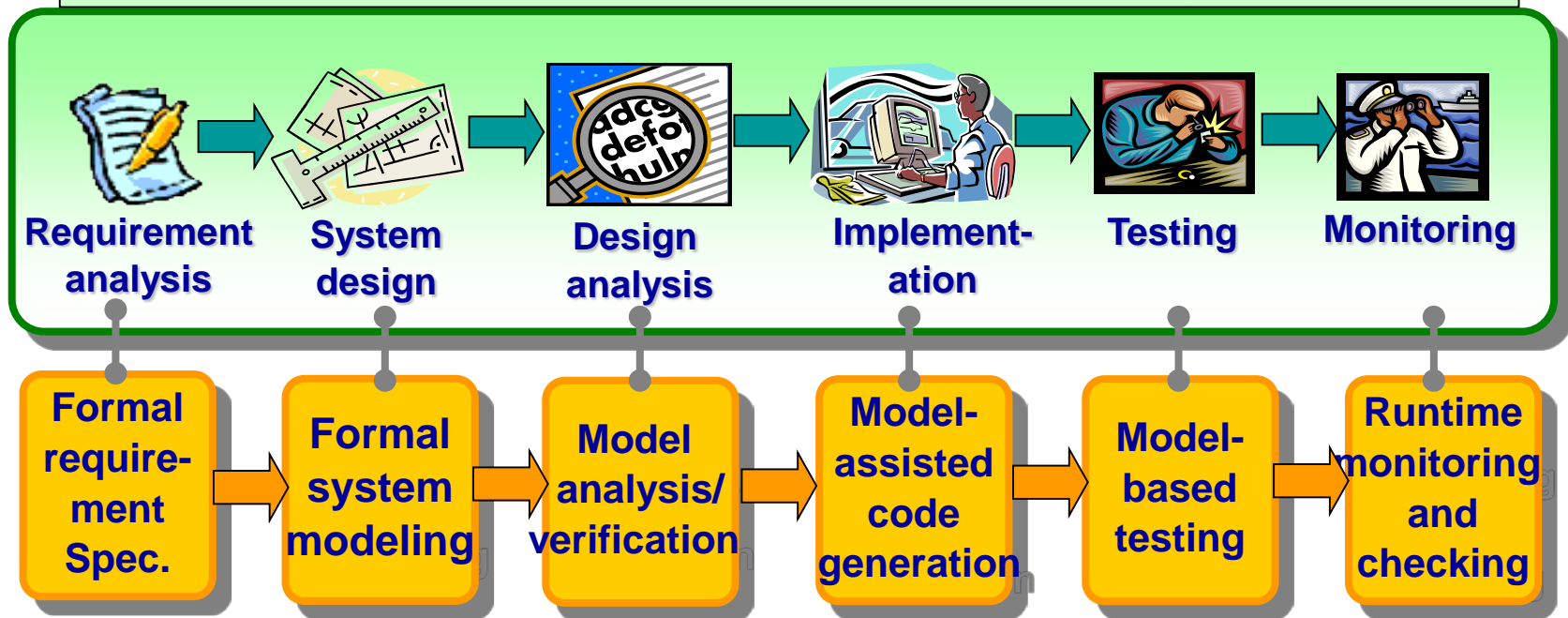


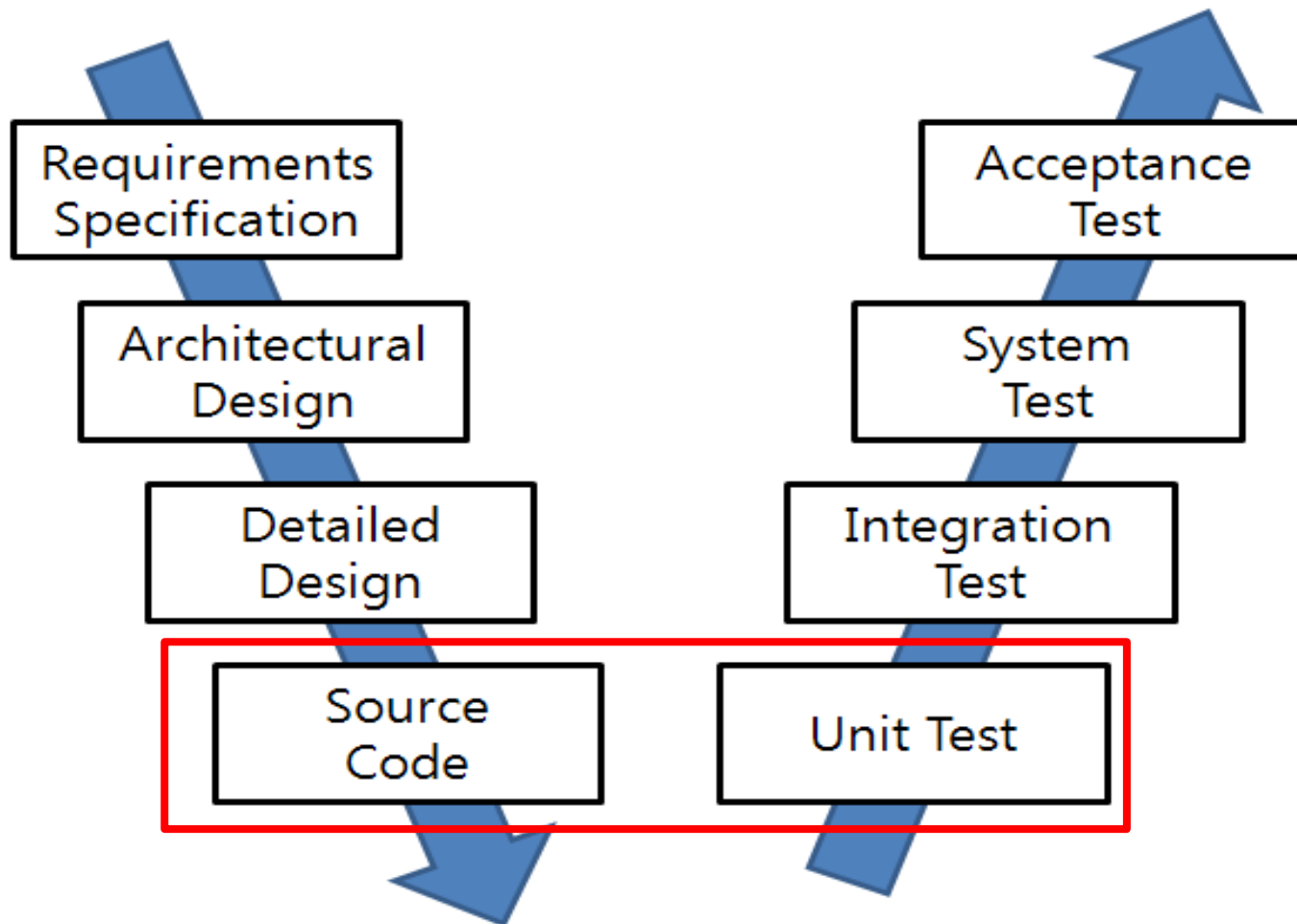
Software Development Cycle

