



FC-MD

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Info-Slide

Subtitle:
Speaker: Forrest Shull
Reviewer:
Notes:
Filename:
Version: 1.0

Slide 1



FC-MD

Software Inspections

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

What is an Inspection?

Definition **Inspection is a static analysis method to verify quality properties of software products.**

Characteristics:

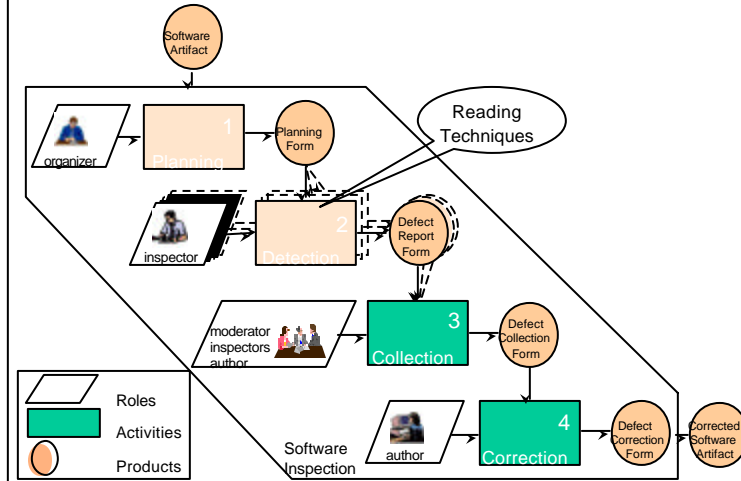
- Structured, well-defined process.
- Inspection team usually consists of technical personnel.
- Participants have well-defined roles.
- Inspection results are documented.

Slide 2

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

How to conduct an Inspection?

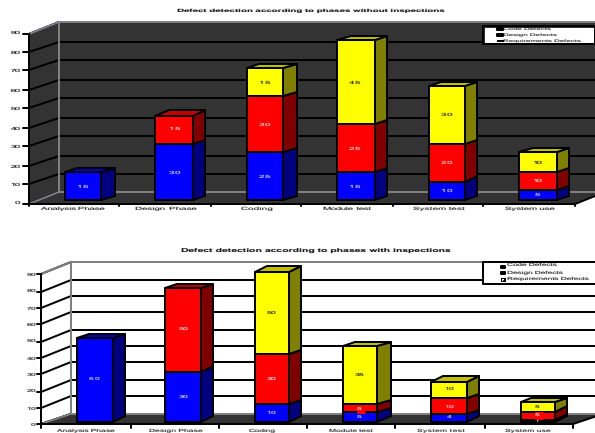


Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Benefits of Inspections: Early defect detection

Inspections help to detect more defects early in the development process
 ® Early Quality Improvement





FC-MD

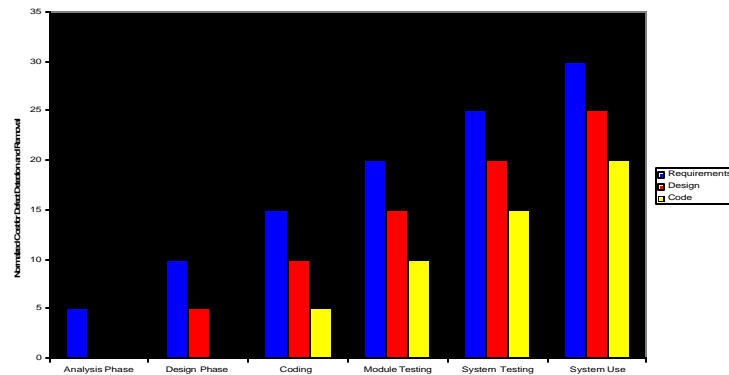
Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Benefits of Inspections: Productivity and Cost

Inspections improve productivity by finding defects when they are cheaper to correct

® Inspections increase productivity and lower cost



Slide 5



FC-MD

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Benefits of Inspections: Useful spin-off for participants

- Learning by experience
- Participants learn about rationales and patterns in defect discovery
 - they avoid these defects in their own work.
- Participants learn about good development patterns
 - they use them in their own work.



