Dasar Pengembangan Sistem Informasi

Implementation and Testing

Tim Pengampu Mata Kuliah

Topics Covered

- Implementation
- Testing
- Testing Stages
  - Development Testing
  - Release Testing
  - User Testing

Implementation

Implementation

- Implementation is a transformation process (*coding*) of software design models to a specific programming language code
  - Structured Programming - SP
  - Object Oriented Programming - OOP

Implementation

- Structural Design (SD):
  - Structural Programming (SP)
- Object Oriented Design (OOD):
  - Object Oriented Programming (OOP)

Programming Language

- Structured Programming :
  - C
  - Pascal
  - Basic, etc.
  - Javascript
- Object Oriented Programming :
  - C++
  - Java
  - C#, etc.

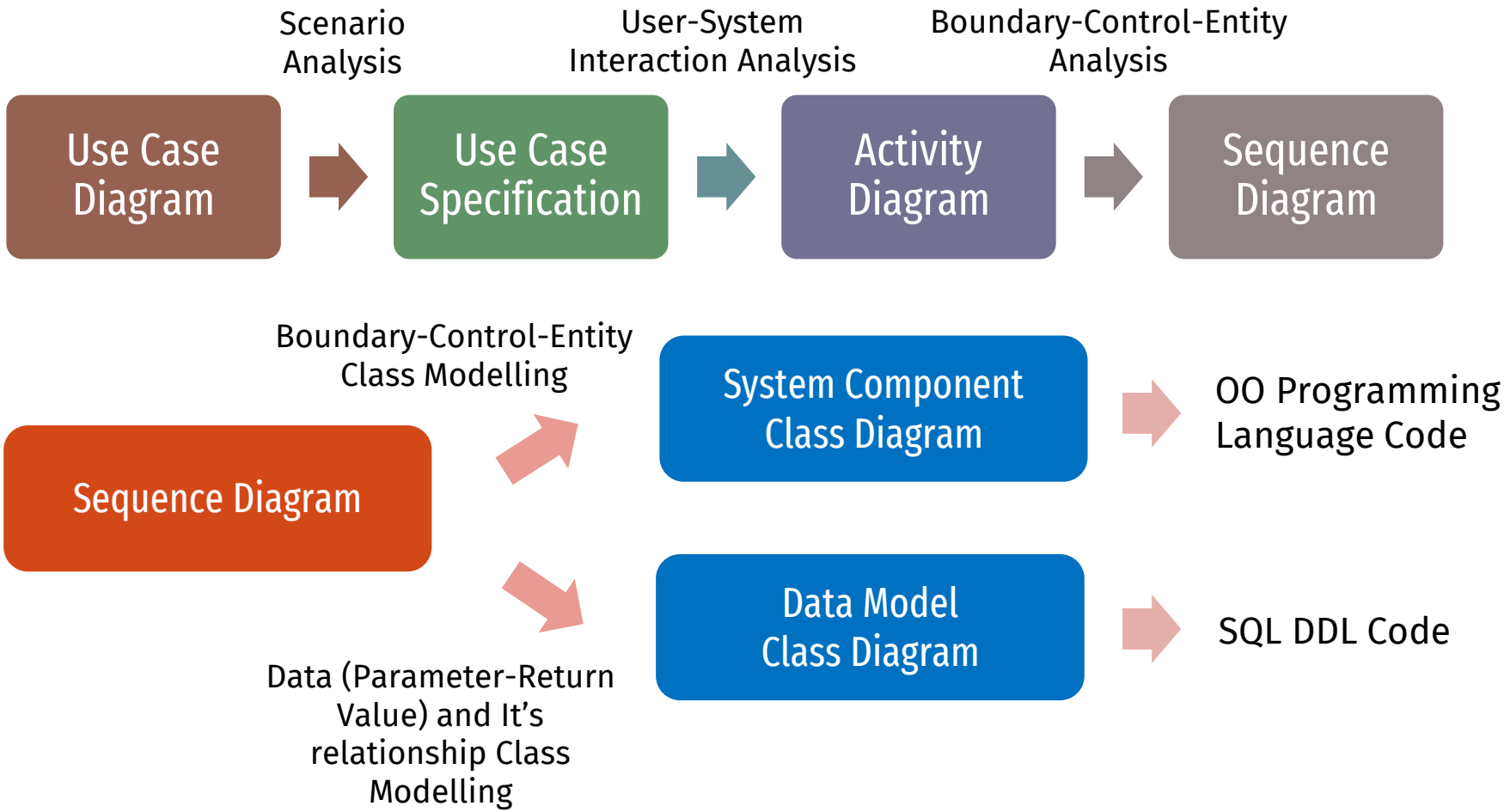
Structured Programming

- Data Flow Diagram, Process Specification
  - Realized into functional program code
- Physical Level ER Diagram and Data Dictionary
  - Realized into SQL Data Definition Language (DDL)
- State Transition Diagram and Control Specification
  - Realized into program code that control functional program code

