

# Software Testing

Lesson 6 – Dynamic Testing II

Uwe Gühl  
Winter 2015 / 2016





# Contents

- Test Design Techniques – Dynamic Testing II
  - Black-box Techniques  
(or Specification-based Techniques)
    - Equivalence Partitioning
    - Boundary Value Analysis
    - Decision Table Testing
    - State Transition Testing
    - Use Case Testing
  - Comparison White-box / Black-box Techniques
  - Experience-based Techniques
  - Choosing Test Techniques



# Black-box Techniques

## Equivalence Partitioning (1/4)

- Inputs to the software or system are divided into groups that are expected to exhibit similar behaviour.
- Equivalence partitions (or classes) can be found for
  - valid data, i.e., values that should be accepted
  - invalid data, i.e., values that should be rejected
- Partitions can also be identified for
  - outputs
  - internal values
  - time-related values (e.g., before or after an event)
  - interface parameters (e.g., integrated components being tested during integration testing)



# Black-box Techniques

## Equivalence Partitioning (2/4)

- Equivalence partitioning
  - is applicable at all levels of testing.
  - can be used to achieve input and output coverage goals.
  - can be applied to
    - human input
    - input via interfaces to a system
    - interface parameters in integration testing



# Black-box Techniques

## Equivalence Partitioning (3/4)

- Example: Input: Day of the week

### Ideas

- Working day
- Weekend day

- Input: Day of the week as number  
(1 = Monday, ..., 7 = Sunday)

### Ideas:

- Valid:  $\geq 1$  and  $\leq 7$
- Not valid:  $<1$  or  $>7$
- Test: 0, 5, 14

Legend:

**Red** values: Not valid values

**Green** values: Valid values



# Black-box Techniques

## Equivalence Partitioning (4/4)

- Example: Input: Month; ideas:
  - Months with 30 days
  - Months with 31 days
  - February with 28 days (non leap year)
  - February with 29 days (leap year)
- Month as number  
(1 = January, ..., 12 = December); ideas:
  - Valid:  $\geq 1$  and  $\leq 12$
  - Not valid:  $< 1$  or  $> 12$
  - Test: **0**, **5**, **14**

Legend:  
**Red** values: Not valid values  
**Green** values: Valid values



# Black-box Techniques

## Boundary Value Analysis (1/3)

- Behaviour at the edge of each equivalence partition is more likely to be incorrect than behaviour within the partition  
⇒ Defect detection probable
- The maximum and minimum values of a partition are its boundary values
- Boundary of a valid partition is a valid boundary value
- Boundary of an invalid partition is an invalid boundary value

Design corresponding **tests** for each boundary value



# Black-box Techniques

## Boundary Value Analysis (2/3)

- Boundary value analysis
  - can be applied at all test levels
  - is relatively easy to apply and its defect finding capability is high
  - Is often considered as an extension of equivalence partitioning or other black-box test design techniques
- Detailed specifications are helpful in determining the interesting boundaries



# Black-box Techniques

## Boundary Value Analysis (3/3)

- Examples for usage:
  - Equivalence classes for user input on screen
  - time ranges  
(e.g., time out, transactional speed requirements)
  - table ranges (e.g., table size is 512\*512).
- Example: Month as number  
(1 = January, ..., 12 = December); ideas:
  - Valid:  $\geq 1$  and  $\leq 12$
  - Not valid:  $< 1$  or  $> 12$
  - Boundaries Test: 0, 1, 12, 13

Legend:  
**Red** values: Not valid values  
**Green** values: Valid values



# Black-box Techniques

## Decision Table Testing (1/6)

- Decision tables
  - to capture system requirements that contain logical conditions
  - to document internal system design
  - to record complex business rules that should be implemented
- Proceeding
  - Analysis of specification
  - Identification of conditions and actions of the system



# Black-box Techniques

## Decision Table Testing (2/6)

- A decision table contains
  - the triggering conditions, often combinations of true and false for all input conditions
  - resulting actions for each combination of conditions

	Visiting rules	1	2	3	4	5	6
Conditions	Infectious sickness	yes	yes	yes	yes	no	no
	Visit during visiting time	yes	yes	no	no	yes	yes
	Fever	yes	no	yes	no	yes	no
Actions	No visit	x	x	x	x		
	Visit with nurse						
	Visit max. 30 minutes						
	Regular visit						

