Lecture 06-08: Software Testing

EGCI341

Outline

- Verification and validation
- Software testing

Verification and Validation

Verification:

"Are we building the product right”

The software should conform to its ***specification***

Validation:

"Are we building the right product”

The software should do what the ***user really requires***

Example


TRUE-H
08:39
93%
HB1234
BOB
HOME BUDDY
1 มองหาแบบสวยโดนใจ →
2 พูดคุยกับผู้เชี่ยวชาญ →
3 ออกแบบบ้านที่ใช่ →
4 ค้นหาวัสดุน่าใช้ →
5 คำนวณงบที่ใช้ →
6 สินเชื่อบ้านไวไว (เร็วๆนี้) →
7 ติดตามสถานะหน้างาน →
MAIN MENU
PROMOTION
REWARD
WALLET
HOME BUDDY


SafeAtHome
11:40 AM
55%
สวัสดีครับ คุณลลิตา
ไม่ต้องเกรงใจ เราพร้อมบริการ
เล่าความต้องการของคุณได้ที่นี่
สร้างบ้านใหม่
พูดคุยผู้เชี่ยวชาญ
บริการสะดวกทันใจ
ปรับปรุงบ้าน
เพิ่มเติม →
ทั้งหลัง
หลังคา
ฝ้าเพดาน
HOME
MATERIALS
REWARDS
SCG FAMILY
Account

Verification and Validation Processes

Two principal objectives

- Discovery of defects in a system
- Assessment of whether the system is useful and useable in an operational situation

Example of Defects in a software

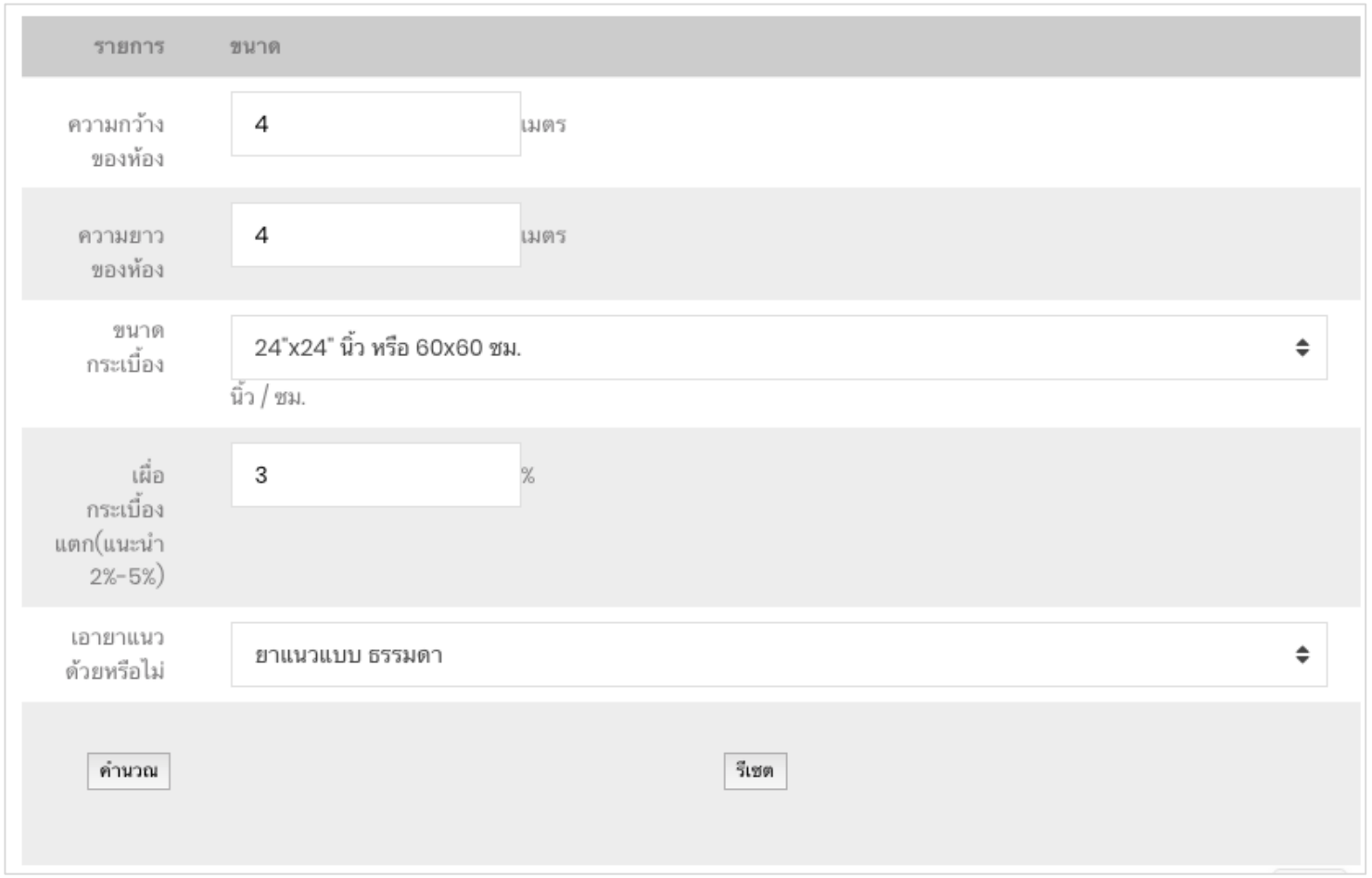
Incorrect default parameters

//Javascript (es5)

```
function add(a, b) {
// if "a" is 0, 1 is assigned to "a".
  a=a||1;
  b=b||2;
  return a + b;
}

let result = add(0, 0); // result = 3
console.log(result);
```

Example of User Interface Design



(1)

(2)

Verification and Validation Goals

- Verification and validation should establish confidence that the software is fit for purpose
- This does NOT mean completely free of defects
- It must be good enough for its intended use