Structured approach does not apply strict standard in its technical implementation

OOAD Flow

- OOD Detailing Flow

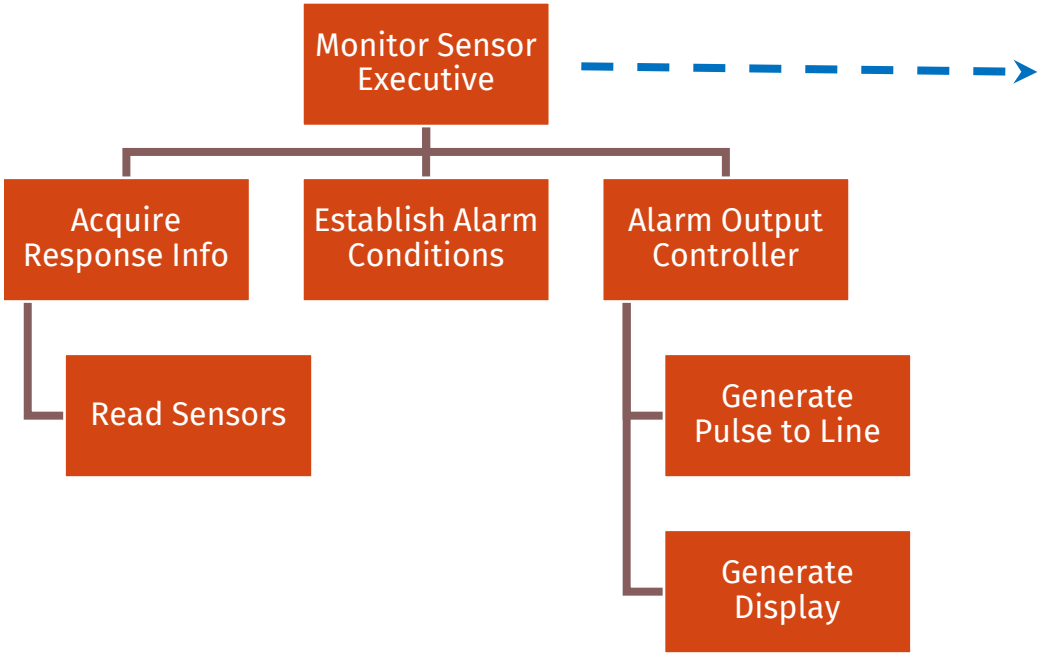


Improving Design and Implementation Quality Object Oriented Technique

FILKOM
FAKULTAS ILMU KOMPUTER

- Encapsulation
  - Data and functional/method abstraction
- Information hiding
  - Access management to a Class member: public, private, protected and friend
- Inheritance
  - Objects characteristics and behavior derivation
- Polymorphism
  - Interfacing between components or objects
  - Overriding, Overloading

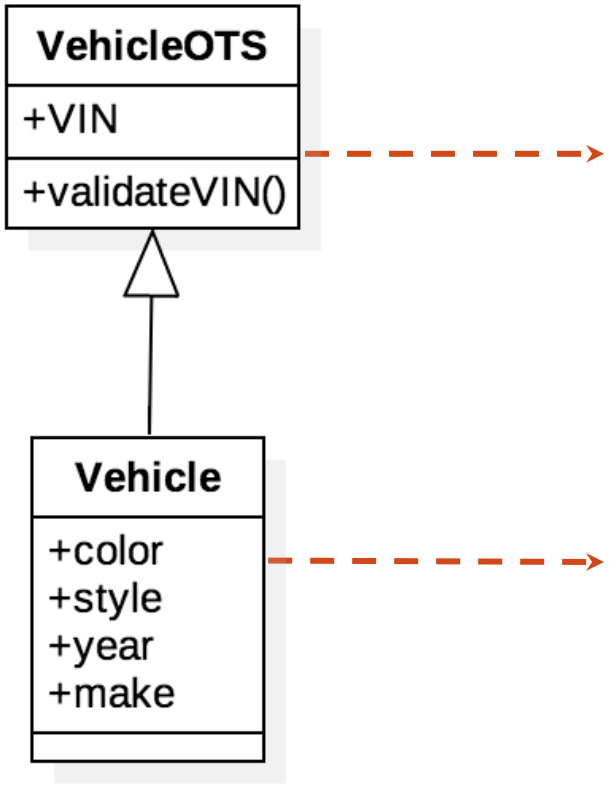
Structured Implementation




```
/* File MonitorSensorsExecutive.c */
# include MonitorSensorsExecutive.h;
void            ResetSensor (int IDSensor);
void            ResetAlarm (int IDAlarm);
.....
```

```
/* File MonitorSensorsExecutive.h */
# include .....
void            ResetSensor (int IDSensor);
void            ResetAlarm (int IDAlarm);
.....
```