Inspections help integrate defect prevention processes into defect detection processes!

Slide 6



Contents

- ▶ 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Other Benefits of Inspections

In the long run...

- inspections convince participants to develop more understandable products
 - ➔ Products are easier to maintain.
- defect data allow
 - ➔ the improvement of the software development process in a project (root cause analysis).
 - ➔ the development of description, prediction, and evaluation models.
- inspections can be used for certification of product quality.



Contents

- 1 Software Inspections
- ▶ 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Reading Technique for Inspections

Question: How does one detect defects?

- Answer:**
- 1 By reading the document.
 - 2 By understanding what the document describes.
 - 3 By checking the required quality properties.

Problem ▶

We often do not know how to read the document!



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Reading Technique for Inspections

Problem

We often do not know how to read the document!

Reason

- We learn how to write requirements, design, code, etc. But, we do not learn how to read them.
- Little technical support for the reading process is currently available.

Solution

- Provide well-defined and tailored reading techniques.

Benefit

- Increase the cost-effectiveness of inspections
- Provide models for writing documents of higher quality
- Reduce human influence on inspection results (i.e., ensure a more engineering approach in inspections)



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Reading Technique for Inspections

What is it?

A reading technique is a series of steps taken to become acquainted with a document. This allows a person to detect defects in a software product throughout the individual defect detection phase of an inspection.

What reading techniques for defect detection are available?

At the moment, four reading techniques for defect detection in requirements are available:

- Ad-hoc
- Checklists
- Defect-based Reading
- Perspective-based Reading



FC-MD

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

Summary

Inspection	is a method to improve software quality with a corollary budget and time benefit
Reading Techniques:	Impact the cost-effectiveness of inspection. Reduces human variability and fallibility
Results of an inspection are:	Artifact of improved quality Documented data on → how to improve the quality of the product → how to improve the quality of the inspection and development process → how to learn locally and globally

Slide 11



FC-MD

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

PBR Concepts: Operational scenarios

- **What is it?** Perspective-Based Reading (PBR) is a set of reading techniques focusing on points of view.
 - The idea is that a document is correct if any potential stakeholder does not find a defect in it. Thus, a document should be read from various point of views (perspectives)
- **Motivation:**
 - make sure important perspectives are identified
 - make each reviewer responsible for particular perspectives
 - allow reviewers to build up expertise in different aspects of the document
- **History:**
 - Piloted at NASA/Goddard Space Flight Center
 - Ongoing studies with students/professionals through universities: UMCP, UMBC, Uni-KI, NTU, Uni-Lund, ...
 - Adapted for industrial use/training at Allianz Life Insurance, Bosch Telecom, Robert Bosch GmbH

Slide 12



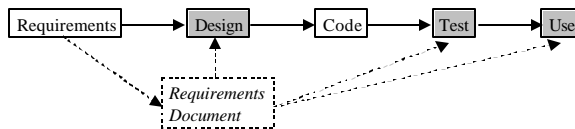
FC-MD

PBR Concepts: Perspectives

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- An example of *perspectives*: Each user of a requirements document has his or her own needs for it:
 - a designer, who uses it to produce a system design;
 - a tester, who produces a test plan to ensure that the system meets the requirements;
 - a user, who has to make sure the requirements adequately capture the functionality needed in the final system;
 - ...



