



Software Testing Foundation Level

Lecture 4 – Test Techniques

Uwe Gühl



Contents

- 4.1 Categories of Test Techniques
- 4.2 Black-box Test Techniques
- 4.3 White-box Test Techniques
- 4.4 Experience-based Test Techniques

Contents

- 4.1 Categories of Test Techniques
- 4.2 Black-box Test Techniques
- 4.3 White-box Test Techniques
- 4.4 Experience-based Test Techniques

Categories of Test Techniques

Test technique

(Synonyms: *test case design technique, test specification technique, test design technique***):**

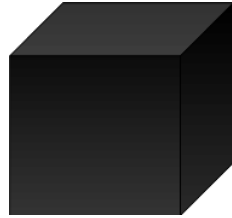
A procedure used to define test conditions, design test cases, and specify test data.

- The international standard ISO/IEC/IEEE 29119-4 contains descriptions of test techniques and their corresponding coverage measures

Categories of Test Techniques

- Which test techniques to use depends on factors like:
 - Component or system complexity
 - Regulatory standards
 - Customer or contractual requirements
 - Risk levels and types
 - Available documentation
 - Tester knowledge and skills
 - Available tools
 - Time and budget
 - Software development lifecycle model
 - Types of defects expected in the component or system

Categories of Test Techniques and Their Characteristics



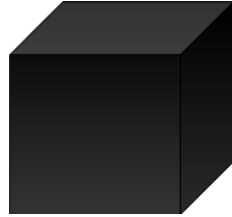
- **Black-box test technique**

(**Synonyms:** *black-box technique, specification-based technique, specification-based test technique*):

A test technique based on an analysis of the specification of a component or system.

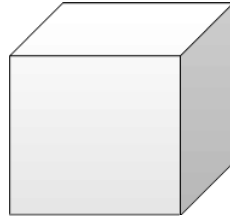
- applicable to both functional and nonfunctional testing.
- focus on the inputs and outputs of the test object without reference to its internal structure.

Categories of Test Techniques and Their Characteristics



- Black-box test technique
 - Test conditions, test cases, and test data are derived from a test basis that may include
 - formal requirements documents like software requirements,
 - specifications,
 - use cases,
 - user stories,
 - business processes.
 - Test cases may be used to detect gaps between the requirements and the implementation of the requirements, as well as deviations from the requirements
 - Coverage is measured based on the items tested in the test basis and the technique applied to the test basis

Categories of Test Techniques and Their Characteristics



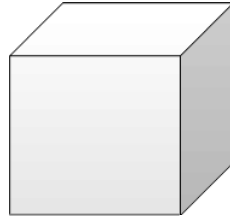
- **White-box test technique**

(Synonyms: *structural test technique, structure-based test technique, structure-based technique, white-box technique***):**

A test technique only based on the internal structure of a component or system.

- focus on the structure and processing within the test object.

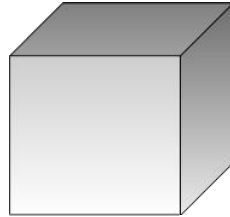
Categories of Test Techniques and Their Characteristics



- White-box test technique
 - Test conditions, test cases, and test data are derived from a test basis that covers
 - software architecture,
 - detailed design,
 - code of the test object,
 - any other source of information regarding the structure of the software
 - Coverage is measured based on the items tested within a selected structure (e.g., the code or interfaces) and the technique applied to the test basis

Categories of Test Techniques and Their Characteristics

Excurses



- Grey-box test techniques
 - Test Strategy based partly on internals of a software, involves knowledge of internal data structures and algorithms for purposes of designing tests
 - execute defined tests at the user, or black-box level.
 - Idea: If you know something about the inside, you can test it better from outside
 - Important with web applications

Categories of Test Techniques and Their Characteristics

- **Experience-based test technique**
(**Synonyms:** *experience-based technique, experience-based test design technique*):
A test technique only based on the tester's experience, knowledge and intuition.
 - Test basis may include knowledge and experience of
 - testers,
 - developers,
 - users,
 - other stakeholders.

Categories of Test Techniques and Their Characteristics

- Experience-based test technique
 - Knowledge and experience includes
 - expected use of the software,
 - its environment,
 - likely defects,
 - distribution of those likely defects.
 - Designing, implementing, and executing tests at the same time
 - Often combined with black-box and white-box test techniques.

Summary



- Categories of test techniques
 - Black-box test technique
 - Not considering the internal structure
 - White-box test technique
 - Based on the internal structure
 - Experience-based test techniques
 - Based on knowledge and experience of stakeholders

Contents

- 4.1 Categories of Test Techniques
- 4.2 Black-box Test Techniques
- 4.3 White-box Test Techniques
- 4.4 Experience-based Test Techniques

Equivalence Partitioning

- **Equivalence partitioning**
(**Synonym:** *partition testing*): A black-box test technique in which test cases are designed to exercise equivalence partitions by using one representative member of each partition.
- **Equivalence partition**
(**Synonym:** *equivalence class*): A subset of the value domain of a variable within a component or system in which all values are expected to be treated the same based on the specification.