Object Oriented Implementation




```
public class VehicleOTS
{
            public String VIN;
            public VehicleOTS( ) { }

            public void validateVIN (String vin)
            {
              // code here
            }
}


public class Vehicle extends VehicleOTS
{
            public String color;
            public String style;
            public Date           year;
            public String make;
            public Vehicle ( ) { }
}
```

Testing

Program testing

- Show that a program does what it is intended to do
- Discover program defects before it is put into use.
  - errors, anomalies program's non-functional attributes
- Reveal the presence of errors NOT their absence
- Executed using artificial data
- Testing is part of general verification and validation process
  - includes static validation techniques.

Program testing goals

- To demonstrate to the programmer and the customer that the software meets its requirements.
- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.

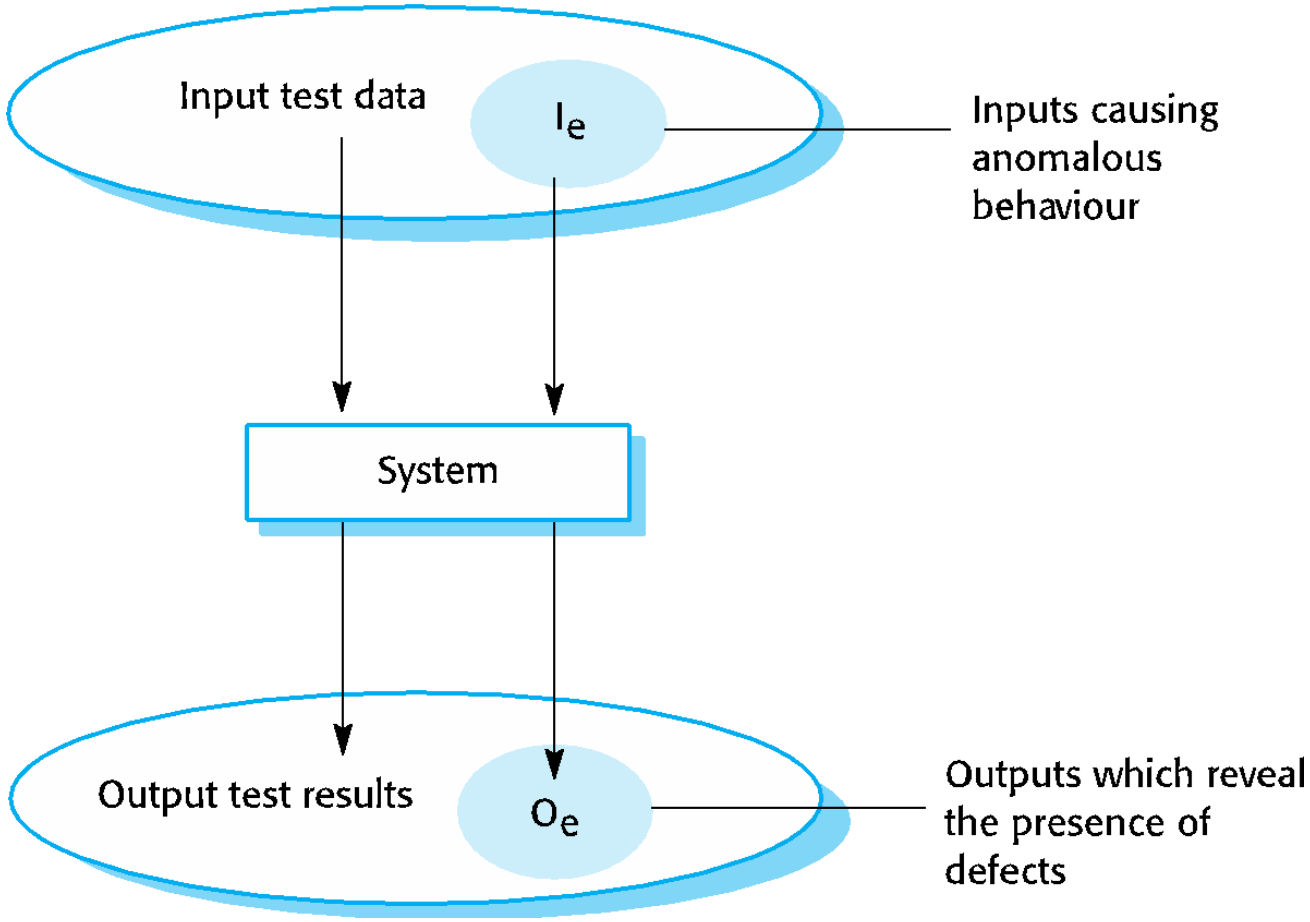