Slide 13



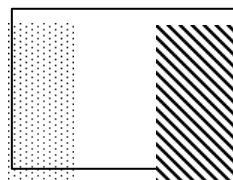
FC-MD

PBR Concepts: Team review

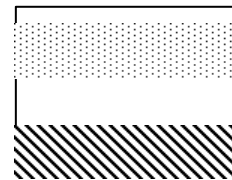
Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- PBR assigns different perspectives to different inspectors, i.e., each inspector of an inspection team reads a document from a particular perspective.
- Benefits:
 - aim to focus reviewer responsibilities (avoid overlap)
 - aim for defect coverage



Ad hoc coverage



PBR coverage

Slide 14



FC-MD

PBR Concepts: Operational Scenarios

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- The ideas behind PBR are implemented in an **operational scenario** for each perspective.
 - a procedural description of activities and questions that guides inspectors through the inspected documented
 - represented as a set of steps to be followed

Slide 15



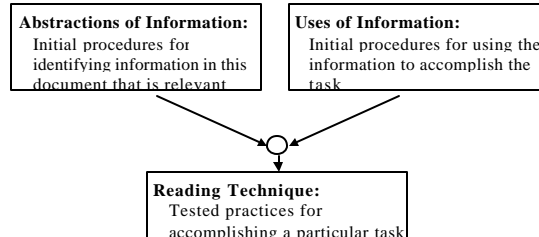
FC-MD

PBR Concepts: Operational Scenarios

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Every Perspective-Based Reading technique consists of two parts:**
- Construct a Model: Create an abstraction of the document under review in order to increase understanding. Based on what perspectives of the document are useful.
- Answer questions about the model: Focused on defect classes, typical problems of the organization, typical features of the document.



Slide 16



FC-MD

PBR Concepts: Operational Scenarios

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Goal for each participant:
 - **Analyze** the documents provided
 - **for the purpose of** evaluation
 - **with respect to** defect detection
 - **from the point of view of** tester (or designer, user, ...)
 - **in the context of** a specific technique for creating some abstraction of the system
- Results:
 - defect report forms
 - a high-level model of the system

Slide 17



FC-MD

PBR Concepts

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- ▶ 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Summary so far: PBR is built on a number of concepts (perspectives, team review, procedural guidelines, defect taxonomy)
- Each of these concepts can be altered to conform to a particular organization:
 - able to add new perspectives of interest to the organization
 - able to change the mix of perspectives
 - if I expect a predominance of defects of a certain type, add more reviewers using relevant perspectives
 - able to change the level of detail incorporated into the scenario
 - able to change the underlying defect taxonomy

Slide 18



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Interpretation of “defect”
 - “defect”: any quality property that is not fulfilled
 - avoid focusing on correctness as the one and only quality property

Defect Classification	Description
<i>Omission</i>	Necessary information about the system has been omitted from the software artifact.
<i>Incorrect Fact</i>	Some information in the software artifact contradicts information in the requirements document or the general domain knowledge.
<i>Inconsistency</i>	Information within one part of the software artifact is inconsistent with other information in the software artifact.
<i>Ambiguity</i>	Information within the software artifact is ambiguous, i.e. any of a number of interpretations may be derived that should not be the prerogative of the developer doing the implementation.
<i>Extraneous Information</i>	Information is provided that is not needed or used.

Slide 19



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Example requirements: ABC video store system
 - **Overview:** “...Customers select at least one video for rental. The maximum number of tapes that a customer can have out on rental at one time is 20. The customer’s account number is entered to retrieve customer data and create an order. Bar code IDs for each tape are entered and video information from inventory is displayed. The video inventory file is updated. When all tape IDs are entered, the system computes the total bill. Money is collected and the amount is entered into the system. Change is computed and displayed. The rental transaction is created, printed and stored. The customer signs the rental form, takes the tape(s) and leaves. . . “
 - **User Characteristics:** “The system will be used by ABC Video management, clerks, and indirectly by customers. From the point of view of the system, clerks and managers are identical. Some system operations are only accessible to managers (such as printing daily reports) and are protected by password. . . “

Slide 20



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Functional Requirement 5

- *Description*
The account number of a customer is entered with the keyboard by the clerk to retrieve the rental transaction record.
- *Input*
The account number is entered with the keyboard.
- *Processing*
Searching for rental transaction record.
- *Output*
Display rental transaction record.