Equivalence Partitioning

- Inputs to the software or system are divided into groups that are expected to exhibit similar behavior.
- Equivalence partitions (or classes) can be found for
 - valid data, i.e., values that should be accepted
→ “valid equivalence partition.”
 - invalid data, i.e., values that should be rejected
→ “invalid equivalence partition.”

Equivalence Partitioning

- 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)

Equivalence Partitioning

- is applicable at all levels of testing.
- can be applied to
 - human input
 - input via interfaces to a system
 - interface parameters in integration testing
- Coverage =
$$\frac{\text{Number of equivalence partitions tested}}{\text{Total number of identified equivalence partitions}}$$
- To achieve 100% coverage, test cases must cover all identified partitions

Equivalence Partitioning

Example 1

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

Ideas:

- Valid: $1 \leq x \text{ and } x \leq 7$
- Invalid_{low}: $x < 1$ or
- Invalid_{high}: $x > 7$

Test data

5

0

14

Legend:

Red values: Invalid values
Green values: Valid values

Equivalence Partitioning

Example 2

- Input: Month as number
(1 = January, ..., 12 = December)

Ideas:

- Valid: $1 \leq x \text{ and } x \leq 12$
- Invalid_{low}: $x < 1$ or
- Invalid_{high}: $x > 12$

Test data

5

0

14

Legend:

Red values: Invalid values

Green values: Valid values

Boundary Value Analysis

- **Boundary value analysis:**

A black-box test technique in which test cases are designed based on boundary values.

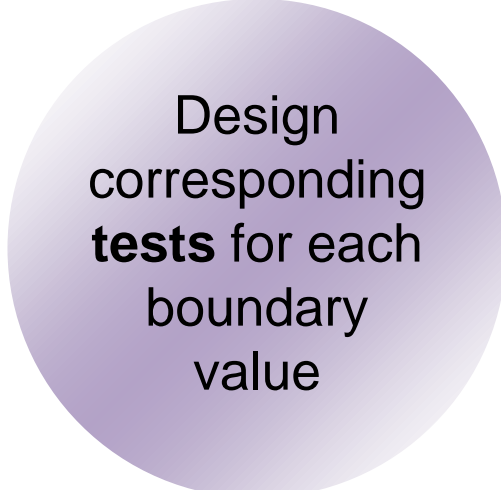
- can only be used when the partition is ordered, consisting of numeric or sequential data

- **Boundary value:**

A minimum or maximum value of an ordered equivalence partition.

Boundary Value Analysis

- Behavior at the edge of each equivalence partition is more likely to be incorrect than behavior 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

Boundary Value Analysis

- for all test levels
- relatively easy to apply
- defect finding capability is high
- often considered as an extension of equivalence partitioning or other black-box test design techniques
- Detailed specifications are helpful in determining the interesting boundaries
- 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).

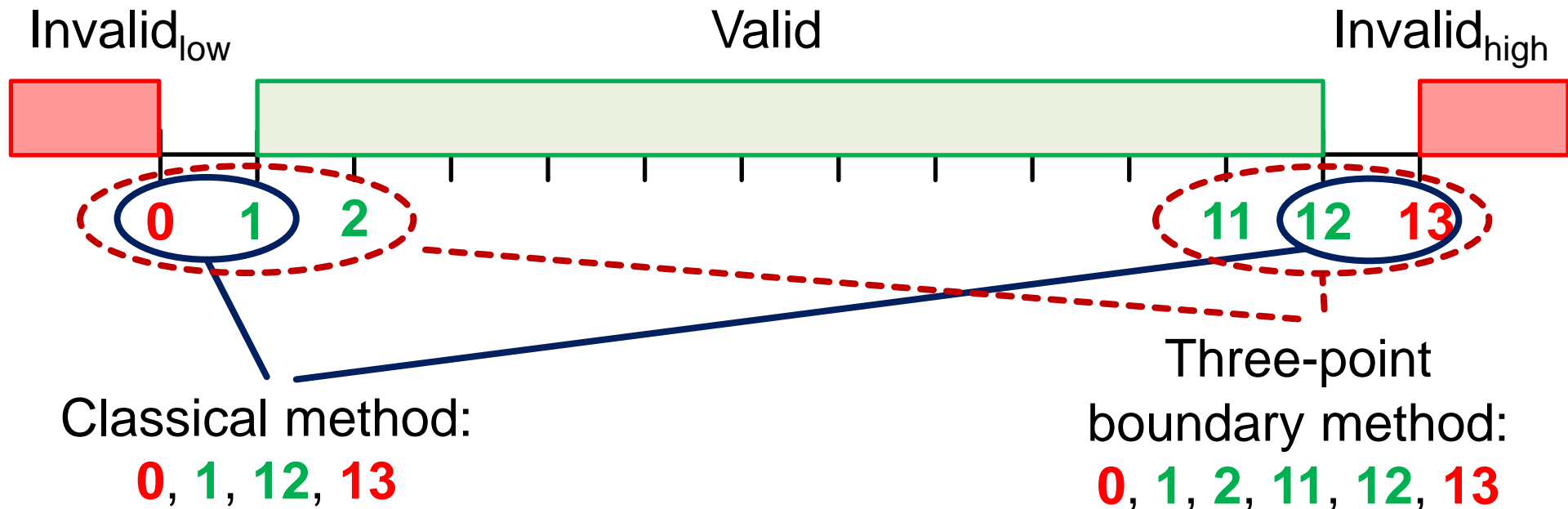
Boundary Value Analysis

- Classical method: The minimum and maximum values (or first and last values) of a partition are its boundary values
- Three-point boundary method: The values per boundary are the values before, at, and just over the boundary
- Coverage =
$$\frac{\text{Number of boundary values tested}}{\text{Total number of identified boundary test values}}$$