Validation and defect testing

- The first goal leads to validation testing
  - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use.
- The second goal leads to defect testing
  - The test cases are designed to expose defects and can be deliberately obscure and need not reflect how the system is normally used.

Testing process goals

- Validation testing
  - To demonstrate to the developer and the system customer that the software meets its requirements
  - A successful test shows that the system operates as intended.
- Defect testing
  - To discover faults or defects in the software where its behaviour is incorrect or not in conformance with its specification
  - A successful test is a test that makes the system perform incorrectly and so exposes a defect in the system.

An input-output model of program testing



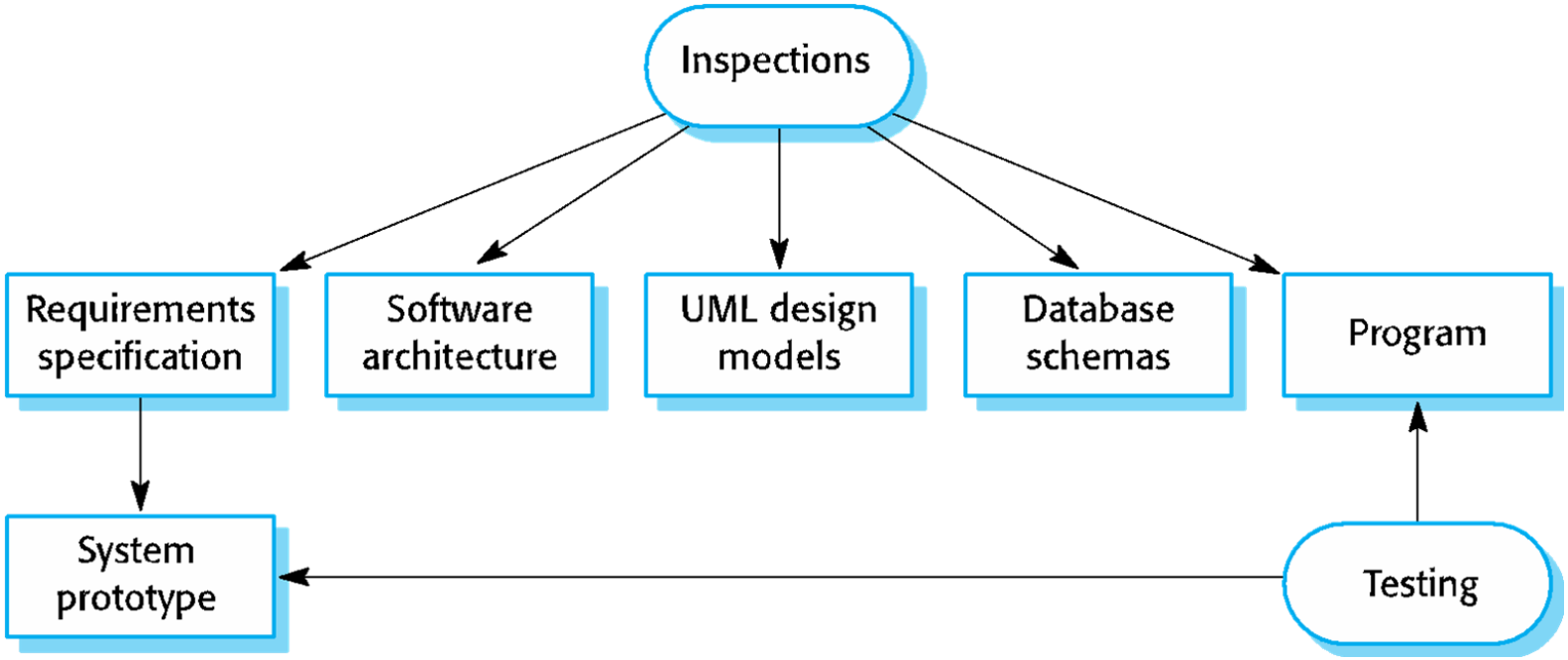
Verification and Validation

- 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.
- Aim of V&V is to establish confidence that the system is "fit for purpose".