Static and Dynamic Verification

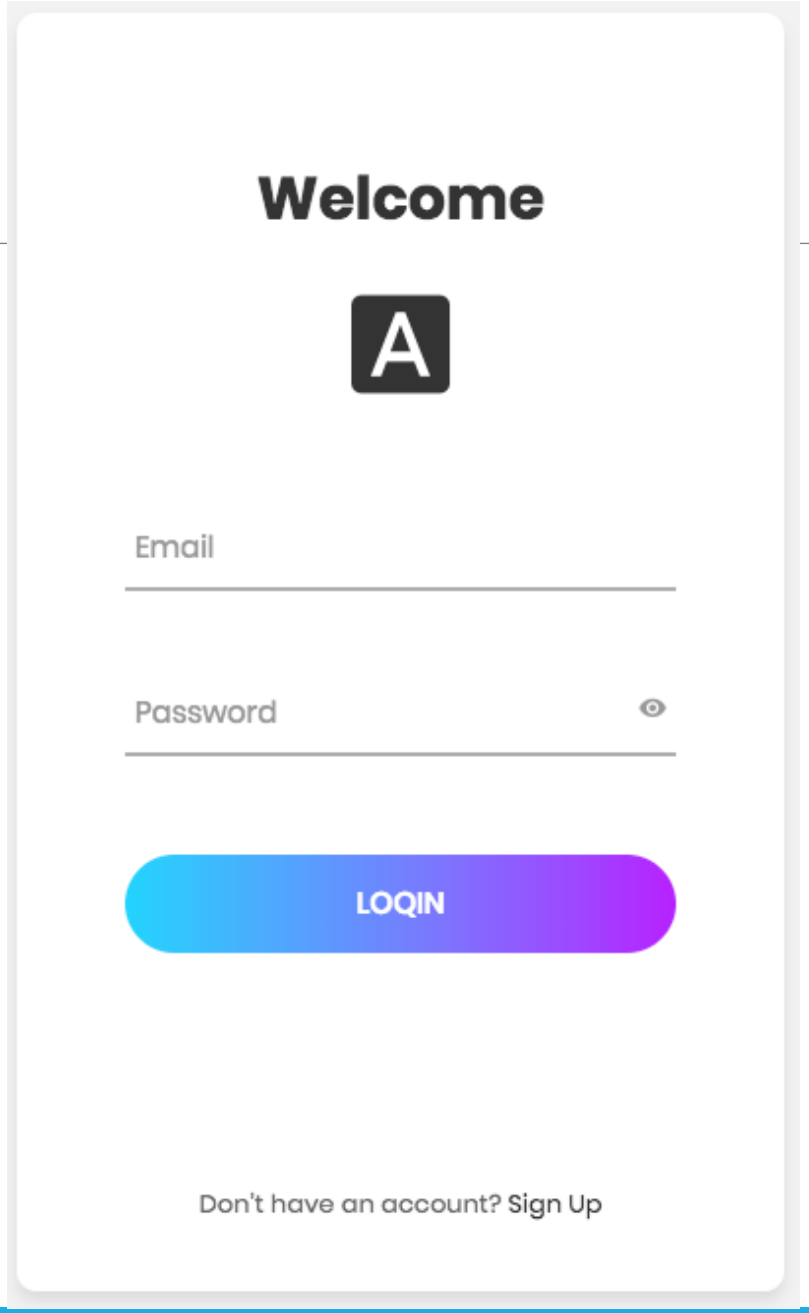
Software inspections

- Concern with analysis of the static system representation to discover problems (static verification)
- Supplement by tool-based document and code analysis

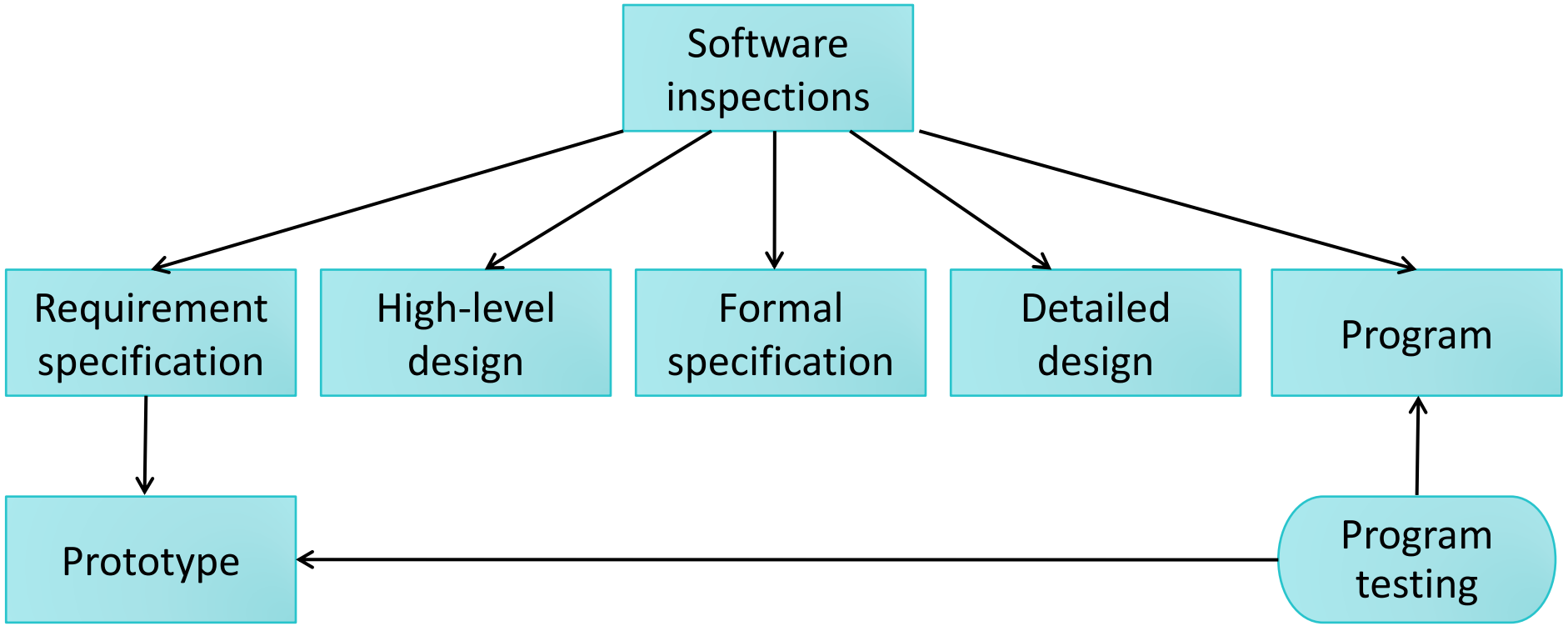
Software testing

- Concern with exercising and observing product behaviour (dynamic verification)
- The system is executed with test data and its operational behaviour is observed

Software Inspection



Static and Dynamic Verification and Validation



Program Testing

- Testing can reveal the presence of errors (NOT their absence)
- Only validation technique for non-functional requirements has to be executed to see how it behaves

Types of Testing

Defect testing

- To discover system defects
- A successful defect test is one which reveals the presence of defects in a system

