalid password.	
1.2. Validation of credit card information.	7.2.4. Credit card information verification.	10.1.1. Valid credit card information input.	
		10.1.2. Invalid credit card number.	
		10.1.3. Invalid credit card name.	
		…	

Test design

- **Testing Levels**
  - Various development models are there in the market
  - Within each development model, there are corresponding levels/stages of testing
  - There are four basic levels of testing that are commonly used within various models:
    - Component (unit) testing
    - Integration testing
    - System testing
    - Acceptance testing

Test design

- **Testing Levels**
  - **Acceptance testing:** Formal testing with respect to user needs, requirements, and business processes conducted to determine whether or not a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity to determine whether or not to accept the system.
  - **System testing:** The process of testing an integrated system to verify that it meets specified requirements.
  - **Integration testing:** Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems.
  - **Component testing:** The testing of individual software components.

Test design

- **Testing Strategies**
  - Depend on Development model.
  - **Incremental:** testing modules as they are developed, each piece is tested separately. Once all elements are tested, integration/system testing can be performed.
    - Requires additional code to be written, but allows to easily identify the source of error
  - **Big Bang**: testing is performed on fully integrated system, everything is tested with everything else.
    - No extra code needed, but errors are hard to find.

Test design

- **Test Types**
  - There are several key types of tests that help improve the focus of the testing:
    - Functional testing: testing specific functions
    - Non-functional testing: testing characteristics of the software and system
    - Structural testing: testing the software structure or architecture
    - Re-testing (confirmation) and regression testing: testing related to changes
    - White and black box testing
  - Each test type focuses on a particular test objective
  - Test objective: A reason or purpose for designing and executing a test.
  - Test object: The component or system to be tested.
  - Test item: The individual element to be tested. There usually is one test object and many test items.

Test design

Test Types (cont'd)

- **Functional testing**:
- Testing based on an analysis of the specification of the functionality of a component or system.
- The functions are "what" the system does:
  - They are typically defined or described in work products such as a requirements specification, use cases, or a functional specification;
  - They may be undocumented;
  - Functional tests are based on both explicit and implicit features and functions;
  - They may occur at all test levels, e.g., tests for components may be based on a component specification;
- Functional testing focuses on the external behavior of the software (black-box testing).

Test design

- **Test Types (cont'd)**
- **Non-Functional testing**:
  - Focuses on "how" the system works;
  - Non-functional tests are those tests required to measure characteristics of systems and software that can be quantified;
  - These quantifications can vary and include items such as: response times, throughput, capacity for performance testing etc.
  - Testing the attributes of a component or system that do not relate to functionality, e.g. reliability, efficiency, usability, maintainability, compatibility and portability.

Test design

- **Test Types (cont'd)**

– **Structural (White box) testing**:
  – Testing based on an analysis of the internal structure of the component or system / architecture of the system, aspects such as a calling hierarchy, data flow diagram, design specification, etc.;
  – May may be performed at all test levels - system, system integration, or acceptance levels (e.g., to business models or menu structures);
  – Structural techniques are best used after specification-based techniques;
  – Can assist in measuring the thoroughness of testing by assessing the degree of coverage of a structure;
  – Tools can be used to measure the code coverage.

Test design

- **Test Types (cont'd)**

- **Black box testing**:
  - The program is treated as black box;
  - Inputs are fed into the program, outputs observed;
  - Search for interesting and challenging input combinations and conditions – they are most likely to expose an error.

Test design

- **Test Types (cont'd)**

- **Regression testing (retesting)**:
  - Retesting of a previously tested program following modification to ensure existing functionality is working properly and new defects/faults have not been introduced or uncovered as a result of the changes made;
  - Tests should be designed to be repeatable – they are to be used for retesting; the more defects found, the more often the tests may have to run;
  - Full regression / Partial regression / No regression – the extent of regression is based on the risk of not funding defects;
  - Applies to functional, non-functional and structural testing;
  - Good candidate for automation.

Test design

- **Static Test Techniques**

– **Static Testing**:
  – Testing of a component or system at specification or implementation level without execution of the software;
  – Non-execution based method for checking life cycle artifacts;
  – Manual static techniques: reviews (inspections, walkthroughs etc.), formal and informal;
  – Automated techniques: supporting reviews, static analysis tools (compilers);
  – Anything can be reviewed – PRD, Specs, Memos, Proposals, User Guides etc.

Test design

- **Test Case optimization**
  - Optimizing Test design and planning methodologies:
    - Boundary testing;
    - Equivalence classes;
    - Decision tables;
    - State transitional diagrams;
    - Risk Analysis.

Test design

- **Equivalence class partitioning :**

– A black box test design technique in which test cases are designed to execute representatives from equivalence partitions. In principle, test cases are designed to cover each partition at least once.