V & V confidence

- Depends on system's purpose, user expectations and marketing environment
  - Software purpose
    - The level of confidence depends on how critical the software is to an organisation.
  - User expectations
    - Users may have low expectations of certain kinds of software.
  - Marketing environment
    - Getting a product to market early may be more important than finding defects in the program.

Inspections and Testing



Software inspections

- These involve people examining the source representation with the aim of discovering anomalies and defects.
- Inspections 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.

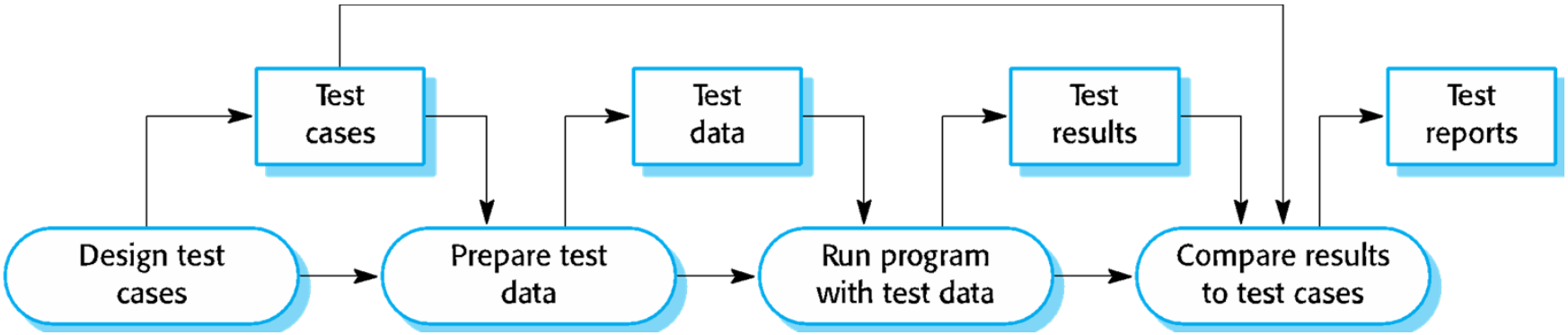
Advantages of inspections

- During testing, errors can mask (hide) other errors. Because inspection is a static process, you don't have to be concerned with interactions between errors.
- Incomplete versions of a system can be inspected without additional costs. If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
- As well as searching for program defects, an inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

Inspections and Testing

- Inspections and testing are complementary and not opposing verification techniques.
- Both should be used during the V & V process.
- 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.

A model of the software testing process



General Testing Guidelines

- Choose inputs that force the system to generate all error messages
- Design inputs that cause input buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated
- Force computation results to be too large or too small.