A SW Development Framework for SW with High Assurance



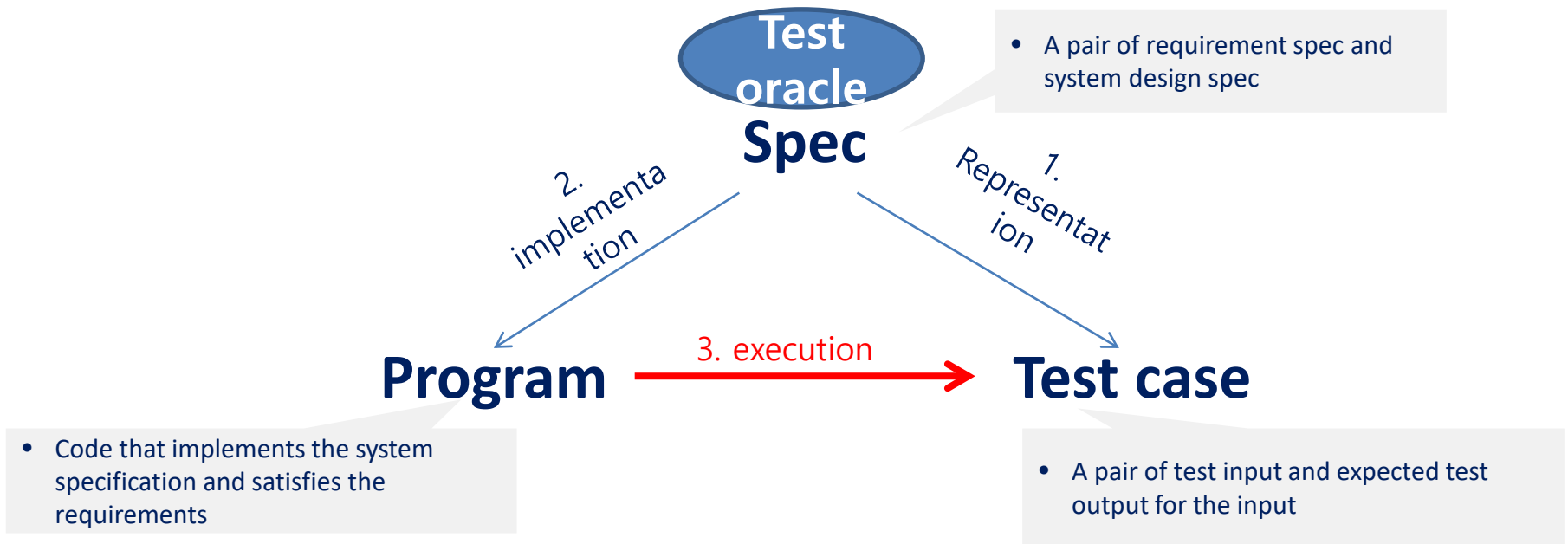
SW Development and Testing Model (a.k.a. V model)