Slide 21



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Defects of Omission

- In the example, the FR gives no information on what a valid or invalid account number looks like.
 - If this info is omitted from the entire document => defect!
- Often requires some judgement call as to whether this info should be the prerogative of the implementor.
- PBR addresses typical difficulties in finding omissions:
 - by helping to identify when specific information, necessary to make the requirements testable, has been left out

Slide 22



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Functional Requirement 7**
 - *Description*
The maximum number of tapes that can be rented at one transaction is 20.
 - *Input*
Bar code ID of tape is entered with the bar code reader
 - *Processing*
If this tape is the 21st taken, rental is rejected
 - *Output*
Error message is displayed

Slide 23



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Defects of Incorrect Fact
 - In the example, there is a defect because the requirement is testing the wrong value (according to the general description).
 - PBR addresses typical difficulties in finding incorrect facts:
 - when the reader goes through the steps of a more detailed scenario, questions about the domain can be focused on key points

Slide 24



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Functional Requirement 3**

- *Description*: The system keeps a rental transaction record for each customer containing contact (name and address) information and currently rented tapes for each customer.

- **Functional Requirement 15**

- *Description*
A new customer wants to rent tapes. The clerk enters all the necessary information, prints the bar code for the ABC card and glues it on a blank ABC card. Then this card is given to the customer.
- *Input*
Clerk enters the following information: Name, credit card information, and address of the customer.
- *Processing*
Crate a new rental record for the customer, containing the input information. The system assigns an account number to the customer and prints the bar code. Go to initial state.
- *Output*
Printing of the bar code. Customer can rent tapes.

Slide 25



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Defects of Inconsistency**

- In the example, the two FRs are inconsistent, and there is nothing in my domain knowledge or the other FRs to tell me which is correct.
- PBR addresses typical difficulties in finding inconsistencies:
 - It is still difficult to remember all previous requirements that might contradict the current one. However, PBR helps focus the search by asking specific questions during the process.

Slide 26



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Functional Requirement 10**

- *Description*
When the clerk presses an “order-complete” option key (defined by the system) this rental is complete and the video inventory file is updated.
- *Input*
Clerk presses the “order-complete” option key.
- *Processing*
Update the video inventory file. Close rental transaction.
- *Output*
Video inventory file is updated. Rental transaction file is updated.



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Defects of Ambiguity**

- In the example, an ambiguous description could result in a number of different implementations. “Updating the file” can seem like a reasonable requirement, but how exactly is the file to be changed?
- PBR addresses typical difficulties in finding ambiguities:
 - when readers are required to make things testable, they have to make sure that a specific behavior is specified for every input (i.e. that each requirement is clear, specific, and has only one interpretation)



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Functional Requirement 11**

- *Description*
After the rental is closed the transaction is printed.
- *Input*
Close the current rental.
- *Processing*
Print form that the customer has to sign. Return to initial state.
Forms will be kept on file in the store for one month after the tapes are returned.
- *Output*
Printed form. Initial menu is displayed.

Slide 29



FC-MD

Requirements Defects

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Defects of Extraneous Information**

- In the example, the requirement “forms are kept on file for one month” is outside the scope of the system. How could this be tested for?
- PBR addresses typical difficulties in finding extraneous information:
 - by forcing readers to think about making system behavior testable, it becomes easier to recognize functionality outside the system

Slide 30



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- ▶ 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- The tester needs to find out if the system does what it is supposed to
 - “What it is supposed to do” is the information contained in the requirements document
- One testing strategy: black-box testing
 - Tries to answer: Does a module perform the functionality it was intended to perform, as stated in the requirements?
 - feed the system with data
 - observe the reaction
 - observed reaction different from expected => defect!

Slide 31



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Equivalence partitioning
 - a type of black-box testing
 - Examines the set of all possible inputs to determine which data items are similar to one another on some relevant dimension.
 - Sets of similar data items are called equivalence sets
 - Helps manage the number of test cases necessary, since need at least one test case from each equivalence set.
 - (Within an equivalence set, one item is as good as any other - if output is correct for one, should be correct for all.)
 - Often used with boundary value testing (experience indicates faults are more likely for items near boundaries of equivalence sets).