Input conditions, often BOOLEAN

capture system requirements that contain logical conditions

Idea: One test per column

Each column is related to a business rule that defines a unique combination of conditions



# Black-box Techniques

## Decision Table Testing (3/6)

- Example [Dus07]:  
Visiting rules in an hospital.  
8 Test Cases in the beginning

	Visiting rules	1	2	3	4	5	6	7	8
Conditions	Infectious sickness	yes	yes	yes	yes	no	no	no	no
	Visit during visiting time	yes	yes	no	no	yes	yes	no	no
	Fever	yes	no	yes	no	yes	no	yes	no
Actions	No visit	x	x	x	x				
	Visit with nurse					x		x	x
	Visit max. 30 minutes							x	
	Regular visit						x		



# Black-box Techniques

## Decision Table Testing (4/6)

- Example [Dus07]: Same impact

	Visiting rules	1	2	3	4	5	6	7	8
Conditions	Infectious sickness	yes	yes	yes	yes	no	no	no	no
	Visit during visiting time	yes	yes	no	no	yes	yes	no	no
	Fever	yes	no	yes	no	yes	no	yes	no
Actions	No visit	x	x	x	x				
	Visit with nurse					x		x	x
	Visit max. 30 minutes							x	
	Regular visit						x		



# Black-box Techniques

## Decision Table Testing (5/6)

- Example [Dus07]: Reducing to 5 Test Cases

	Visiting rules	1	5	6	7	8
Conditions	Infectious sickness	yes	no	no	no	no
	Visit during visiting time	-	yes	yes	no	no
	Fever	-	yes	no	yes	no
Actions	No visit	x				
	Visit with nurse		x		x	x
	Visit max. 30 minutes				x	
	Regular visit			x		



# Black-box Techniques

## Decision Table Testing (6/6)

- Advantage
  - It creates combinations of conditions that otherwise might not have been exercised during testing
- When to use?
  - When the action of the software depends on several logical decisions



# Black-box Techniques

## State Transition Testing (1/6)

- A system may exhibit a different response depending on current conditions or previous history (its state).
  - ⇒ can be shown with a state transition diagram.
- It allows to view the software in terms of
  - its states
  - transitions between states
  - the inputs or events that trigger state changes (transitions)
  - the actions which may result from those transitions



# Black-box Techniques

## State Transition Testing (2/6)

- The states of the system or object under test are separate, identifiable and finite in number
- A state table
  - shows the relationship between the states and inputs
  - can highlight possible transitions that are invalid