Boundary Value Analysis

Example

- Input: Month as number
(1 = January, ..., 12 = December)



Decision Table Testing

- **Decision table testing:**

A black-box test technique in which test cases are designed to exercise the combinations of conditions and the resulting actions shown in a decision table.

- **Decision table**

(*Synonym: cause-effect decision table*):

A table used to show sets of conditions and the actions resulting from them.

Decision Table Testing

- 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

Decision Table Testing

- A decision table contains
 - 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	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				
	Visit with nurse								
	Visit max. 30 minutes								

capture system requirements that contain logical conditions

Input conditions, often BOOLEAN

Idea: One test per column

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

Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

Decision Table Testing

- Example: 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		

Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

Decision Table Testing

- Example: Visiting rules in an hospital.
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		

Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

Decision Table Testing

- Example: Visiting rules in an hospital.
Reduction 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		

Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

Decision Table Testing

- Advantage
 - Creates combinations of conditions that otherwise might not have been exercised during testing
- When to use?
 - Action of the software depends on several logical decisions
 - For reviewing use cases/user stories with logical dependencies
- Coverage =
$$\frac{\text{Number of decision rules tested by at least 1 test case}}{\text{Total number of decision rules}}$$

State Transition Testing

- **State transition testing**
(**Synonym:** *finite state testing*):
A black-box test technique in which test cases are designed to exercise elements of a state transition model.
- **State transition diagram**
(**Synonym:** *state diagram*):
A diagram that depicts the states that a component or system can assume, and shows the events or circumstances that cause and/or result from a change from one state to another.

State Transition Testing

- Systems may respond differently to an event depending on current conditions or previous history
⇒ can be visualized with a state transition diagram.
- A state transition diagram shows
 - possible software states,
 - how the software enters,
 - how the software exits,
 - transitions between states,
 - the inputs or events that trigger state changes (transitions),
 - the actions which may result from those transitions.

State Transition Testing

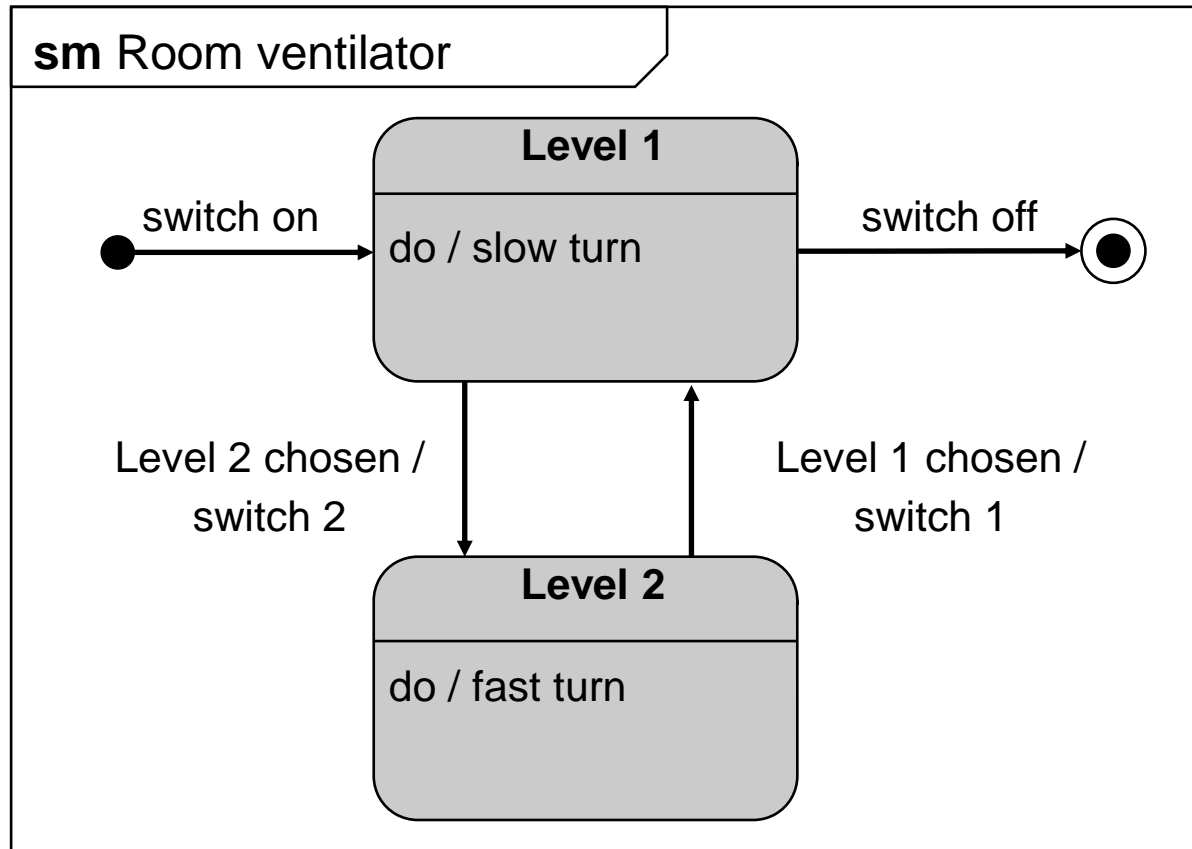
- The states of the system or object under test are separate, identifiable and finite in number
- A state transition table
 - shows the relationship between the states and inputs,
 - can highlight possible transitions that are invalid.
- State coverage =
$$\frac{\text{Number of identified states tested}}{\text{Total number of identified states}}$$
- Transition coverage =
$$\frac{\text{Number of identified transitions tested}}{\text{Total number of identified transitions}}$$