Slide 32



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Equivalence partitioning: Example 1
 - Module keeps a running total of amount of money received
 - Input: AmountofContribution, specified as being a value from \$0.01 to \$99,999.99
 - 3 equivalence classes:
 - Values from \$0.01 to \$99,999.99 (valid)
 - Values less than \$0.01 (invalid)
 - Values greater than \$99,999.99 (invalid)
- Equivalence partitioning: Example 2
 - Module prints an appropriate response letter
 - Input: MemberStatus, specified as having a value of {Regular, Student, Retiree, StudioClub}
 - 2 equivalence classes:
 - Values of {Regular, Student, Retiree, StudioClub} (valid)
 - All other input (invalid)

Slide 33



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Equivalence partitioning: Example 3
 - Input: user enters Size, a whole number between 1 and 48
 - *Multiple* equivalence classes:
 - Values less than 1 (invalid)
 - Values in the range 1 to 48 (valid)
 - Values greater than 48 (invalid)
 - Whole number values (valid)
 - Fractional values (invalid)
 - Numeric characters (valid)
 - Nonnumeric characters (invalid)

Slide 34



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Reviewing using equivalence partition testing:**
- **Step 1**
 - **General instructions:** For each requirement, identify the inputs.
 - **Specific instructions:** (Guidelines for identifying inputs)
 - **Questions:**
 - Does the requirement make sense from what you know about the application or from what is specified in the general description?
 - Do you have all the information necessary to identify the inputs to the requirement? Based on the general requirements and your domain knowledge, are these inputs correct for this requirement?
 - Have any of the necessary inputs been omitted?
 - Are any inputs specified which are not needed for this requirement?
 - Is this requirement in the appropriate section of the document?

Slide 35



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Step 2**
 - **General instructions:** For each input, construct equivalence sets.
 - **Specific instructions:** (Guidelines for constructing the equivalence sets for an input class)
 - **Questions:**
 - Do you have enough information to construct the equivalence sets for each input? Can you specify the boundaries of the sets at an appropriate level of detail?
 - According to the information in the requirements, are the sets constructed so that no value appears in more than one set?
 - Do the requirements state that a particular value should appear in more than one equivalence set (that is, do they specify more than one type of response for the same value)? Do the requirements specify that a value should appear in the wrong equivalence set?

Slide 36



FC-MD

The "tester" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- ▶ 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Step 3

- **General instructions:** For each equivalence set, create test cases and specify the expected output.
 - **Specific instructions:** (Guidelines for constructing specific test cases based on system requirements)
- **Questions:**
 - Do you have enough information to create test cases for each equivalence set?
 - Are there other interpretations of this requirement that the implementor might make based upon the description given? Will this affect the tests you generate?
 - Is there another requirement for which you would generate a similar test case but would get a contradictory result?
 - Can you be sure that the tests generated will yield the correct values in the correct units? Is the resulting behavior specified appropriately?

Slide 37



FC-MD

A tester scenario example

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- ▶ 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Functional Requirement 8

- *Description*
When all tapes are entered the system computes the total amount due
- *Input*
“Enter” key is pressed on keyboard after the last tape was entered.
- *Processing*
Computation of the total due. The total is the sum of the past-due fees, other fees and current video fees.
- *Output*
Total due.

Slide 38

Form A - Equivalence Sets and Test Cases for New Requirements

Reviewer Name:

Document Reviewed:

Requirement Number: 8	Page:		
Description of Input: user input			
Valid			
Equiv. sets:	"enter"		
Test cases:	"enter"		
Test result:	Total due = past_due_fees + other_fees + current_fees		
Invalid			
Equiv. sets:	any other key		
Test cases:	1, a, F1		
Test result:	no action)		
Description of Input:			
Valid			
Equiv. sets:			
Test cases:			
Test result:			
Invalid			
Equiv. sets:			
Test cases:			
Test result:			



FC-MD

A tester scenario example

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Functional Requirement 5

- *Description*
The account number of a customer is entered with the keyboard by the clerk to retrieve the rental transaction record.
- *Input*
The account number is entered with the keyboard.
- *Processing*
Searching for rental transaction record.
- *Output*
Display rental transaction record.