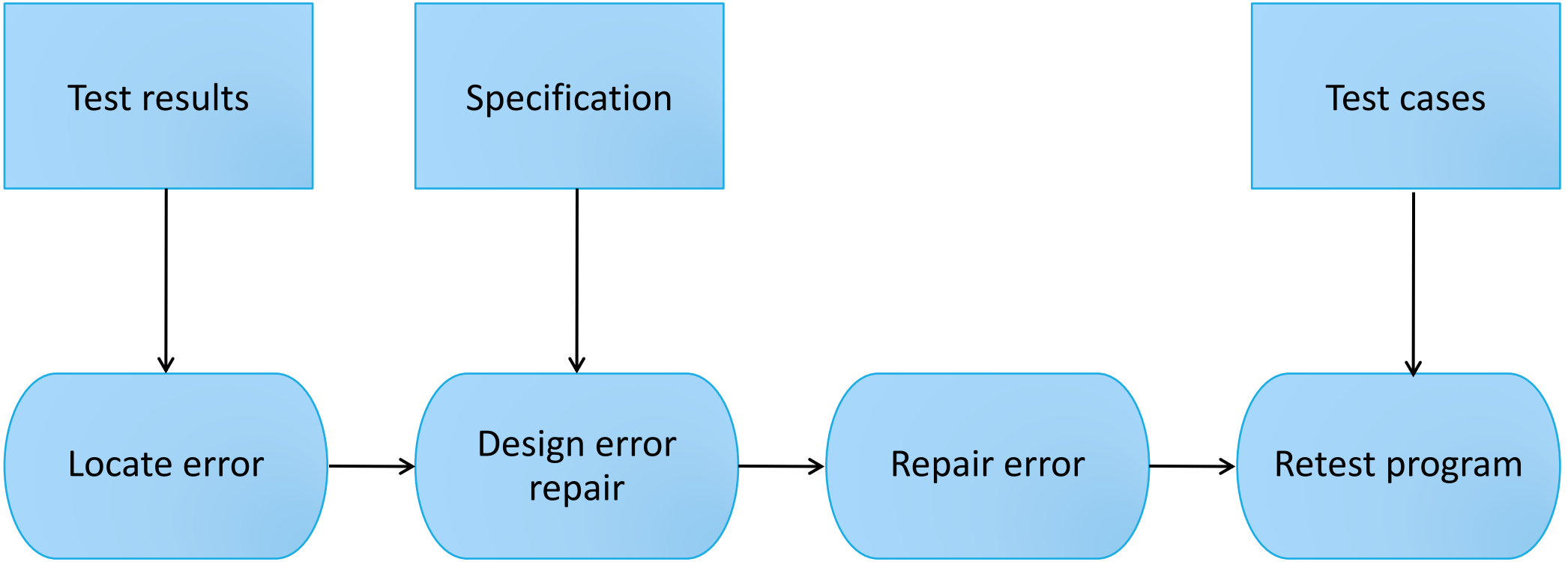
Validation testing

- To show that the software meets its requirements
- A successful test shows that a requirements has been properly implemented

Testing and Debugging

- Defect testing and debugging are distinct processes
- Verification and validation are concerned with establishing the existence of defects in a program
- Debugging is concerned with locating and repairing these errors
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.

Debugging Process



Verification and Validation Planning

- Careful planning is required to get the most out of testing and inspection processes
- Planning should start early in the development process
- The plan should identify the balance between static verification and testing
- Test planning is about defining standards for the testing process rather than describing product tests

The Structure of a Software Test Plan

- Testing process
- Requirements traceability
- Tested items
- Testing schedule
- Test recording procedures
- Hardware and software requirements
- Constraints

Software Inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects
- Inspections do not require execution of a system so may be used before implementation
- They may be applied to any representation of the system (requirements, design, configuration data, test data, etc.)
- They have been shown to be an effective technique for discovering program errors

Inspections and Testing

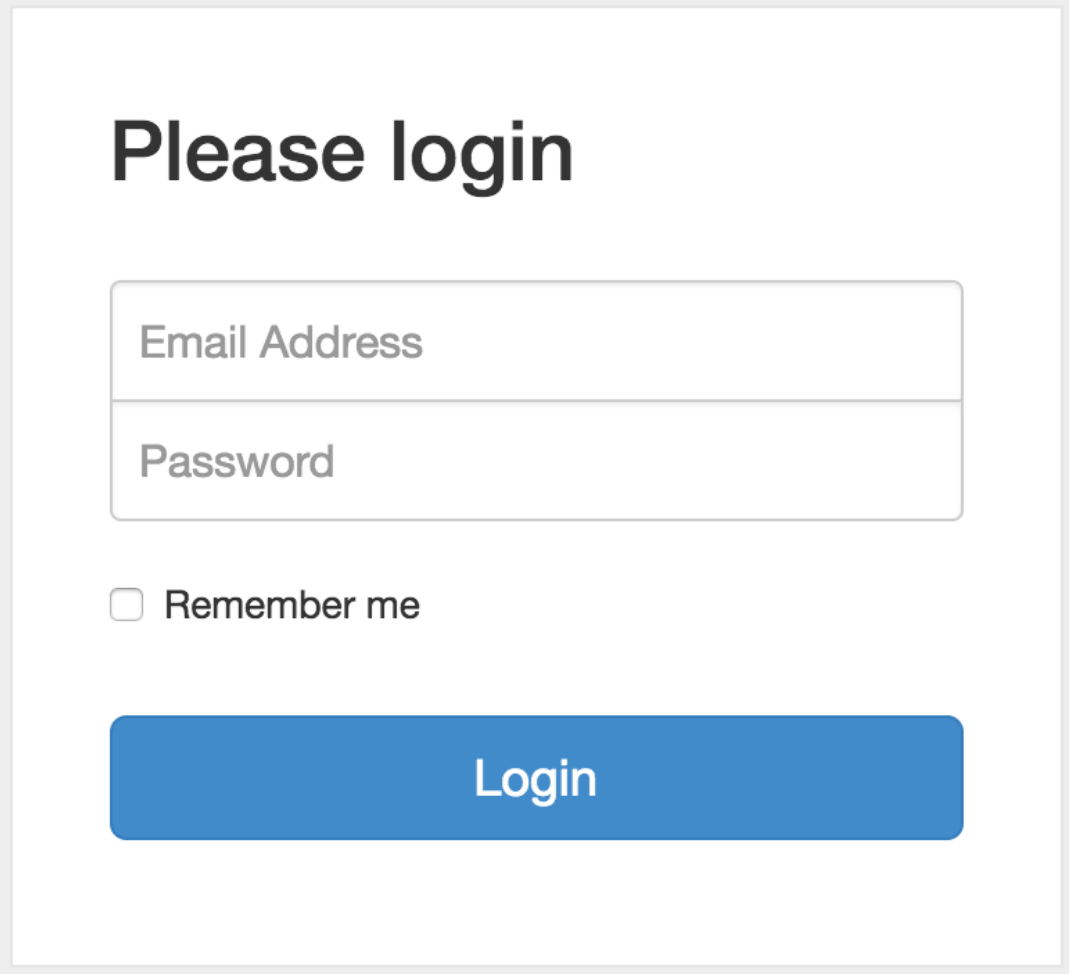
- Inspections and testing are complementary and not opposing verification techniques
- Both should be used during the verification and validation processes
- Inspections can check conformance with a specification but not conformance with the customer's real requirements
- Inspections cannot check non-functional characteristics such as performance, usability, etc.