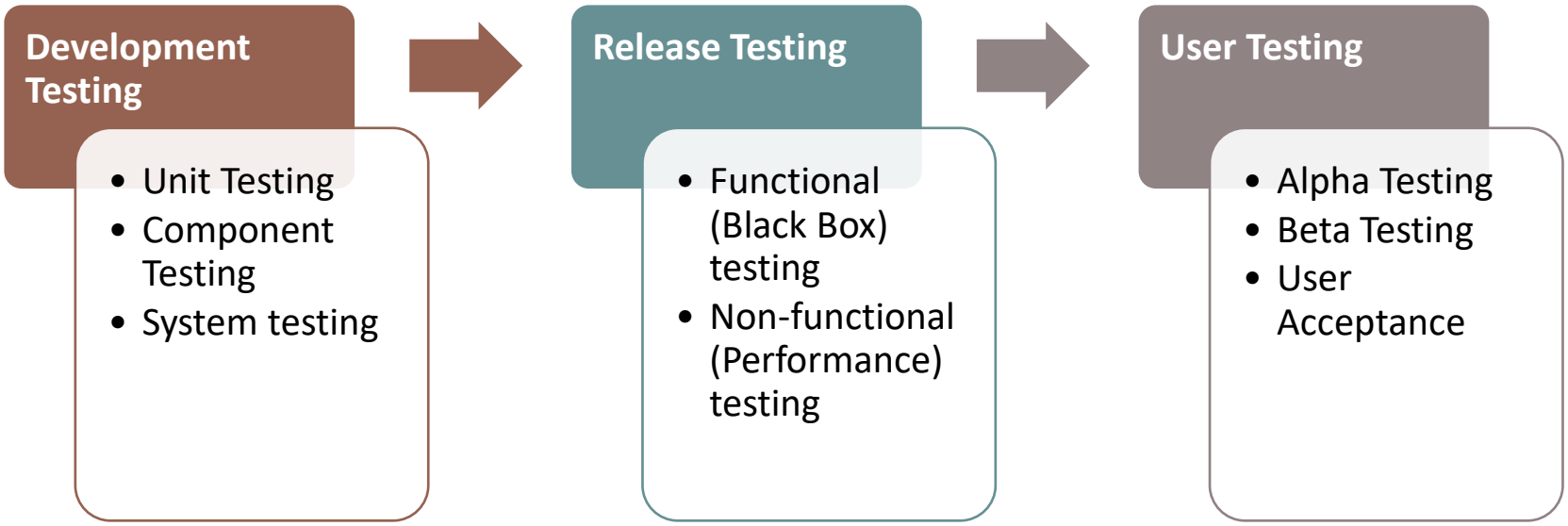
Automated testing

- Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
- In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.

Questions?

Testing Stages

Testing Stages



Based on design specification

By developer

Based on system specification

By developer, system functionality test

Based on user requirements and real data

By user and assisted by developer

Stages of Testing

- **Development testing**, where the system is tested during development to discover bugs and defects.
- **Release testing**, where a separate testing team test a complete version of the system before it is released to users.
- **User testing**, where users or potential users of a system test the system in their own environment and their own data.

FILKOM
FAKULTAS ILMU KOMPUTER

Development Testing

Development testing

- **Unit testing**, where individual program units or object classes are tested. Unit testing should focus on testing the functionality of objects or methods.
- **Component testing**, where several individual units are integrated to create composite components. Component testing should focus on testing component interfaces.
- **System testing**, where some or all of the components in a system are integrated and the system is tested as a whole. System testing should focus on testing component interactions.

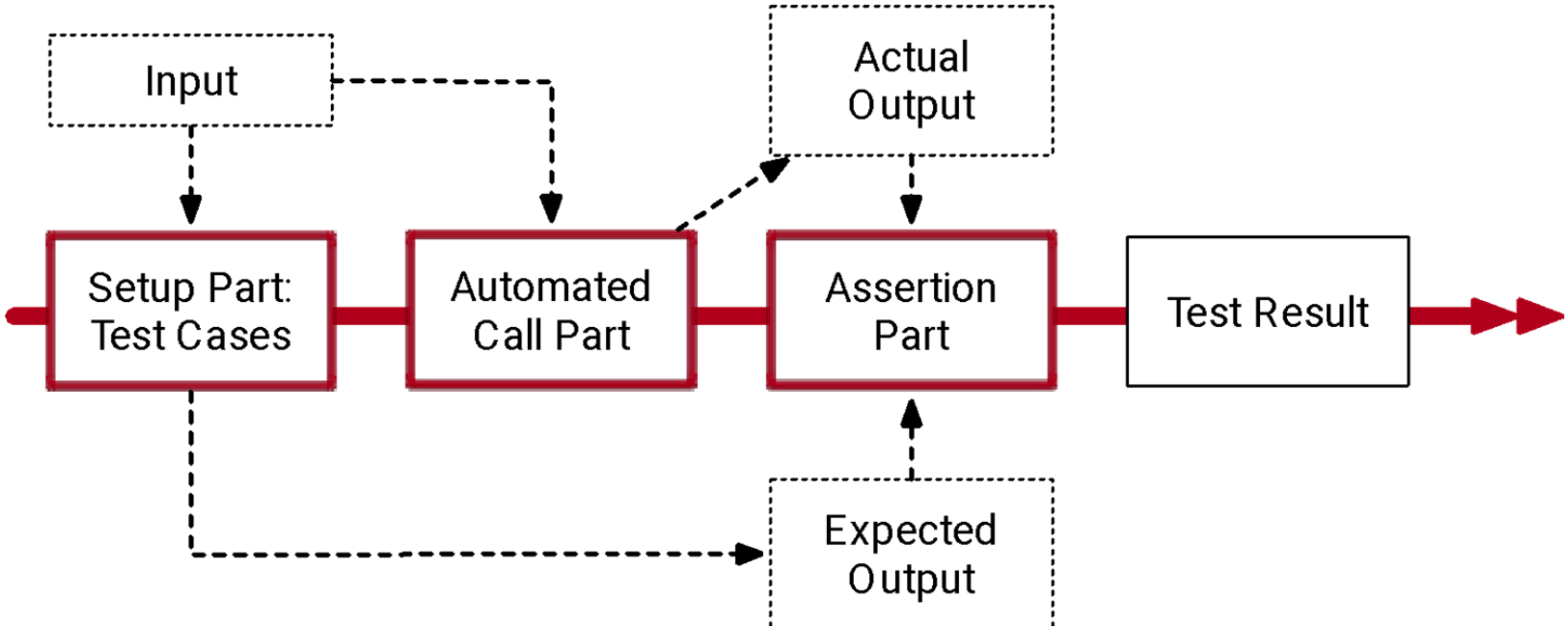
Unit testing

- **Unit testing** is the process of testing individual components in isolation.
- It is a defect testing process.
- Units may be:
  - Individual functions or methods within an object
  - Object classes with several attributes and methods
  - Composite components with defined interfaces used to access their functionality.