State Transition Testing

- Tests can be designed to
 - cover a typical sequence of states,
 - cover every state,
 - exercise every transition,
 - exercise specific sequences of transitions,
 - check invalid transitions.
- Usage of state transition testing:
 - Menu-based applications
 - Embedded software
 - Technical automation
 - Modelling a business object with specific states (in business scenarios)
 - Testing screen-dialogue flows (Web applications)

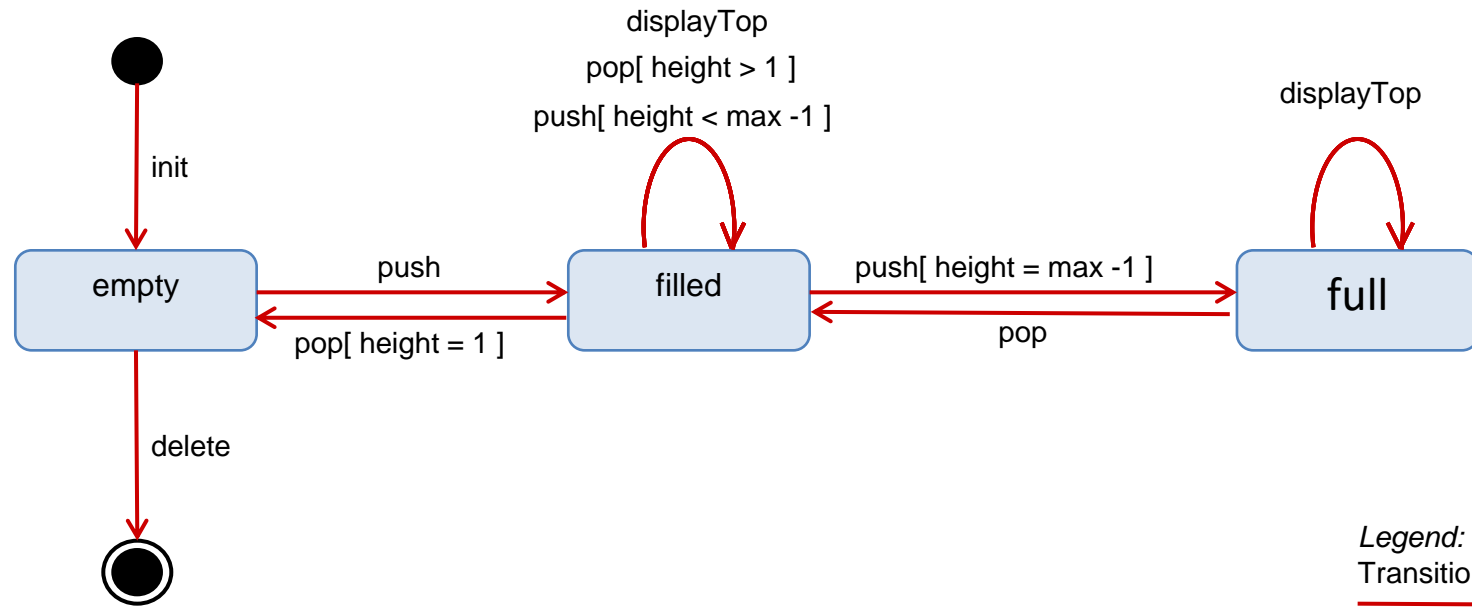
State Transition Testing

- Example 1: Fan



State Transition Testing

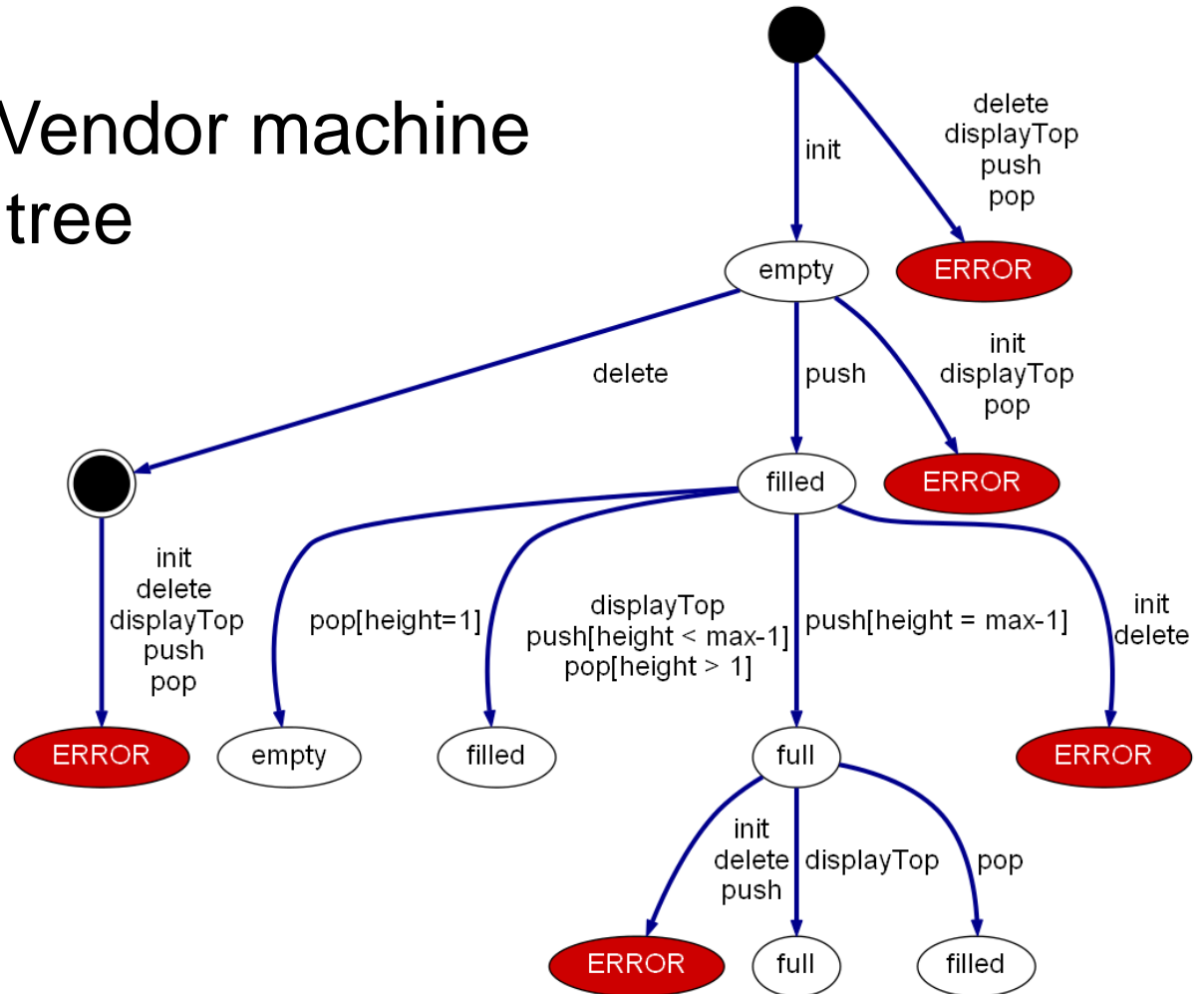
- Example 2: Vendor machine
- Status: „empty“, „filled“ (less then full), „full“



Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

State Transition Testing

- Example 2: Vendor machine
 - Transition tree



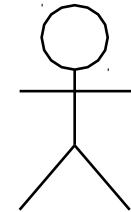
Source: Dr. K. Dussa-Zieger: Testen von Software-Systemen, FAU Erlangen-Nürnberg, SS 2007

Use Case Testing

- **Use case testing**
(**Synonyms:** *scenario testing, user scenario testing*):
A black-box test technique in which test cases are designed to exercise use case behaviors.
- **Use case:**
The specification of the behavior of a system with regards to its interaction with its users and any other systems.
 - Process level
Business process level: Business use case
 - System level
System process level: Use case
- **Specification:**
Documentation that provides a detailed description of a component or system for the purpose of developing and testing it.

Use Case Testing

- Components of a use case diagram
 - An actor represents
 - human user,
 - external hardware,
 - other component,
 - other system.
 - Use Case, describing what the system should do
 - System, might be as well a sub system or component