Inspection Procedure

- System overview presented to inspection team
- Code and associated documents are distributed to inspection team in advance
- When inspection takes place, discovered errors are noted
- Modifications are made to repair discovered errors
- Reinspection may or may not be required

Example of Inspection

Inspect the UI of Login form and find out defects



Example of Inspection (cont.)

Improvement after inspection

Automated Static Analysis

- Static analyser is a software tool for source text processing
- They parse the program text and try to discover potentially erroneous conditions and bring these to the attention of the verification and validation team
- Static analyser is very effective as an aid to inspections
- Static analyser is a supplement tool but not a replacement for inspections

Stages of Static Analysis

Control flow analysis

- Checks for loops with multiple exit or entry points, finds unreachable code, etc.

Data usage analysis

- Detect uninitialized variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.

Interface analysis

- Checks the consistency of routine and procedure declarations and their use

Example of Data Usage Analysis

```
//es6
function foo() {
  let num=1;
  // ... too many statements
      if (num==1) {
           // different scope!
           let num=2;
      }
    console.log(num);
    //1 (Is it a correct answer?)
}
foo()
```

Stages of Static Analysis (Cont.)

Information flow analysis

- Identify the dependencies of output variables
- Do not detect anomalies itself but highlights information for code inspection or review
- Ex. “10/10/10” --> “10 October 2010”

Path analysis

- Identify paths through the program and sets out the statements executed in that path

Use of Static Analysis

C Language

- Weak typing and many errors are undetected by the compiler

Java Language

- Less cost-effective
- Strong type checking
- Detect many errors during compilation

Testing Process

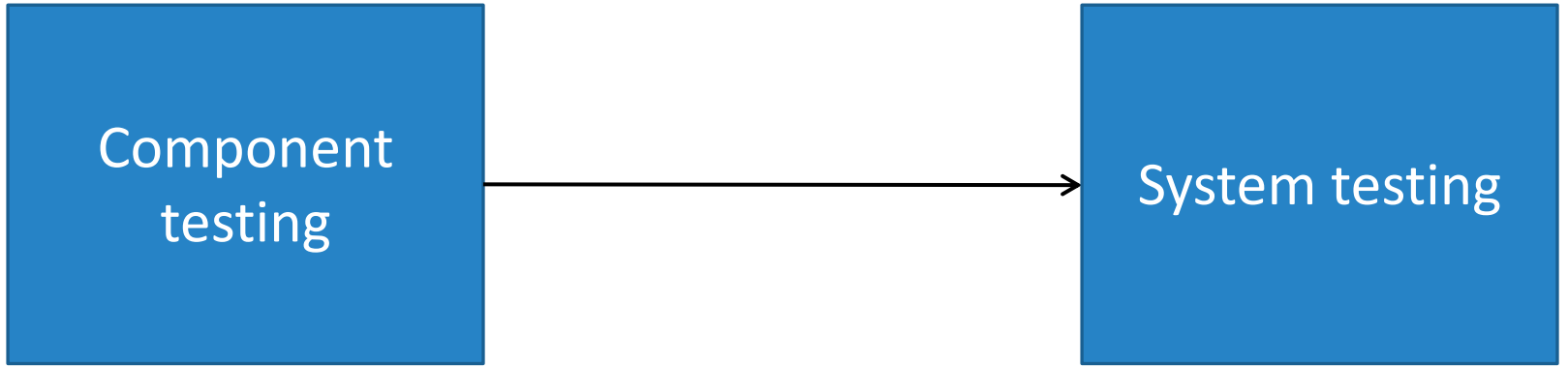
Component Testing

- Testing of individual program components
- Responsibility of the component developers

System Testing

- Testing of component's groups integrated to a system or subsystem
- Responsibility of the independent testing team
- Tests are based on a system specification

Testing Phases



Software developer

Independent testing team

Testing Process Goals

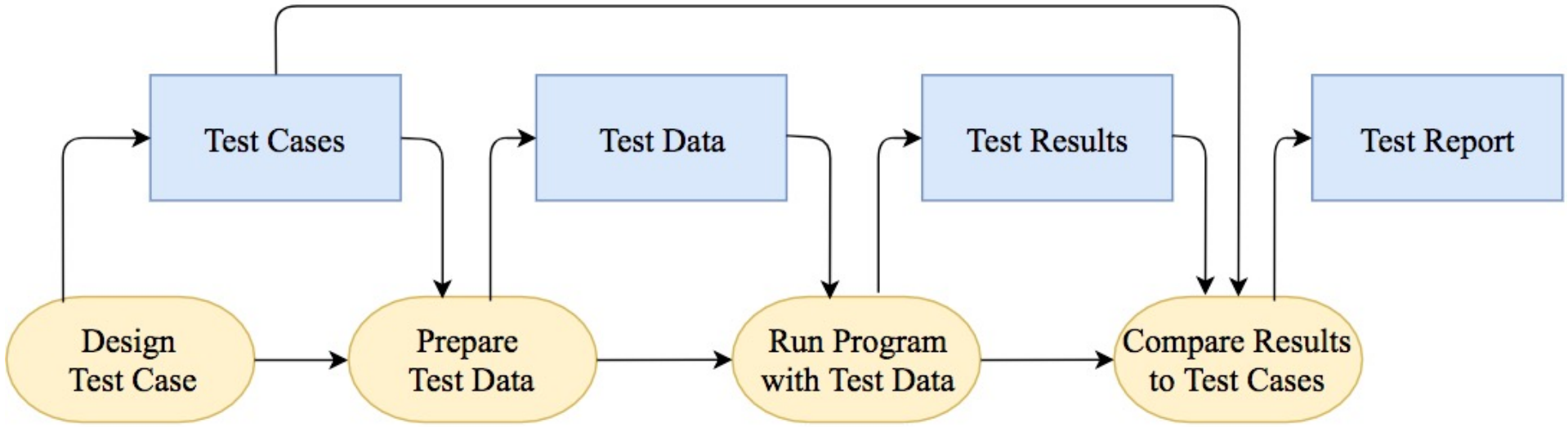
Validation testing

- Demonstrate to the developer and the customer that the software meets requirements
- A successful test shows that the system operates as intended use

Defect testing

- Discover faults or defects in the software where its behavior is incorrect or not in conformance with its specification
- A successful test makes that the system perform incorrectly and so exposes a defect in the system

Software Testing Process



Testing Approaches

Architectural validation

- Top-down integration testing is better at discovering errors in the system architecture

System demonstration

- Top-down integration testing allows a limited demonstration at an early stage in the development