Manual
Labor



Abstraction

Foundation of Software Testing



Multiple targets for software testing

1. Does the test cases represent the requirement spec correctly?
→ Scenario based testing (black-box testing)
2. Is the design spec implemented as program correctly?
→ Model-based testing (grey-box testing)
3. Does the program satisfy test cases correctly?
→ Code-based testing (white-box testing)

Black Box Testing

- A main goal of testing is to generate multiple test cases, one of which may reveal a bug.
- Black box testing concerns only input/output of a target program (i.e., ignore program code)
 - Ex1. Requirement specification based testing
 - Ex2. Random (input generation) testing
 - Ex3. Category partitioning method
 - Ex4. T-way testing
- Advantage of black box testing
 - Intuitive and simple
 - Requires little expertise on program/code analysis techniques
 - Requires less effort compared to white-box testing
 - cheaper but less effective

Example of Blackbox Testing Technique:

The Category-Partition Method for Specifying and Generating Functional Tests (Thomas J. Ostrand and Marc J. Balcer [CACM, 1988])

Content:

- Introduction.
- The category-partition method:
 - characteristics.
 - the method.
 - examples.
- Other methods.

The goal of functional testing

- To find discrepancies between the **actual behavior** of the implemented system's function and the **desired behavior** as described in the system's functional specification.

How to achieve this goal ?

- Tests have to be execute for all the system functions.
- Tests have to be designed to maximize the chances of finding errors in the software.

Functional test can be derived from 3 sources:

1. The software specification.
2. Design information.
3. The code itself.

Partition - The standard approach

- The main idea is to **partition the input domain** of function being tested, and then **select test data for each class** of the partition.
- The problem of all the existing techniques is the **lack of systematic**.

The category partition method - main characteristics:

- **The test specification :**
 - is concise and uniform representation of the test information for a function.
 - it can be easily modified.
 - it gives the tester a logical way to control the volume of tests.

The category partition method - main c haracteristics (cont.):

- Using **generator tool** help us :
 - to provides an automated way to produce thorough tests.
 - to avoid impossible or undesirable tests.
- The method emphasizes both the specification **coverage** and the **error detection** aspects of testing.

A strategy for test case generation

1. Transform the system's specification to be more concise and structured.
2. Decompose the specification into **functional units** - to be tested independently.
3. Identify the **parameters** and **environment conditions**.

A strategy for test case generation (cont)

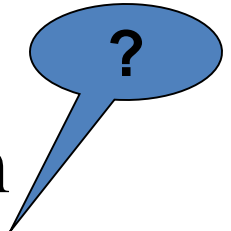
4. Find **categories** that characterize each parameter and environment condition.
5. Every category should be partitioned into distinct **choices** .



formal test specification

A strategy for test case generation (cont)]

6. **test frames** - set of choices, one from each category.



test cases - test frame with specific values for each choices.

test scripts - sequence of test cases.

Example

Command: find

Syntax: find <pattern> <file>

Function: The find command is used to locate one or more instance of a given pattern in a text file. All lines in the file that contain the pattern are written to standard output. A line containing the pattern is written only once, regardless of the number of times the pattern occurs in it.

The pattern is any sequence of characters whose length does not exceed the maximum length of a line in the file .To include a blank in the pattern, the entire pattern must be enclosed in quotes (“”).To include quotation mark in the pattern , two quotes in a row (“ ”) must be used.