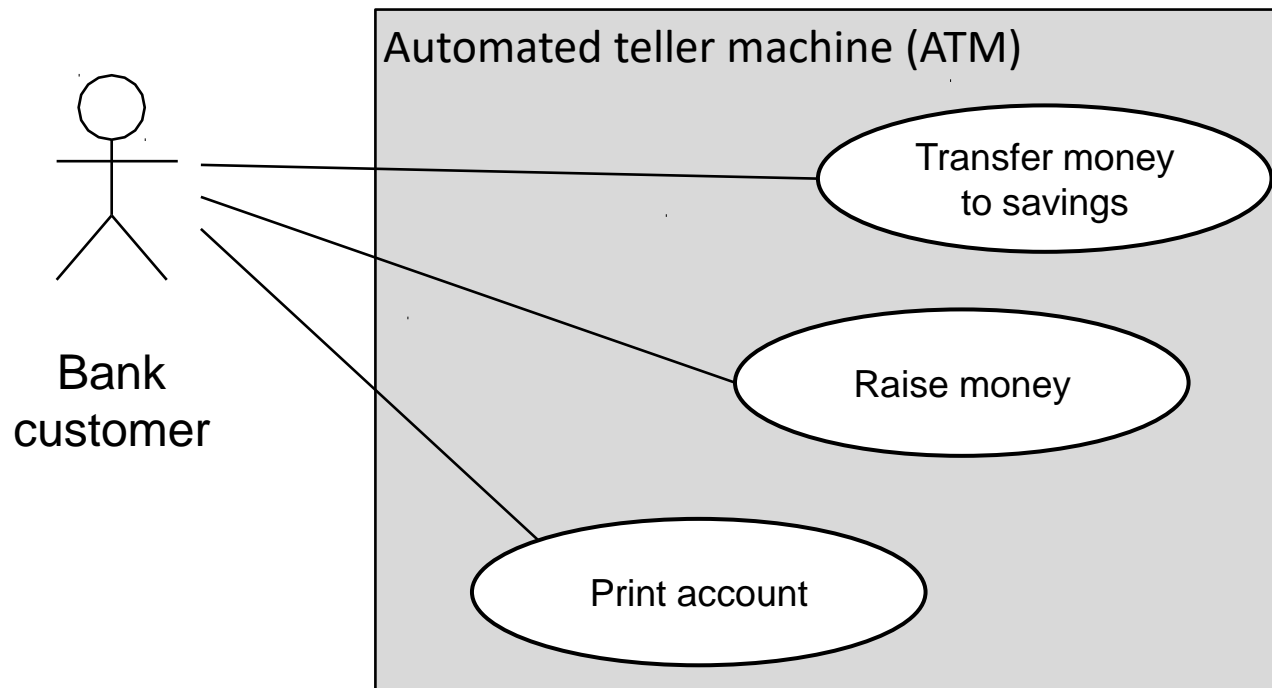


Actor



Use Case Testing

Example of a use case diagram



Use Case Testing

- A use case usually describes
 - basic behavior,
 - exceptional or alternative behaviors,
 - error handling.
- A use case contains
 - 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,
 - description in natural language.

Positive
Variants
Negative

Use Case Testing

- Example for a use case description

Id / Name	214 / Rent a car
Summary	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
Precondition	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
Postcondition	The rental system is ready to get customer data and to realize a lease contract
Activities	<ol style="list-style-type: none">1. Enter customer data. If customer is yet not registered ⇒ UC 12 <i>Register customer</i>.2. Enter desired car category3. Enter desired leasing period4. If a car is available in the desired period:<ol style="list-style-type: none">1. Reserve a car2. Enter credit card information3. Print contract and sign <p>Otherwise: Adapt item 2. or 3., if possible</p>

Use Case Testing

- Best practice: prioritization of use cases
 - 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
- Coverage =
$$\frac{\text{Number of use case behaviors tested}}{\text{Total number of use case behaviors}}$$

Use Case Testing

- Tests based on use cases
 - are designed to exercise
 - the defined basic behaviors,
 - exceptional or alternative behaviors,
 - error handling.
 - are typically used in
 - system test,
 - system integration test,
 - acceptance test.
 - find often defects related to integration of components
- Functional tests could be used as basis for non-functional tests, especially for performance tests

Use Case Testing

Excurses

User story

- Short, simple description of a feature told from the perspective of the person who desires the new capability, usually a user or customer of the system.

Proposed template:

As a <type of user>,
I want <some goal>
so that <some reason>

- Tests are based on acceptance criteria

As a scheduler
I want to update a
given appointment
so that I could add
another date

Example of a
user story

Summary



- Black-box test techniques
 - Equivalence partitioning
 - Boundary value analysis
 - Decision table testing
Recommended if behavior depends on several parameters and their combination
 - State transition testing
Recommended for automates
 - Use case testing
Standard techniques in typical IT projects
- Black-box test techniques could be used at any test level

Contents

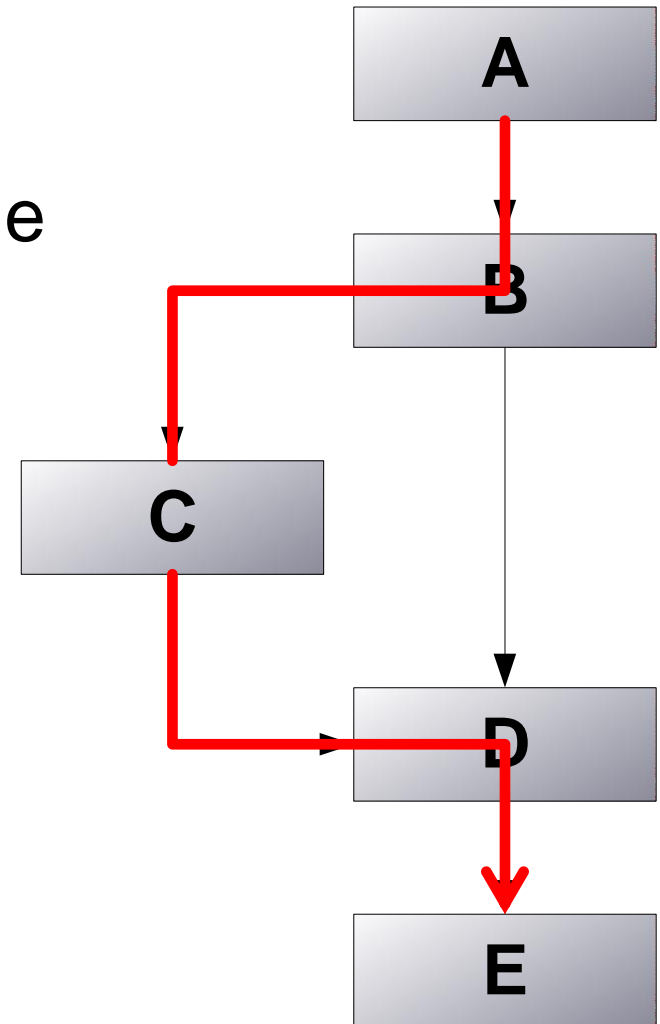
- 4.1 Categories of Test Techniques
- 4.2 Black-box Test Techniques
- 4.3 White-box Test Techniques
- 4.4 Experience-based Test Techniques