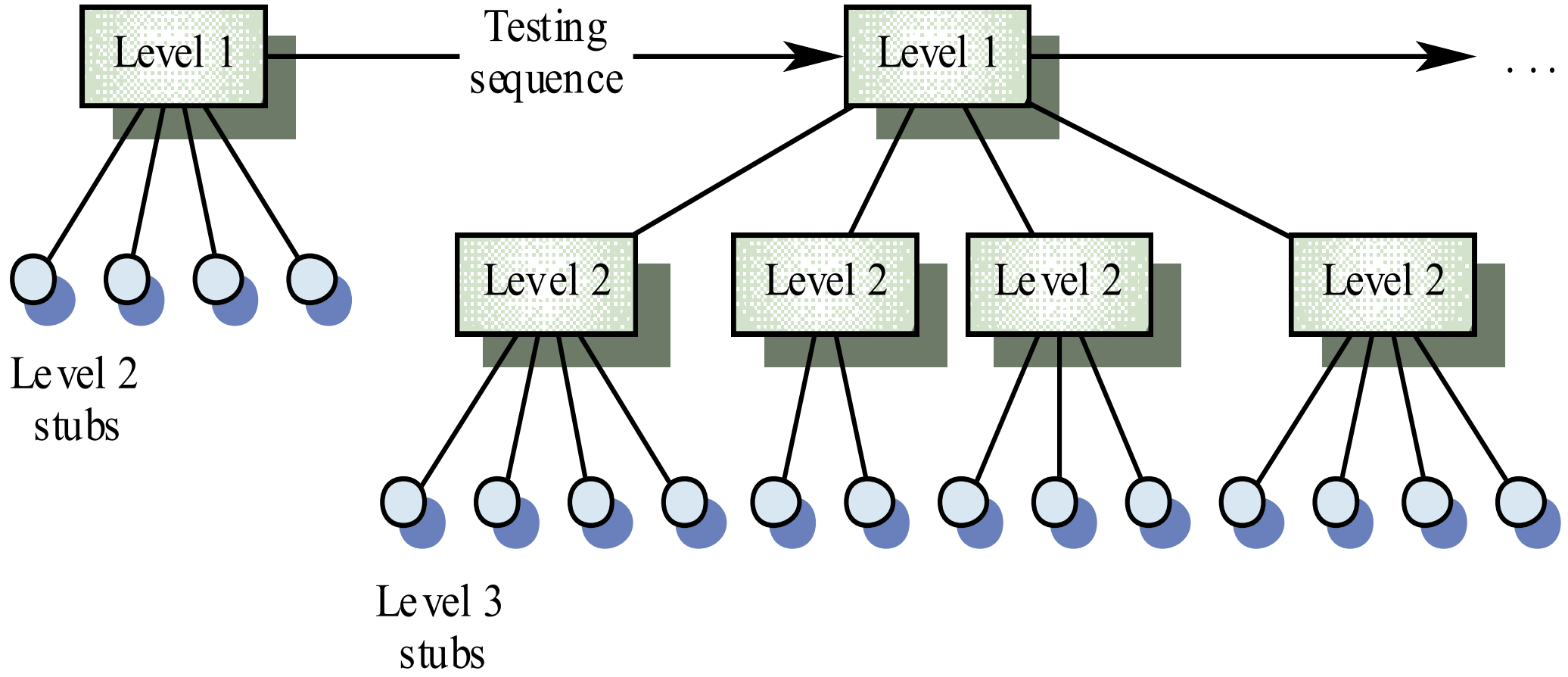
Test implementation

- Easier with bottom-up integration testing

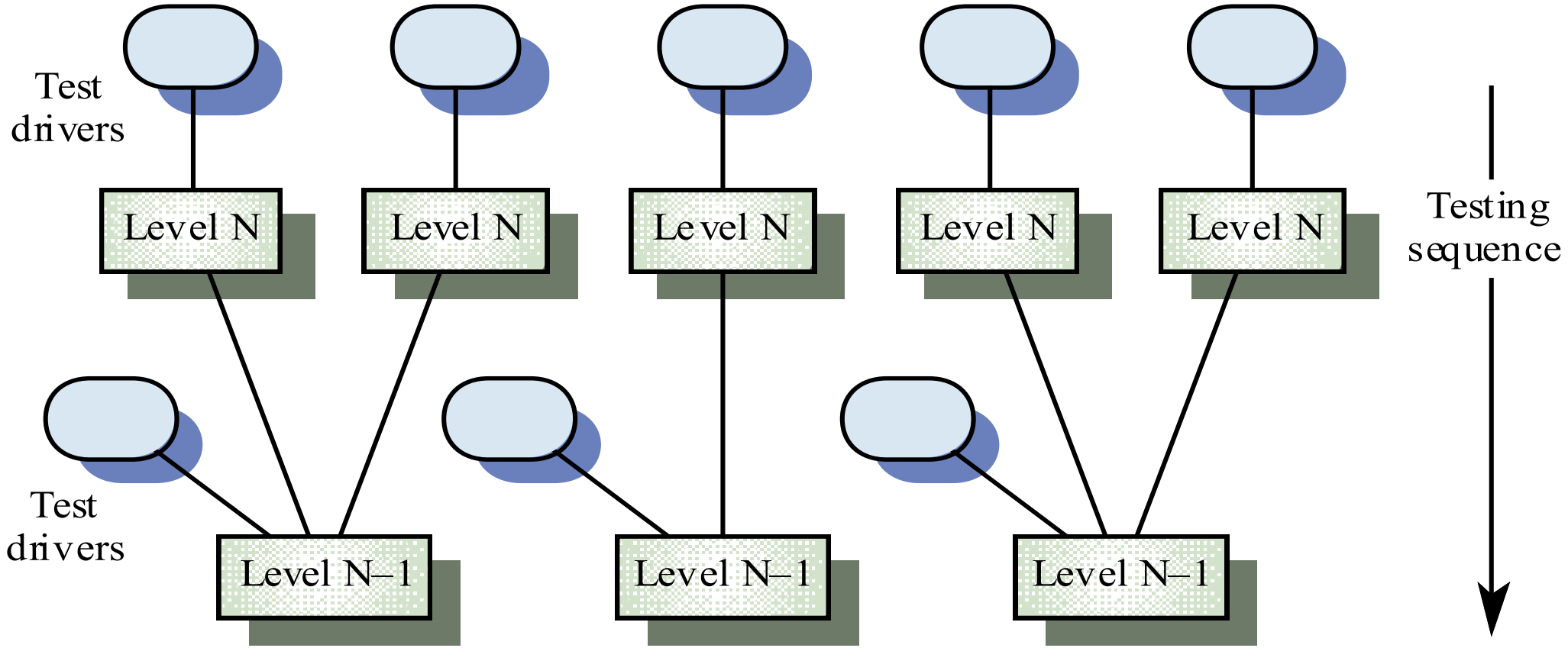
Test observation

- Extra codes are required to observe tests

Top-Down Approach [1]