# Black-box Techniques

## State Transition Testing (3/6)

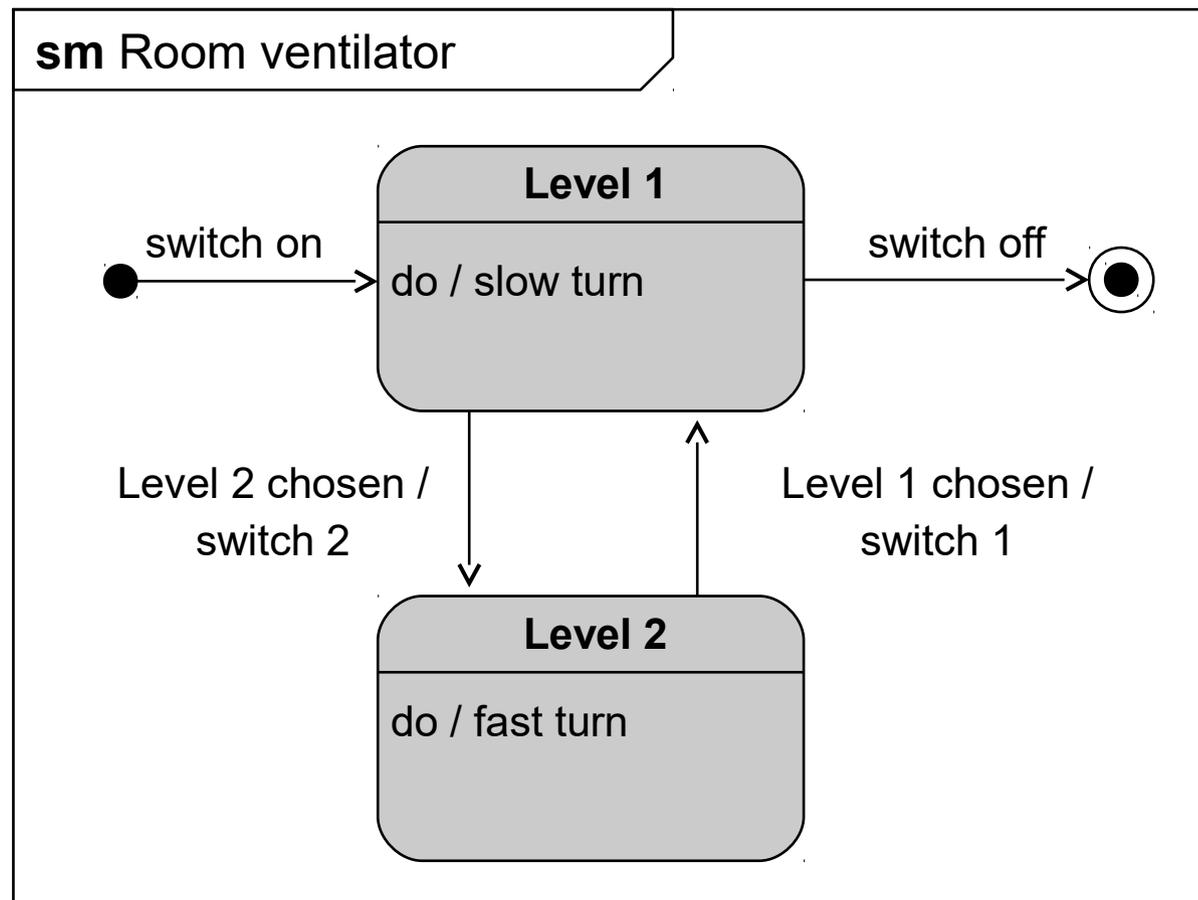
- Tests can be designed to
  - cover a typical sequence of states
  - cover every state
  - exercise every transition
  - exercise specific sequences of transitions
  - test invalid transitions
- Usage of State transition testing
  - embedded software
  - technical automation
  - modelling a business object with specific states (in business scenarios)
  - testing screen-dialogue flows (Web applications)