Form A - Equivalence Sets and Test Cases for New Requirements

Reviewer Name:

Document Reviewed:

Requirement Number: 5		Page:	
Description of Input: account number			
Valid	valid		
Equiv. sets:	acct. numbers		
Test cases:	can't determine)		
Test result:			
Invalid	invalid (illegal, unassigned) acct. numbers		
Equiv. sets:	can't determine)		
Test cases:			
Test result:			
Description of Input:			
Valid			
Equiv. sets:			
Test cases:			
Test result:			
Invalid			
Equiv. sets:			
Test cases:			
Test result:			

"Do you have enough information to construct the equivalence sets for each input? Can you specify the boundaries of the sets at an appropriate level of detail?"

Notice that the requirement does not give any information on what a valid or invalid account number looks like. If this information has been omitted from the entire document, this is something that should be noticed as soon as we try to construct the test cases. Whether or not this is an actual fault depends on a judgement call as to whether or not this information should be the prerogative of the implementor.



FC-MD

A tester scenario example

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Reporting the defect:

Defect #	Page #	Req #	Description
1	8	FR5	A definition of the "account number", at a useful level of detail, has been omitted.



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- ▶ 7 The „user“ scenario
- 8 A user scenario example

- User needs to make sure requirements adequately capture the functionality needed in the system
- One strategy for handling functionality is **use cases**:
 - A use case describes a particular set of system functionality by modeling the “dialogue” a user undertakes with the system.
 - A use case is based on a **descriptive scenario** of how the user interacts with the system. It identifies events that can occur and describes the system response.
 - A use case is a complete and meaningful flow of events, taken from the perspective of a particular user of the system.
 - Taken together, all use cases constitute all possible ways of using the system.

Slide 43



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- ▶ 7 The „user“ scenario
- 8 A user scenario example

- Use cases provide a view of the system that is very focused on **functionality**.
 - They provide a very understandable format for **communication**, especially between system designers and the customers of a system. They can help validate whether the system being described will work the way the customers expect it will.
 - They feed into **later stages of the lifecycle** by helping to identify objects, develop test plans, and develop documentation.
 - They are widely accepted by the OO community as a way to **supplement object modeling** during system analysis.

Slide 44



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Use Cases May Help in Finding Requirements Defects**
- Use cases provide a different representation of the functionality described in English-language requirements.
- Assumptions for defect detection:
 - If the English-language requirements are well-specified, we should be able to convert them into use cases relatively easily.
 - If there are hidden problems with the English-language requirements, they may prevent us from creating a clear, complete use case translation. These problems would probably also cause difficulties in constructing other artifacts based on the requirements, such as a design or code.
- The process of constructing use cases can be used as an aid in identifying defects.