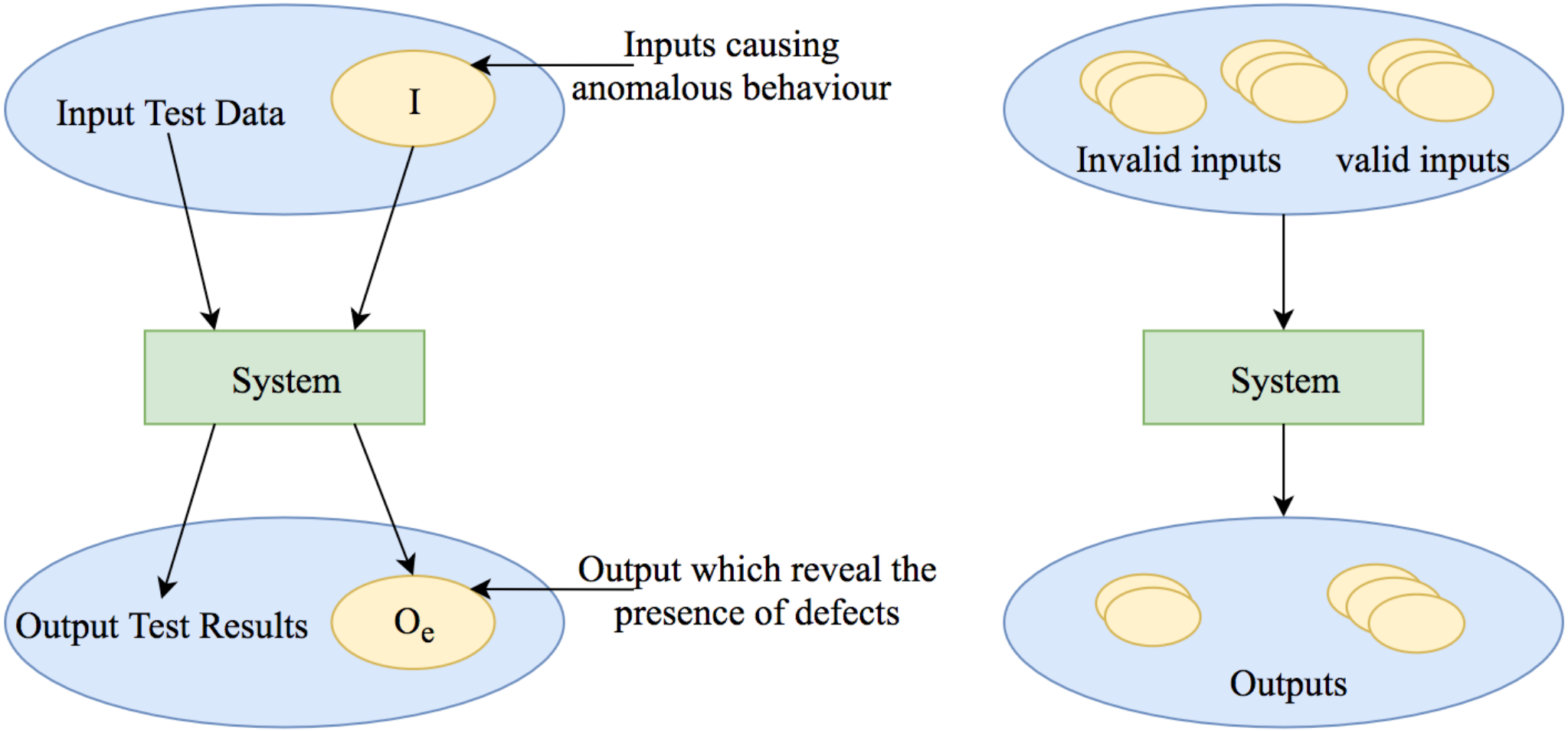
Bottom-Up Approach [2]



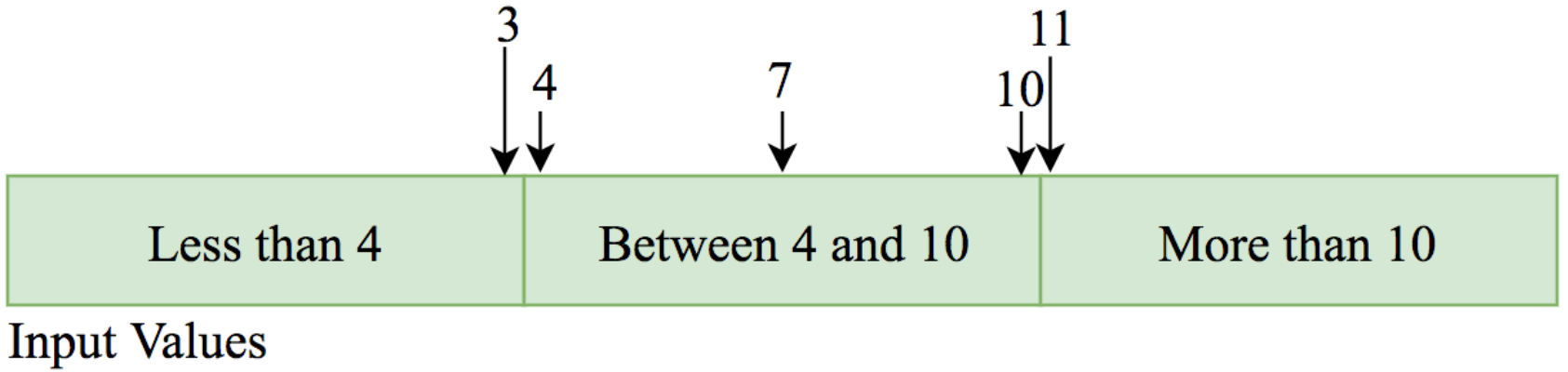
Release Testing

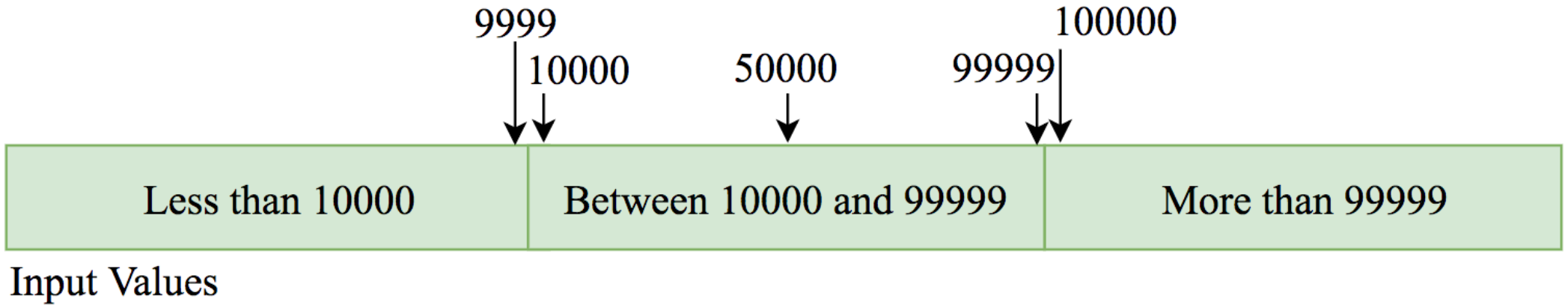
- Release Testing of a system that will be distributed to customers
- Primary goal is to increase the supplier's confidence that the system meets the requirements
- Release testing is usually black-box or functional testing
  - Based on the system specification only
  - Testers do not need to have knowledge of the system implementation
- Alpha Testing and Beta Testing