# Black-box Techniques

## State Transition Testing (4/6)

- Example 1: Fan

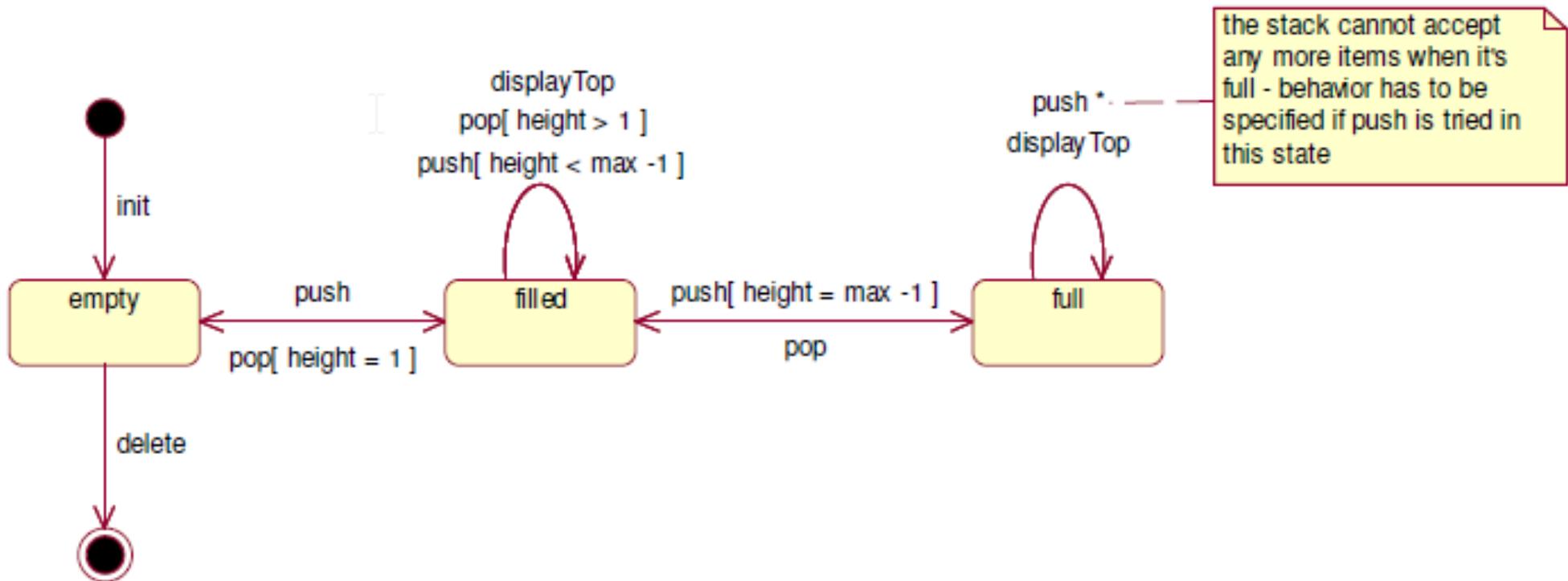




# Black-box Techniques

## State Transition Testing (5/6)

- Example 2: Vendor machine [Dus07]
- Status: „empty“, „filled“ (less than full), „full“

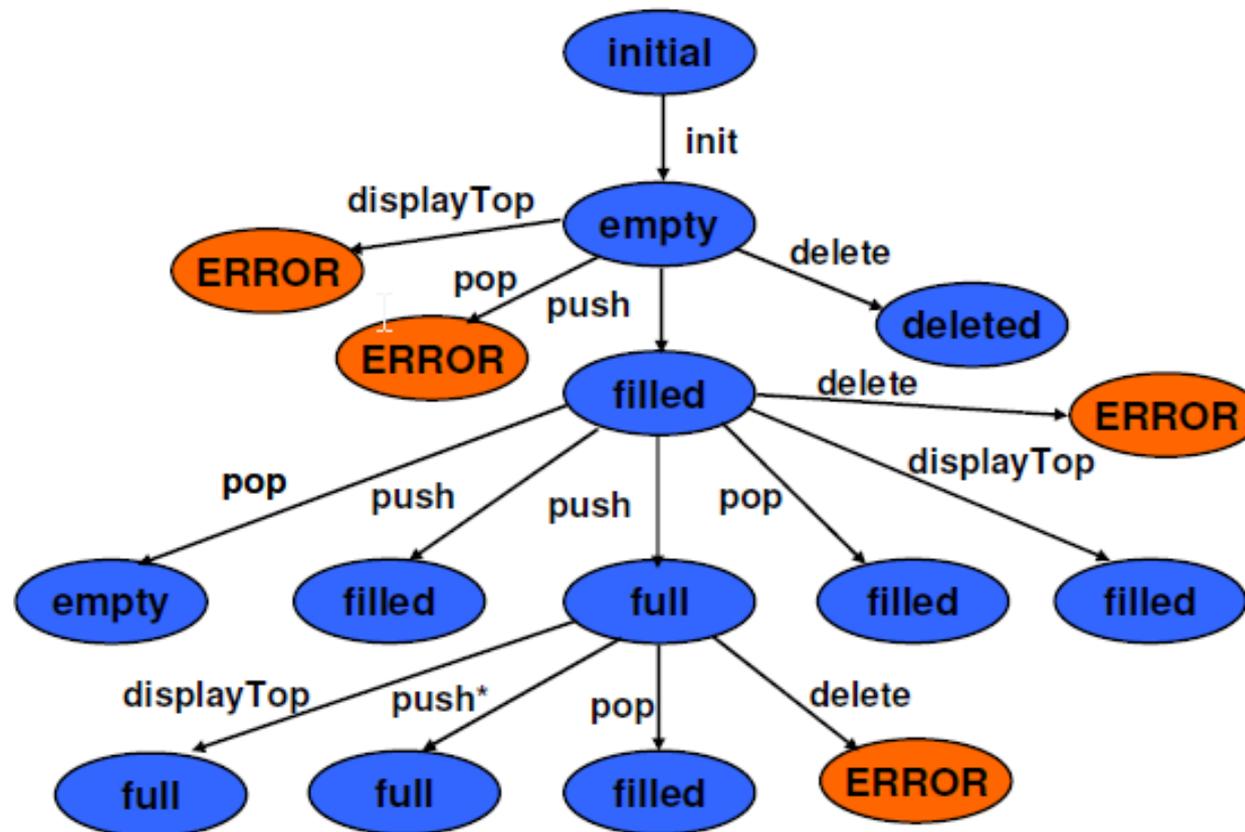




# Black-box Techniques

## State Transition Testing (6/6)

- Example 2: Transition Tree [Dus07]