Types of Unit Test

- The test cases should show that, when used as expected, the component that you are testing does what it is supposed to do.
  - Should reflect normal operation of a program and should show that the component works as expected.
- If there are defects in the component, these should be revealed by test cases.
  - Based on testing experience of where common problems arise.
  - It should use abnormal inputs to check that these are properly processed and do not crash the component.

Automated Test Components



Automated Test Components

- A setup part, where you initialize the system with the test case, namely the inputs and expected outputs.
- A call part, where you call the object or method to be tested.
- An assertion part where you compare the result of the call with the expected result. If the assertion evaluates to true, the test has been successful--if false, then it has failed.

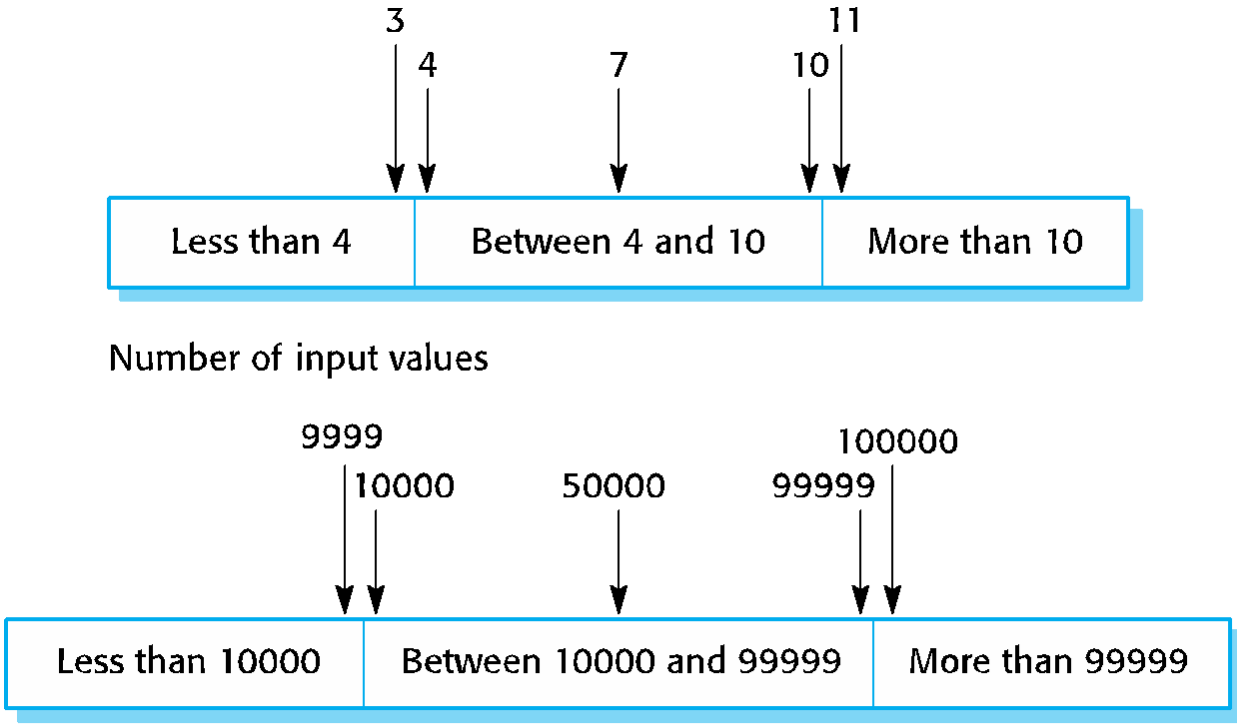
Testing strategies

- Partition testing, where you identify groups of inputs that have common characteristics and should be processed in the same way.
  - You should choose tests from within each of these groups.
- Guideline-based testing, where you use testing guidelines to choose test cases.
  - These guidelines reflect previous experience of the kinds of errors that programmers often make when developing components.

Partition Testing

- Input data and output results often fall into different classes where all members of a class are related.
- Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
- Test cases should be chosen from each partition.