Black-Box Testing



Black-Box Testing





Test Table: Blackbox Testing

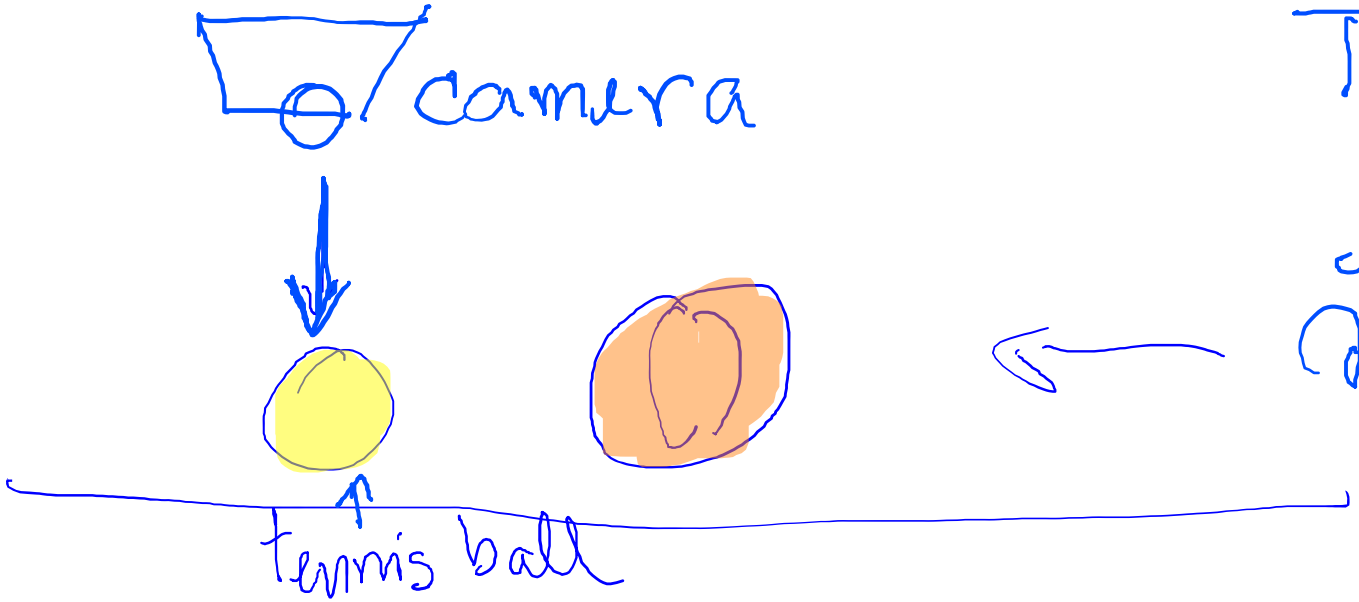
	Test Case	Test Data	Expected Result	Program Result	Note
Valid	1.Between 4 and 10	4, 7, 10	T	T	pass
Valid	2. <4	3,0,-1	F	T	Not pass
Valid	3. >10	11,50,100	F	F	pass
Invalid	4. Characters and Symbols	A, %, +	Reject	Reject	pass
Invalid	5. null		Reject	Reject	pass

$4 \leq x \leq 10 \rightarrow$ True

Invariant - match

	Test case	Descrip.	Test set	Expected Res.	Program Res.		
1	$4 \leq x \leq 10$		4, 10, 7	T	T	✓	valid case
2	$x < 4$		3, 2, -2	F	T, F	✗	
3	$x > 10$		11, 12, 100	F	F	✓	
4	x not int		1.01, 5.55	Invalid Inp.	T	✗	
5	x Any characters		a, *, -	in	Invalid Inp.	✓	

Detect tennis ball and count only tennis ball



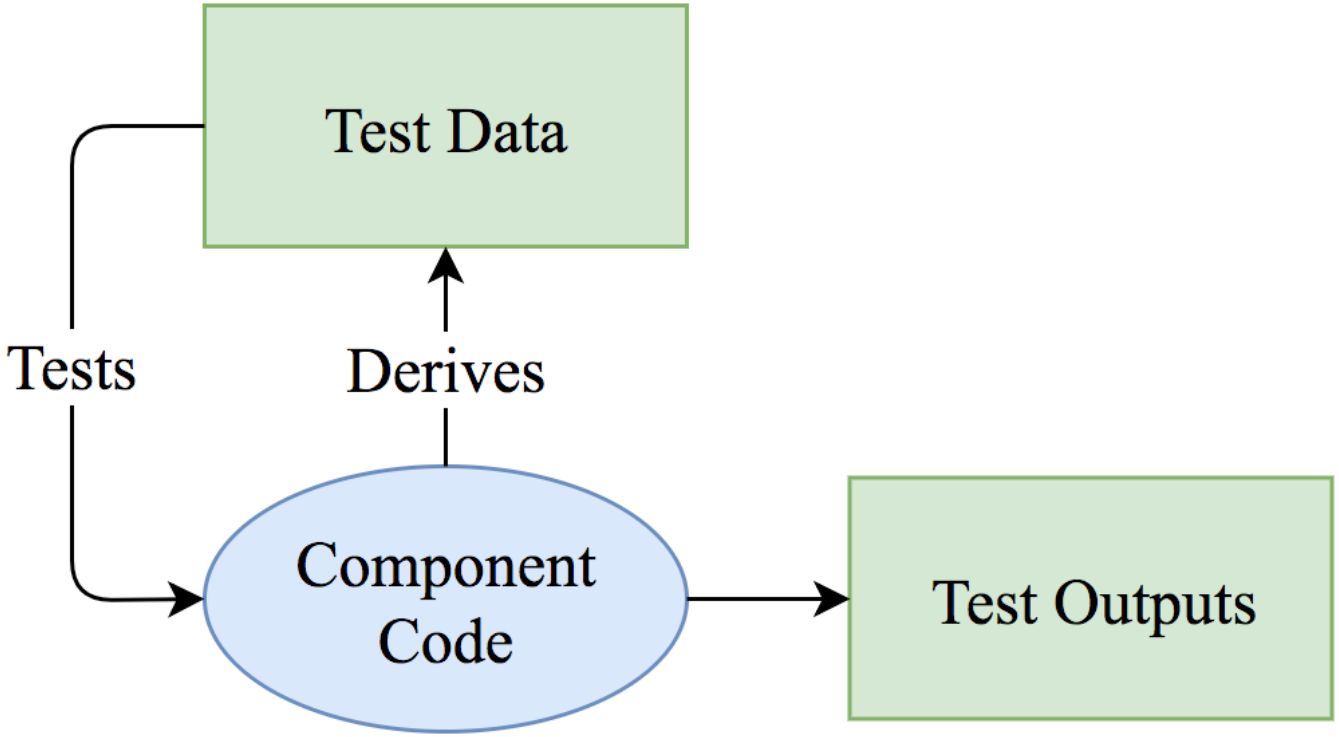
Tennis Ball → F

ลูกเทนนิส → F

Valid Input

Write a Test Plan

White-Box Testing



Test Table: White box testing

Program Path/ Test Case	Test Data	Expected Result	Program Result	Note
1. When i==0	0	‘yes’	‘no’	Not pass
2. When i !=0	1	‘no’	‘no’	pass