# Black-box Techniques

## Use Case Testing (1/16)

- Tests can be derived from use cases
- A use case describes interactions between actors (users or systems), which produce a result of value to a system user or the customer
- Use cases may be described at the
  - abstract level  
Business use case, technology-free, business process level
  - system level  
System use case on the system functionality level



# Black-box Techniques

## Use Case Testing (2/16)

- Use Case has
  - a main scenario *Positive*
  - alternative scenarios *Variants*
  - failure scenarios *Negative*
  - preconditions  
which need to be met for the use case to work successfully
  - postconditions  
which are the observable results and final state of the system after the use case has been completed

# Black-box Techniques

## Use Case Testing (3/16)



### Example of a Use case description

Id / Name	214 / Rent a car
Short description	A customer comes to the car rental agency and chooses a car which he rents for a fixed period
Actors	Customer, agent
Trigger	Customer asks agent
Pre condition	The rental system is ready to get customer data and to realize a lease contract
Result	Leasing is done, and the customer has signed the contract
Post condition	The rental system is ready to get customer data and to realize a lease contract
Activities	<ol style="list-style-type: none"><li>1. Enter customer data. If customer is yet not registered <math>\Rightarrow</math> UC 12 <i>Register customer.</i></li><li>2. Enter desired car category</li><li>3. Enter desired leasing period</li><li>4. If a car is available in the desired period:<ol style="list-style-type: none"><li>1. Reserve a car</li><li>2. Enter credit card information</li><li>3. Print contract and sign</li></ol>Otherwise: Adapt item 2. or 3., if possible</li></ol>



# Black-box Techniques

## Use Case Testing (4/16)

- Use cases
  - describe the “process flows” through a system based on its actual likely use
  - are very useful for designing acceptance tests with customer/user participation
- Test cases derived from use cases detect
  - defects in the process flows during real-world use of the system
  - integration defects caused by the interaction and interference of different components



# Black-box Techniques

## Use Case Testing (5/16)

- Test Case
  - Sequence of steps consisting of actions to be performed on the system under test (SUT) [Bla04]
  - “Basic unit” in testing
  - serves to validate the functionality and to confirm the realization of a requirement
    - functional
    - non functional (quality criteria)
  - originates typically out of an use case



# Black-box Techniques

## Use Case Testing (6/16)

- Test Case
  - describes the role who should execute it
  - contents test steps with
    - activities of the tester
    - input values
    - expected output values
  - describes
    - preconditions
    - postconditions
- Non functional requirements (NFR) Test Cases are usually derived from functional Test Cases



# Black-box Techniques

## Use Case Testing (7/16)

- Test Case – Example

- Test Case name „IU22\_Create-Object“
- Test Case ID 7
- Priority 1
- Test classification Standard
- Preparation Hours 1
- Execution Hours 1
- Description Creation of an object. The user must select an object He has to decide which specific kind ...
- Risk Without creation of objects Software can't be used
- Version 01
- is Test Case Chain [ ]



# Black-box Techniques

## Use Case Testing (8/16)

- Test Case – Example (cont.) Condition

– Goal	Creation of a new object
– Prerequisites	Following objects must be available in database to execute this test case: * object A * object B
– Remarks	Function „select module“ is described in Test Case „IU21_Display-Object“



# Black-box Techniques

## Use Case Testing (9/16)

- Test Case – Example (cont.) Test Steps

– StepNo.	Description	Expected result	Comments
– 10	Select an object in the tree structure	Selected item is highlighted	
– 20	Choose „create“	Dialog box opens	
– 30	Choose radio button		
– 40	Enter Obj ID		



# Black-box Techniques

## Use Case Testing (10/16)

- Following prioritization  
Use Cases are typically prioritized
  - High priority: Must – necessary
  - Medium priority: Should – important
  - Low priority: Could – Nice to have
- Prioritization of use cases should be considered in derived test cases