Example:

find john myfile

display lines in the file **myfile** which contain **john**

find "john smith" in myfile

display lines in the file **myfile** which contain **john smith**

find "john"" smith" in myfile

display lines in the file **myfile** which contain **john" smith**

Categories

Parameters:

Pattern size:

- empty
- single character
- many character
- longer than any line in the file

Quoting:

- pattern is quoted
- pattern is not quoted
- pattern is improperly quoted

Embedded blanks:

- no embedded blank
- one embedded blank
- several embedded blanks

Embedded quotes:

- no embedded quotes
- one embedded quotes
- several embedded quotes

File name:

- good file name
- no file with this name
- omitted

Environments:

Number of occurrence of pattern in file:

- none
- exactly one
- more than one

Pattern occurrences on target line:

- one
- more than one

Total Tests frames:
1944 (=4*3*3*3*3*3*2)

Test Frame - Example:

Pattern size : empty

Quoting : pattern is quoted

Embedded blanks : several embedded blanks

Embedded quotes : no embedded quote

File name : good file name

Number of occurrence of pattern in file : none

Pattern occurrence on target line : one

Adding Constraints to Reduce #of TC'S

Parameters:

Pattern size:

empty	[property Empty]
single character	[property NonEmpty]
many character	[property NonEmpty]
longer than any line in the file	[property NonEmpty]

Quoting:

pattern is quoted	[property Quoted]
pattern is not quoted	[if NonEmpty]
pattern is improperly quoted	[if NonEmpty]

Embedded blanks:

no embedded blank	[if NonEmpty]
one embedded blank	[if NonEmpty and Quoted]
several embedded blanks	[if NonEmpty and Quoted]

Embedded quotes:

no embedded quotes	[if NonEmpty]
one embedded quotes	[if NonEmpty]
several embedded quotes	[if NonEmpty]

File name:

good file name
no file with this name
omitted



Total Tests frames:
678

Environments:

Number of occurrence of pattern in file:

none	[if NonEmpty]
exactly one	[if NonEmpty] [property Match]
more than one	[if NonEmpty] [property Match]

Pattern occurrences on target line:

one	[if Match]
more than one	[if Match]

Parameters:

Pattern size:

empty	[property Empty]
single character	[property NonEmpty]
many character	[property NonEmpty]
longer than any line in the file	[error]

Quoting:

pattern is quoted	[property quoted]
pattern is not quoted	[if NonEmpty]
pattern is improperly quoted	[error]

Embedded blanks:

no embedded blank	[if NonEmpty]
one embedded blank	[if NonEmpty and Quoted]
several embedded blanks	[if NonEmpty and Quoted]

Embedded quotes:

no embedded quotes	[if NonEmpty]	
one embedded quotes	[if NonEmpty]	
several embedded quotes	[if NonEmpty]	[single]

File name:

good file name	
no file with this name	[error]
omitted	

Total Tests frames:
40

Environments:

Number of occurrence of pattern in file:

none	[if NonEmpty]	[single]
exactly one	[if NonEmpty]	[property Match]
more than one	[if NonEmpty]	[property Match]

Pattern occurrences on target line:

one	[if Match]	
more than one	[if Match]	[single]

Test Frame :

Test case 28 : (Key = 3.1.3.2.1.2.1.)

Pattern size : many character

Quoting : pattern is quoted

Embedded blanks : several embedded blanks

Embedded quotes : one embedded quote

File name : good file name

Number of occurrence of pattern in file : exactly none

Pattern occurrence on target line : one

Command to set up the test:

```
copy/testing/sources/case_28 testfile
```

find command to perform the test:

```
find "has" "one quote" testfile
```

Instruction for checking the test :

the following line should be display:

This line has " one quote on it

White Box Testing (1/2)

- White box testing concerns program code itself
- Many different viewpoints on “program code”
 - program code as a graph (i.e., structural coverage)
 - program code as a set of logic formulas (i.e., logical coverage)
 - program code as a set of execution paths (i.e., behavioral/dynamic coverage)
- Advantages:
 - More effective than blackbox testing in general
 - Can measure the testing progress quantitatively based on coverage achieved
- Should be used with blackbox testing together for maximal bug detection capability
 - Blackbox testing and whitebox testing often explore different segments of target program space

White Box Testing (2/2)

- Coverage is a **good predictor/indicator of testing effectiveness**
 - Utilizing correlation between structural coverage and fault detection ability