Use Cases

- To create a Use cases,
  - Identify tested operations
  - Design the required test cases
- From a sequence diagram, the inputs and outputs, created for the tests, can be identified

Performance Testing

- A part of release testing may involve testing the properties of a system, such as performance and reliability
- Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable

Stress Testing (a kind of performance testing)

- Exercise the system beyond its maximum design load
  - Stressing the system often causes defects to come
- Stressing the system test failure behaviour
  - Systems should not fail catastrophically
  - Stress testing checks for unacceptable lost of services or data
- Stress testing is particularly relevant to distributed systems
  - Exhibit severe degradation as a network becomes overloaded

Component Testing

- Component or unit testing is the process of testing individual components in isolation
- It is a defect testing process
- Types of Components:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality

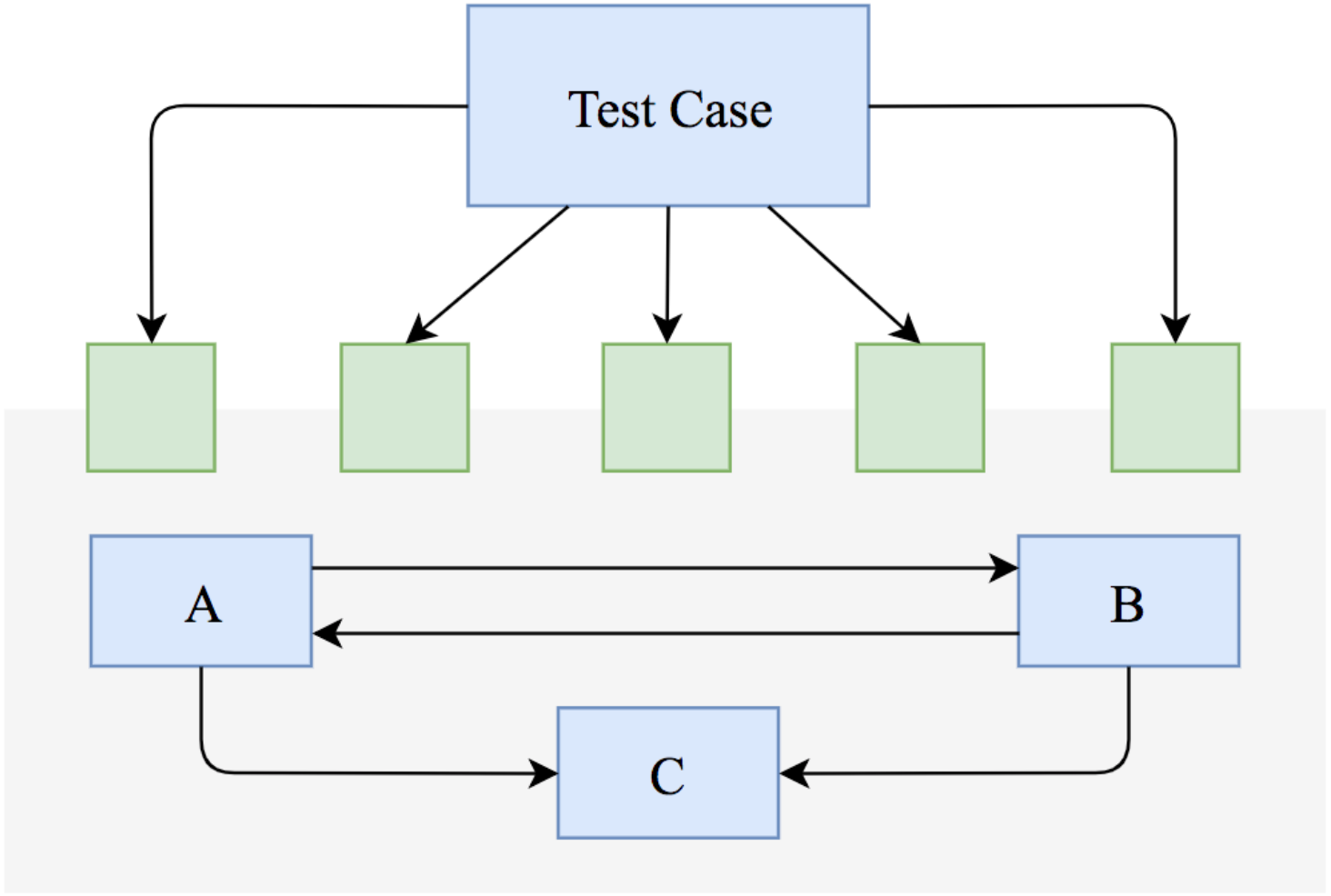
Object Class Testing

- Testing all operations associated with an object
- Setting and interrogating all object attributes
- Exercising the object in all possible states

Interface Testing

- Objectives are to detect faults due to interface errors or invalid assumptions about interfaces
- Importance for object-oriented development as objects are defined by their interfaces

Interface Testing (Cont.)



Interface Types

Parameter interfaces

- Data passed from one procedure to another

Shared memory interfaces

- Block of memory is shared between procedures or functions

Procedural interfaces

- Subsystem encapsulates a set of procedures to be called by other sub-systems

Message passing interfaces

- Subsystems request services from other sub-systems

Interface Errors

Interface misuse

- A calling component calls another component and makes an error in its use of its interface
- e.g. parameters in the wrong order

Interface misunderstanding

- A calling component embeds assumptions about the behaviour of the called component which are incorrect

Timing errors

- The called and the calling component operate at different speeds and out-of-date information is accessed

References

1. Ian Sommerville, Software Engineering 10th Edition, Addison-Wesley, April 2015.

Exercise

- Identify reusable parts from your example's system
- If you build a reuse software, what would you do for the web service module? Imagine a scenario where your web service can be reused by any client's programs. (Develop a web service module for reusing by other programs)
- If you build software by using reuse software as application service, what would you do for the graph editor client module? Imagine a scenario where your client program can reuse web service modules. (Develop the software that call the application service as a graph editor)

Software Licenses

1. Apache License
2. GNU License (Version 2 and 3)
3. BSD License
4. MIT License
5. Mozilla License
6. Microsoft Public Licenses (Ms-PL)
7. Creative Commons License

Software Licenses

1. What is your license
   - History
   - Description
   - Version
2. How to use your license
   - Condition/Constraint
   - Copy-left
3. Example of software using your license