# Black-box Techniques

## Use Case Testing (11/16)

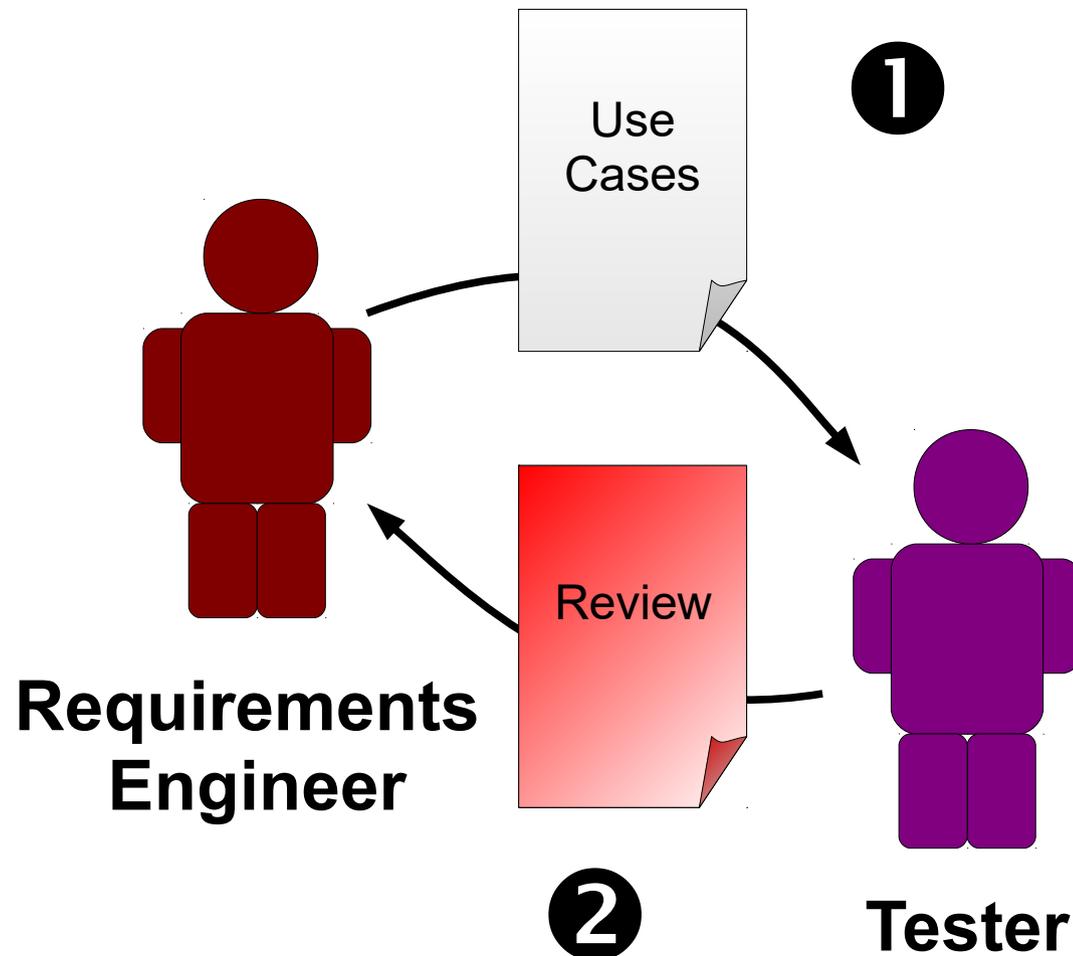
- Effort calculation  
The more complex and the more effort in creating Use Cases, the higher the estimated test effort, e.g.
    - Very complex Use Cases ( $> 1$  week effort)  
→ 12 Test Cases
    - Medium complex Use Cases ( $> 1$  day,  $\leq 1$  week effort)  
→ 8 Test Cases
    - Low complex Use Cases ( $\leq 1$  day effort)  
→ 4 Test Cases
- ⇒ Good basic to calculate effort in designing test cases