– Creates the minimum number of black box tests needed to provide minimum test coverage

– Steps:

  – Identify equivalence classes, the input values which are treated the same by the software:

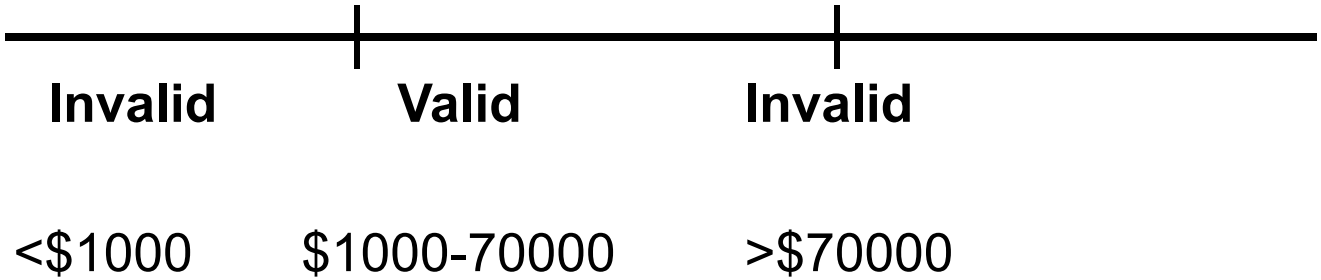
    – Valid classes: legal input values;

    – Invalid classes: illegal or unacceptable input values;

  – Create a test case for each equivalence class.

Test design

- **Equivalence class partitioning (cont'd):**

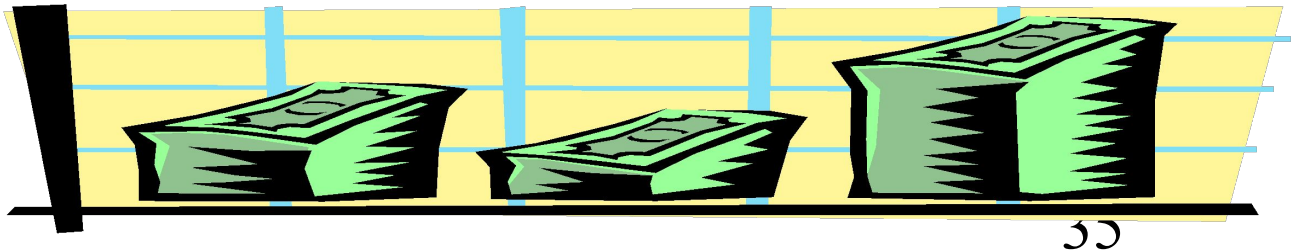


Equivalence partition (class):

– A portion of an input or output domain for which the behavior of a component or system is assumed to be the same, based on the specification.

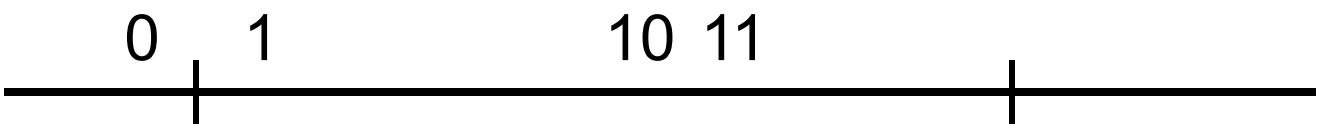
Test design

- **Boundary value testing:**
  - A black box test design technique in which test cases are designed based on boundary values.
  - Each input is tested at both ends of its valid range(s) and just outside its valid range(s). This makes sense for numeric ranges and can be applied to non-numeric fields as well. Additional issues, such as field length for alphabetic fields, can come into play as boundaries.
  - Boundary value: An input value or output value, which is on the edge of an equivalence partition or at the smallest incremental distance on either side of an edge, for example the minimum or maximum value of a range.



Test design

- **Boundary value testing (cont'd):**
  - Run test cases at the boundary of each input:
    - Just below the boundary;
    - Just above the boundary;
  - The focus is on one requirement at a time;



  - Can be combined across multiple requirements – all valid minimums together, all valid maximums together;
  - Invalid values should not be combined.