Slide 45

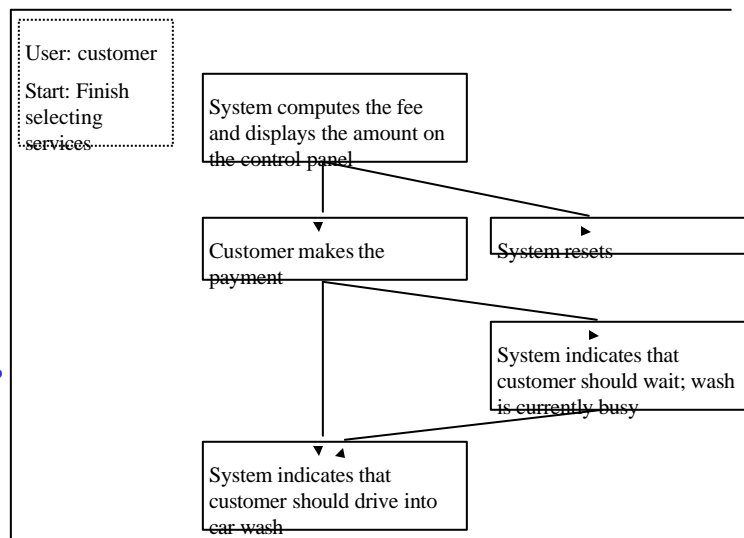


FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example



Slide 46



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- First major component: participants
 - **entities** in the environment (i.e. **external** to the system)
 - that **interact with the system** to achieve some functionality.
 - (often known as *actors*).
- Participants can be: human users, other systems, external organizations, external devices, etc., which interact with the system.
- Some participants initiate events; others interact with the system as a result of other events.
- When users are involved: a participant is a role (i.e. a characteristic way of interacting with the system) NOT a specific person.

Slide 47



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Which user groups use the system to perform a task?
- Which user groups are needed by the system in order to perform its functions?
- Which are the external systems that use the system in order to perform a task?
- Which are the external systems that are managed or otherwise used by the system in order to perform a task?
- Which are the external systems or user groups that send information to the system?
- Which are the external systems or user groups that receive information from the system?

Slide 48



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- Second major component: Functionality
- More than anything else, use cases are concerned with describing system behavior.
- Problem: Very hard to specify an exact definition for the intuitive way in which we usually describe system behavior.
- Lot of variation in how formally use case concepts are applied to describing a system: very loosely in some situations, very rigorously in others.
- We ask for functionality to be described at 2 levels:
 - product functions (What features does the product have? What are the specific activities it can do?)
 - use cases/user scenarios (What can customers use the system for?)

Slide 49



FC-MD

The "user" scenario

Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

- **Step 1**
 - **General instructions:** Identify the participants in the requirements.
 - **Specific instructions:** (guidelines for recognizing participants)
 - **Questions:**
 - Q1.1 Are multiple terms used to describe the same participant in the requirements?
 - Q1.2 Is the description of how the system interacts with a participant inconsistent with the description of the participant? Are the requirements unclear or inconsistent about this interaction?
 - Q1.3 Have necessary participants been omitted? That is, does the system need to interact with another system, a piece of hardware, or a type of user that is not described?
 - Q1.4 Is an external system or a class of "users" described in the requirements which does not actually interact with the system?

Slide 50



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Step 2.

- **General goal:** Describe system functionality.
- **Specific goal:** (Procedure for identifying specific system product functions and understanding how they contribute to use cases)
- **Questions:**
 - Q2.1 Are the start conditions for each use case specified at an appropriate level of detail?
 - Q2.2 Are the class(es) of users who use the functionality described, and are these classes correct and appropriate?
 - Q2.3 Is there any system functionality that should be included in a use case but is described in insufficient detail or omitted from the requirements?
 - Q2.4 Has the system been described sufficiently so that you understand what activities are required for the user to achieve the goal of a use case? Does the description allow more than one interpretation as to how the system achieves this goal?
 - Q2.5 Do the requirements omit use cases that you feel are necessary, according to your domain knowledge or the general description?

Slide 51



Contents

- 1 Software Inspections
- 2 Software Reading Techniques
- 3 PBR Concepts
- 4 Requirements Defects
- 5 The „tester“ scenario
- 6 A tester scenario example
- 7 The „user“ scenario
- 8 A user scenario example

• Step 3.

- **General goal:** Cross-index participants and functionality.
- **Specific goal:** (Procedure for helping identify which participants are involved in which system functionalities)
- **Questions:**
 - Q3.1 Is it clear from the requirements which participants are involved in which use cases?
 - Q3.2 Based on the general requirements and your knowledge of the domain, has each participant been connected to all of the relevant use cases?
 - Q3.3 Are participants involved in use cases that are incompatible with the participant's description?
 - Q3.4 Have necessary participants been omitted (e.g., are there use cases which require information that cannot be obtained from any source described in the requirements)?

Slide 52