# Black-box Techniques

## Use Case Testing (12/16)

- Reviews of Use Cases as basic

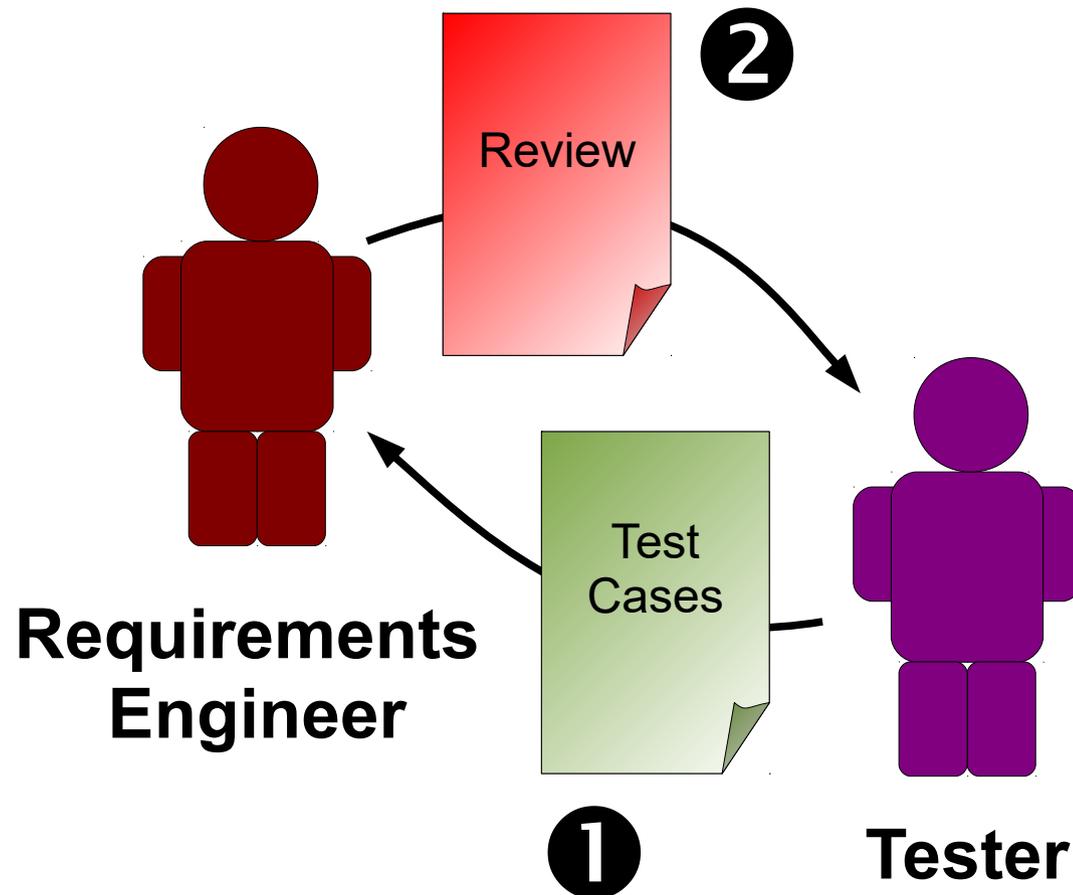




# Black-box Techniques

## Use Case Testing (13/16)

- Reviews of Test Cases to achieve better quality





# Black-box Techniques

## Use Case Testing (14/16)

- Test Scenario
  - Synonym: *Test Case Chain*, *Test Suite* [Bla04]
  - to test processes
    - Are process requirements implemented completely and correct?
  - to test the data flow in the system
  - arises typically from a Business Scenario (Business Use Case)
  - combination of logically related test cases [Bla04]
  - In case: Related test cases could be modified (as a rule simplified)



# Black-box Techniques

## Use Case Testing (15/16)

- Test Scenario Example “User logs in a vocabulary training system and does 1<sup>st</sup> lecture”
  - Test case 17 „First login“
  - Test case 33 „Choose Language“
    - Source language „English“, Target language „Thai“
    - Level „Starter“
    - Lesson „Vacancy“
    - Learning strategy 1
  - Test case 46 „First lesson“
  - Test case 103 „Follow-up lesson“
  - Test case 132 „Score“
    - Choose Bar Chart



# Black-box Techniques

## Use Case Testing (16/16)

- Test data
  - All data needed for testing
  - Discussion
    - Based on Business Object Data Model or Physical Data Model
    - Artificial data or based on real business data, e. g. out of legacy systems
    - Which test data are included with delivery?
    - Feed of test data
    - Remove of test data („naked system“, “database clean-up”)



# Comparison (1/2)

- White Box Testing
  - Based on internal structure, code, database
  - Typically done by developer, designer
  - Test Levels: Component, Component Integration
- Black Box Testing
  - Based on requirements, functionality
  - Typically done by (professional) testers, users
  - Test Levels: System, System Integration, Acceptance