Statement Testing and Coverage

- **Statement testing**:
A white-box test technique in which test cases are designed to execute statements.
- **Statement** (***Synonym: source statement***):
An entity in a programming language, which is typically the smallest indivisible unit of execution.
- Coverage =
$$\frac{\text{Number of statements executed by tests}}{\text{Total number of executable statements}}$$

Statement Testing and Coverage

- Example 1
For 100 % statement coverage
1 test case required
A, B, C, D, E



Statement Testing and Coverage

- Example 2
For 100 % statement coverage
1 test case required
(x=1, y=2)
Expected result: z=3

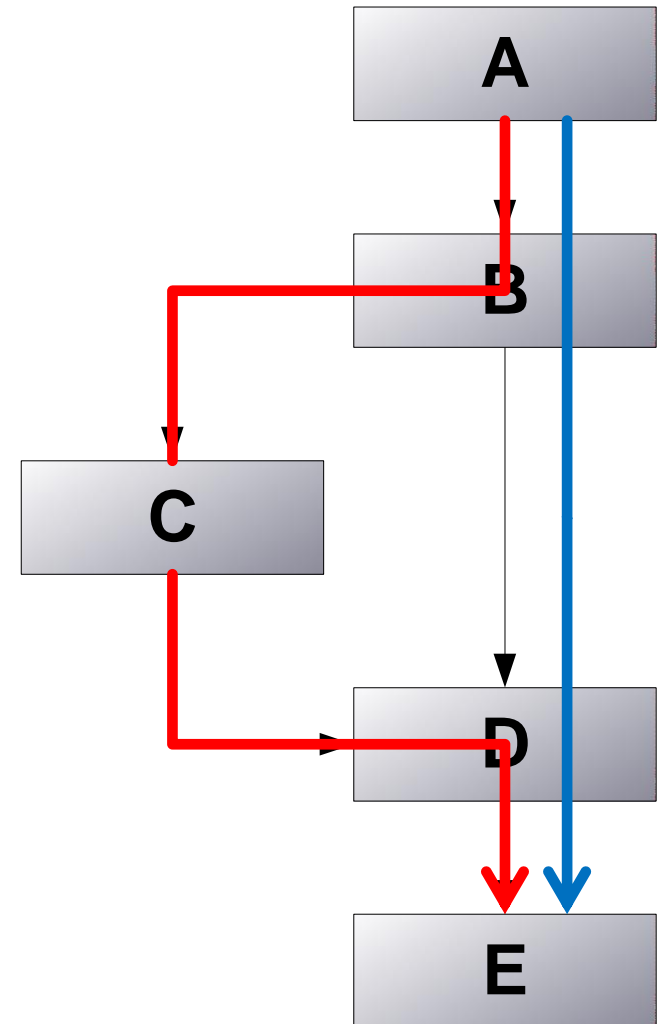
```
/* z is greater value+1 */  
int foo(int x, int y) {  
    int z = x;  
    if (y > x) {  
        z = y;  
    }  
    z = z + 1;  
    return z;  
}
```

Decision Testing and Coverage

- **Decision testing:**
A white-box test technique in which test cases are designed to execute decision outcomes.
- **Decision outcome:**
The result of a decision that determines the next statement to be executed.
- Coverage =
$$\frac{\text{Number of decision outcomes executed by tests}}{\text{Total number of decision outcomes}}$$
- Decision coverage considers
 - If/else clauses
 - case statements
 - while/do while loops

Decision Testing and Coverage

- Example 1
For 100 % decision coverage
2 test cases required
 - A, B, C, D, E
 - A, B, D, E



Decision Testing and Coverage

- Example 2
For 100 % decision coverage
2 test cases required
 - (x=1, y=2)
Expected result: z=3
 - (x=3, y=2)
Expected result: z=4

```
/* z is greater value+1 */  
int foo(int x, int y) {  
    int z = x;  
    if (y > x) {  
        z = y;  
    }  
    z = z + 1;  
    return z;  
}
```

The Value of Statement and Decision Testing

- “Statement-coverage criterion is so weak that it is generally considered useless.” [Myers*]
- Statement coverage and decision coverage should be considered as a minimal requirement
- Decision coverage is stronger than statement coverage:
 - 100% decision coverage guarantees 100% statement coverage
 - ⇒ but not vice versa.