Test design

- **Decision table:**
  - composed of rows and columns, separated into quadrants:

Conditions	Condition Alternatives
Actions	Action Entries

Test design

- **Decision table:**

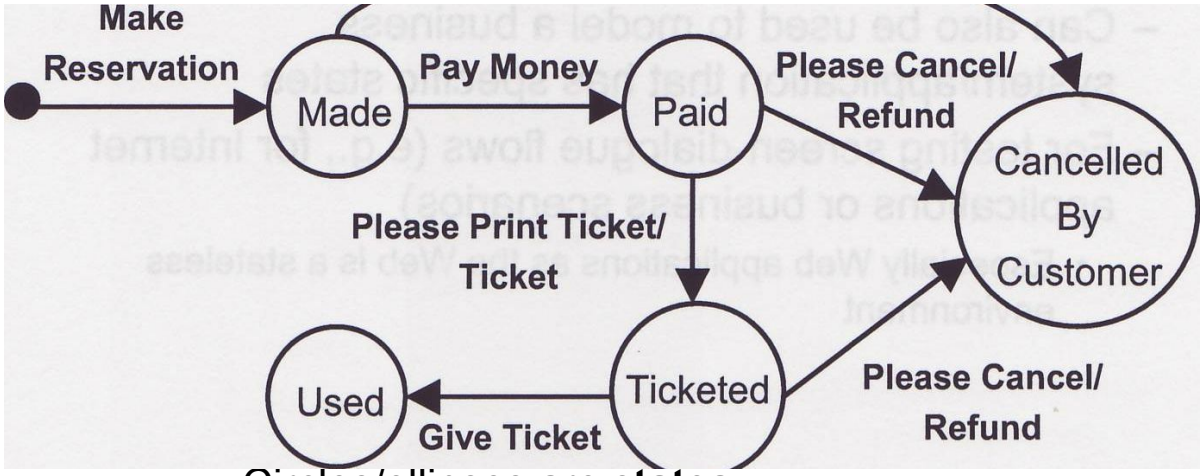
	Conditions/ Courses of Action	Rules					
		1	2	3	4	5	6
Condition Stubs	Employee type	S	H	S	H	S	H
	Hours worked	<40	<40	40	40	>40	>40
Action Stubs	Pay base salary	X		X		X	
	Calculate hourly wage		X		X		X
	Calculate overtime						X
	Produce Absence Report		X				

Test design

- **State transitional diagrams:**
  - Identify a finite number of *states* the model execution goes through
  - Create a state *transition* diagram showing how the model transitions from one state to the other
  - Assess the model accuracy by *analyzing* the conditions under which a state change occurs
  - **State transition:** A transition between two states of a component or system.

Test design

- **State transitional diagrams (cont'd):**



  - Circles/ellipses are s**tates**
  - Lines represent **transitions** between states
  - Text represents the **events** that cause transitions
  - The solid circle represents an **initial state**
  - A solid circle or ellipses/circles with no exit lines (transitions) are **final states**
  - In the example above, minimal number of test cases to cover each state is two

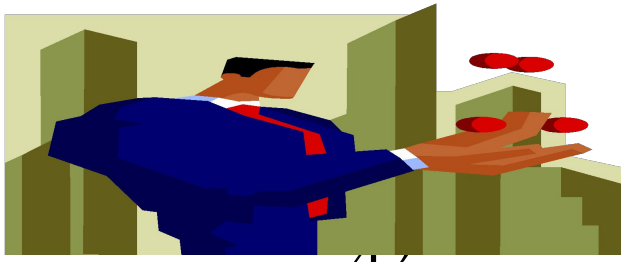
Test design

- **Risk Analysis:**
  - **What:** The process of assessing identified risks to estimate their impact and probability of occurrence (likelihood).
    - Likelihood = The probability or chance of an event occurring (e.g., the likelihood that a user will make a mistake and, if a mistake is made, the likelihood that it will go undetected by the software)
    - Impact = The damage that results from a failure (e.g., the system crashing or corrupting data might be considered high impact)

Test design

Risk Analysis (cont'd):

- **Who:** PM, Tech Support, Sales, Engineers, QA;
- **When:** (design)/coding/testing stage(s);
- **Why:**
  - It helps us choose the best test techniques
  - It helps us define the extent of testing to be carried out
  - The higher the risk, the more focus given
  - It allows for the prioritization of the testing
  - Attempt to find the critical defects as early as possible
  - Are there any non-testing activities that can be employed to reduce risk? e.g., provide training to inexperienced personnel



Test design

- **Risk Analysis (scale 1-10):**

Feature	Likelihood of failure (Eng, QA, Tech Support)	Impact of failure (PM, Tech Support, Sales, QA)	Risk Factor
1. Login	2	10	20
2. Create database record	5	7	35
3. Modify database record	3	6	18

Test design

- **Risk Analysis (example)**

Test documentation

- **Catching an Error -> Bug Report**
  - Reproducing an error;
  - Reporting an error:
    - Bug report – main elements:
      - ID #;
      - What is the problem (what happened);
      - Where the problem occurred;
      - Steps to reproduce.
    - Bug report – structure:
      - ID;
      - Title/short description;
      - Long description (steps to reproduce);
      - Priority;
      - Severity;
      - Project ID / milestone / release.

Q&A

Homework

- **Chapters 7, 12**

Thank you!

and Happy Testing!