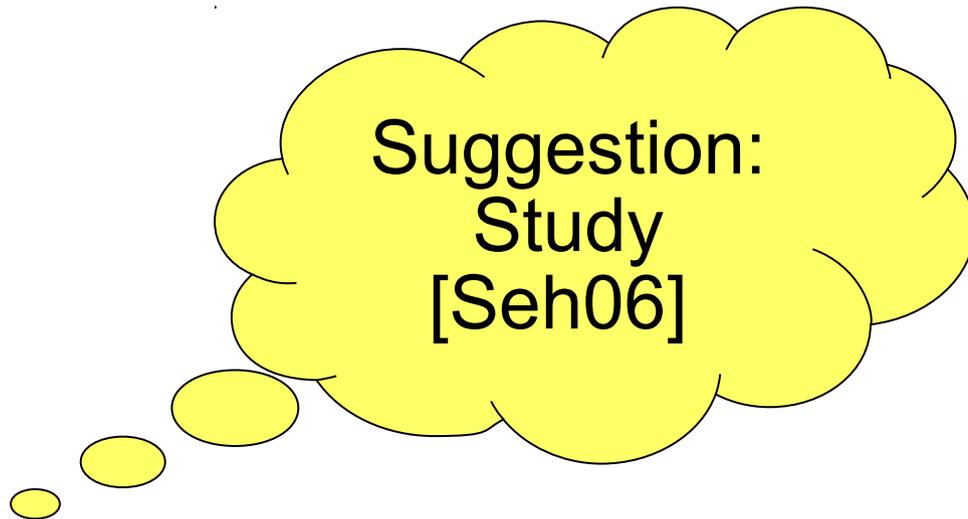
# Comparison (2/2)

- White Box Testing

- Less organizational effort
- Automation easy
- Higher code quality

- Black Box Testing

- Good testing of the complete software
- Review of specification
- Independent from implementation
- Test focus only on specification; means: Less quality of specification  
→ less quality of test results



Suggestion:  
Study  
[Seh06]



# Experience-based Techniques

- Tests are derived from the tester's
  - skill and intuition
  - experience with similar applications and technologies
- Good to find additional special tests, especially when applied after more formal approaches
- Dependency on the tester's experience



# Experience-based Techniques

## Error guessing

- Commonly used experience-based technique
- Testers anticipate defects based on experience
- A structured approach called “fault attack”
  - Enumerate a list of possible defects, based on
    - experience
    - available defect data
    - common knowledge about why software fails
  - Design tests that attack these defects



# Experience-based Techniques

## Exploratory testing

- Concurrent test design, test execution, test logging and learning
- Based on a test charter containing test objectives
- Carried out within time-boxes
- Approach is useful under following conditions:
  - Only few or inadequate specifications available
  - severe time pressure
  - to augment or complete other, more formal testing



# Choosing Test Techniques

- The choice of which test techniques to use depends on a number of factors, including
  - type of system
  - regulatory standards
  - customer or contractual requirements
  - level of risk
  - type of risk
  - test objective
  - documentation available



# Choosing Test Techniques

- The choice of which test techniques to use depends on a number of factors, including (c'td)
  - knowledge of the testers
  - time and budget
  - development life cycle
  - use case models
  - previous experience with types of defects found
- Some techniques are more applicable to certain situations and test levels; others are applicable to all test levels



# Sources

- [Bla04] Rex Black: Critical Testing Processes: Plan, Prepare, Perform, Perfect; Addison-Wesley Professional, 2004
- [Dus07] , Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007
- [ISTQB-CTFLS11] International Software Testing Qualifications Board: Certified Tester Foundation Level Syllabus, Released Version 2011, <http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html>
- [ISTQB-GWP15] Glossary Working Party of International Software Testing Qualifications Board: Standard glossary of terms used in Software Testing, Version 3.01, 2015, <http://www.istqb.org/downloads/glossary.html>
- [Seh06] Scott Sehlhorst: Software Testing Series: Black Box vs White Box Testing, 2006, <http://tynerblain.com/blog/2006/01/13/software-testing-series-black-box-vs-white-box-testing/>
- [Wik16] wikipedia.org: Software testing, 2016, [https://en.wikipedia.org/wiki/Software\\_testing](https://en.wikipedia.org/wiki/Software_testing)