Equivalence partitions



System and Component testing

- During system testing, reusable components that have been separately developed may be integrated with newly developed components.
  - The complete system is then tested.
- Components developed by different team members or sub-teams may be integrated at this stage.
- System testing is a collective rather than an individual process.
  - In some companies, system testing may involve a separate testing team with no involvement from designers and programmers.

Component testing

- Software components are often composite components that are made up of several interacting objects.
  - For example, in the weather station system, the reconfiguration component includes objects that deal with each aspect of the reconfiguration.
- You access the functionality of these objects through the defined component interface.
- Testing composite components should therefore focus on showing that the component interface behaves according to its specification.
  - You can assume that unit tests on the individual objects within the component have been completed.

System testing

- System testing during development involves integrating components to create a version of the system and then testing the integrated system.
- The focus in system testing is testing the interactions between components.
- System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.
- System testing tests the emergent behaviour of a system.

Key Points

- Testing can only show the presence of errors in a program. It cannot demonstrate that there are no remaining faults.
- Development testing is the responsibility of the software development team.
- A separate team should be responsible for testing a system before it is released to customers.
- Development testing includes:
  - Unit testing, in which you test individual objects and methods,
  - Component testing in which you test related groups of objects and system testing, in which you test partial or complete systems.

Release Testing

Release testing

- Release testing is the process of testing a particular release of a system that is intended for use outside of the development team.
- The primary goal of the release testing process is to show it is good enough for use.
  - Show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
- Release testing is usually a black-box testing process where tests are only derived from the system specification.

Release Testing and System Testing

- Release testing is a form of system testing.
- Important differences:
  - A separate team that has not been involved in the system development, should be responsible for release testing.
  - System testing by the development team should focus on discovering bugs in the system (defect testing).
  - The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

Performance Testing

- Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
- Tests should reflect the profile of use of the system.
  - Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
  - Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behavior.

Use-case Testing

- The use-cases developed to identify system interactions can be used as a basis for system testing.
- Each use case usually involves several system components so testing the use case forces these interactions to occur.
- The sequence diagrams associated with the use case documents the components and interactions that are being tested.

FILKOM
FAKULTAS ILMU KOMPUTER

User Testing

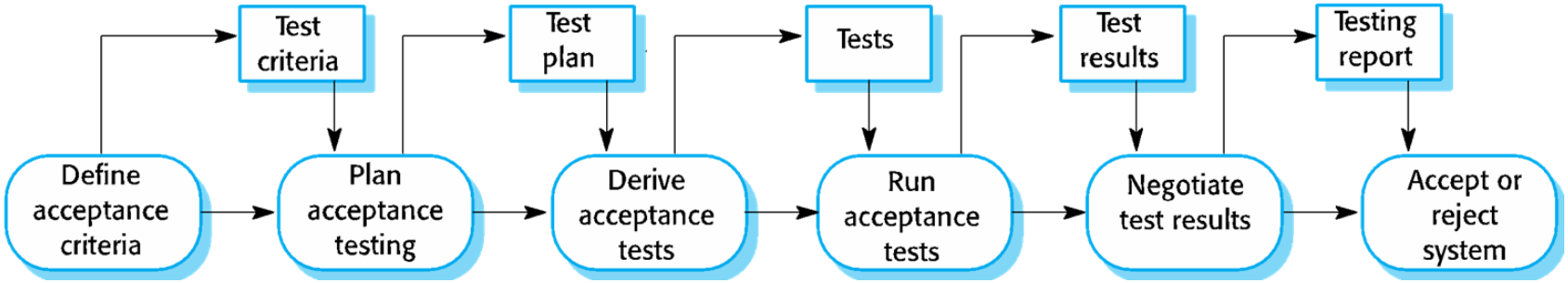
User Testing

- User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
- User testing is essential, even when comprehensive system and release testing have been carried out.
  - User's working environment have a major effect on the reliability, performance, usability and robustness of a system.
  - These cannot be replicated in a testing environment.

Types of User Testing

- Alpha testing
  - Users of the software work with the development team to test the software at the developer's site.
- Beta testing
  - A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
- Acceptance testing
  - Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment.

Acceptance Testing Process



1. Define acceptance criteria
2. Plan acceptance testing
3. Derive acceptance tests
4. Run acceptance tests
5. Negotiate test results
6. Reject/accept system

Key points

- When testing software, you should **try to 'break' the software** by using experience and guidelines to choose types of test case that have been effective in discovering defects in other systems.
- Wherever possible, you should **write automated tests**. The tests are embedded in a program that can be run every time a change is made to a system.

Key Points (2)

- Scenario testing involves inventing a typical usage scenario to derive test cases.
- Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operational environment.

Questions?