**Source: Glenford J. Myers, Tom Badgett, Corey Sandler: The art of software testing, 3rd edition, 2012*



Summary

- White-box test techniques
 - Statement testing and coverage
 - Decision testing and coverage
- Achieving 100% decision coverage guarantees 100% statement coverage (but not vice versa).

Contents

- 4.1 Categories of Test Techniques
- 4.2 Black-box Test Techniques
- 4.3 White-box Test Techniques
- 4.4 Experience-based Test Techniques

Experience-based test technique

- Test cases are based on tester's
 - skill and intuition,
 - experience with similar applications and technologies.
- Coverage
 - difficult to assess and may not be measurable,
 - somehow depending on the tester's approach and experience.

Error Guessing

- **Error guessing:**

A test technique in which tests are derived on the basis of the tester's knowledge of past failures, or general knowledge of failure modes.

- Based on questions like
 - How did the application work in the past?
 - What kind of errors tend to be made?
 - Which kind of failures occurred in other similar applications?

Error Guessing

- Possible sources
 - Issues reported to 1st level support of previous version
 - Archived defects of previous version
 - User reports, e.g., in forums
 - Experience of end users
- Methodical approach
 - Create a list of possible errors, defects, and failures
 - Design tests that will expose those failures and the defects that caused them.

Exploratory Testing

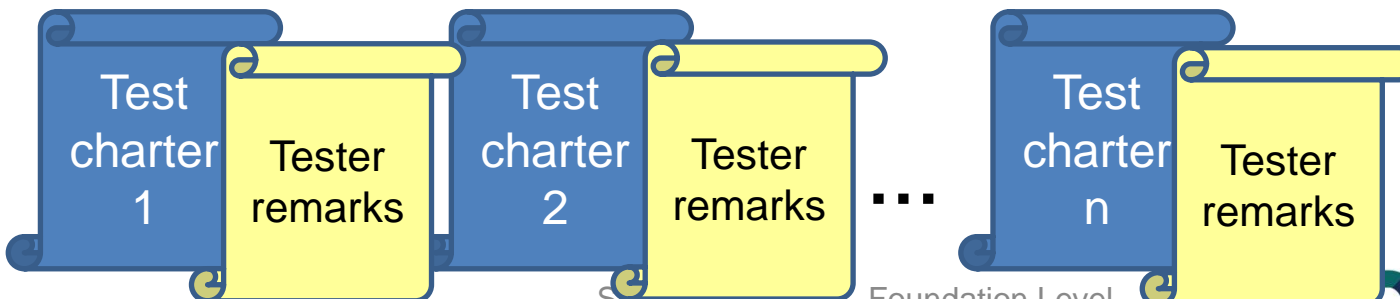
- **Exploratory testing:**
An approach to testing whereby the testers dynamically design and execute tests based on their knowledge, exploration of the test item and the results of previous tests.
- **Test session:**
An uninterrupted period of time spent in executing tests.
In exploratory testing, each test session is focused on a charter, but testers can also explore new opportunities or issues during a session.
The tester creates and executes on the fly and records their progress.

Exploratory Testing

- Informal (not pre-defined) tests are
 - designed,
 - executed,
 - logged,
 - evaluated dynamically during test execution.
- Test results are used to
 - learn more about the component or system,
 - to create tests for the areas that may need more testing.

Exploratory Testing

- Session-based testing to structure the activity; for example a testing day; time-box approach
 - 09:00 Planning session
 - 09:30 *Individual testing*
 - 13:00 Alignment session
 - 13:30 *Individual testing*
 - 17:00 Closing session, discussion of failures
 - Evaluation of testers remarks



4.2 →

Exploratory Testing

- Recommendation
 - useful when
 - few or inadequate specifications available
 - significant time pressure on testing.
 - complement to other testing techniques.
- Exploratory testing is strongly associated with reactive test strategies



Checklist-based Testing

- **Checklist-based testing:**

An experience-based test technique whereby the experienced tester uses a high-level list of items to be noted, checked, or remembered, or a set of rules or criteria against which a product has to be verified.

Checklist-based Testing

- Based on a checklist testers
 - design tests,
 - implement tests,
 - execute tests.
- Dealing with checklists
 - Creating a new one during analysis
 - Expanding/modifying an existing one
 - Using without modification
- Checklists might support various test types, such as
 - functional testing,
 - non-functional testing.

Checklist-based Testing

- Checklists can be built based on
 - experience,
 - business processes,
 - knowledge about what is important for the user,
 - an understanding of why and how software fails.
- Checklist-based testing
 - provides guidelines,
 - offers a degree of consistency,
 - will probably produce some variability in the actual testing, resulting in
 - greater coverage,
 - less repeatability.



Summary

- Experience-based test techniques
 - Error guessing
 - Exploratory testing
 - Checklist-based testing
- Experience-based test techniques
 - could support formal testing,
 - are based on knowledge and skills of participating testers,
 - are a reliable approach in practice.