

Software Engineering

Prepared by Dr. Yasser Abdelhamid

IT 382
Software Engineering

Lecture 10



LECTURE OUTLINE

- Software testing
- Development testing
- Unit testing
- System testing
- Guidelines to choose types of test case
- Automated tests.
- Acceptance testing

PROGRAM TESTING

- Testing is intended:
 - To ensure that the program does what it is intended to do → (validation test)
 - to discover program defects before it is put into use → (defect test)
- When you test software, you execute a program using artificial data.
- Testing can reveal the presence of errors **NOT their absence**.
- Testing is **part** of a more general **verification and validation process**, which also includes static validation techniques.

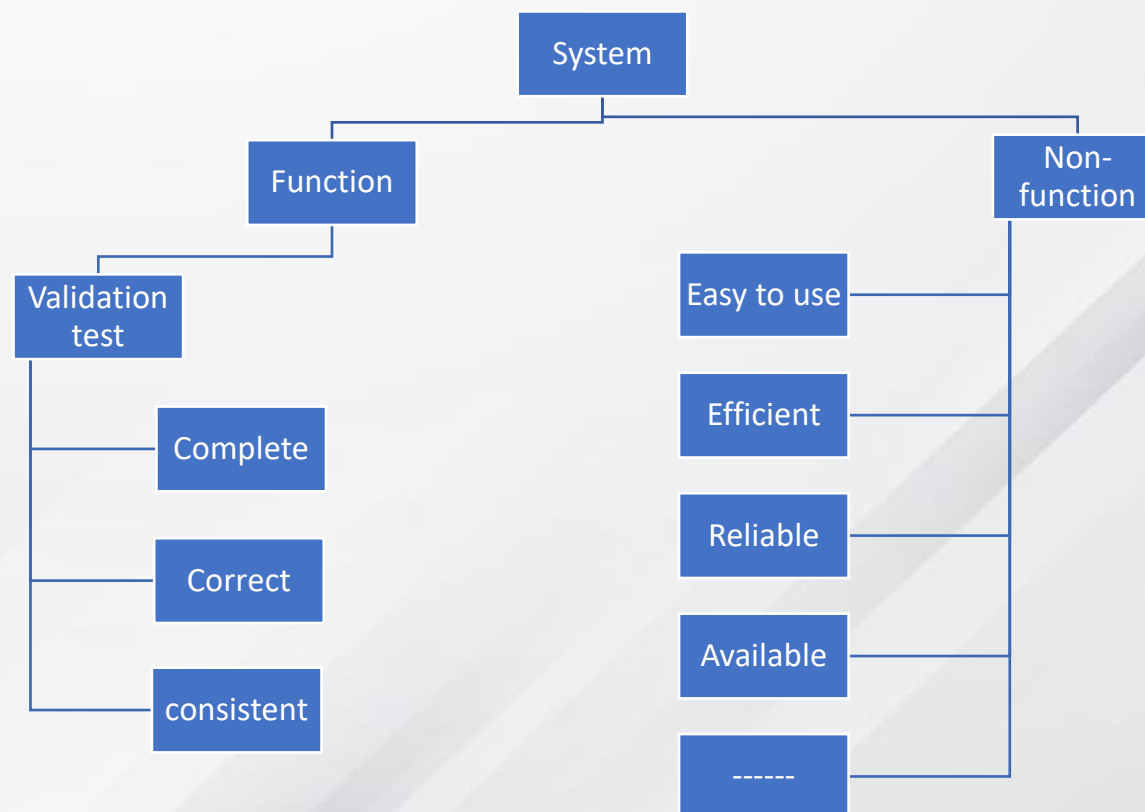
PROGRAM TESTING GOALS

- To demonstrate that the software meets its requirements (Validation test).
 - For **custom software**, this means that there should be at least one test for every requirement in the requirements document.
 - For **generic software** products, it means that there should be tests for all of the system features, plus combinations of these features, that will be incorporated in the product release.

PROGRAM TESTING GOALS

- To discover situations in which the behavior of the software is incorrect, undesirable or does not conform to its specification.
 - **Defect testing** is concerned with rooting out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations and data corruption.

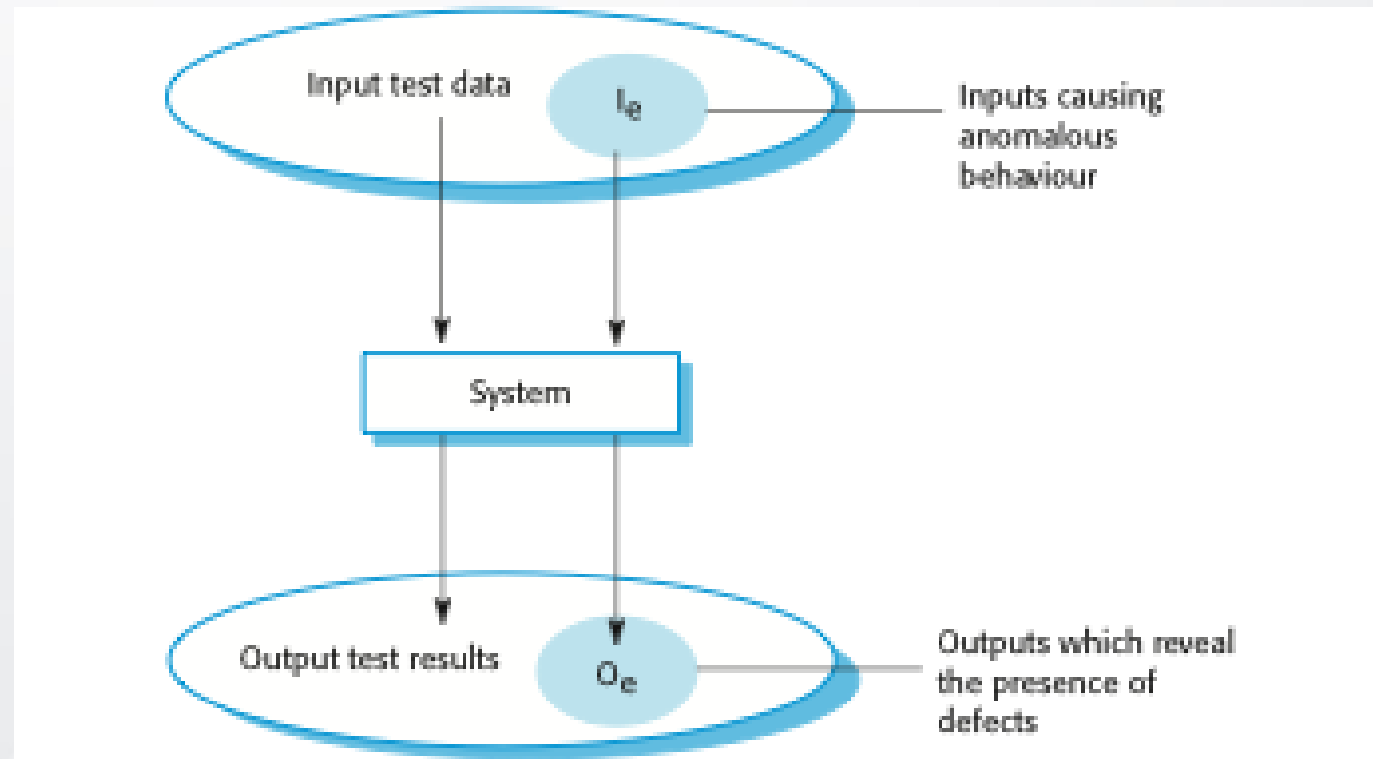
TESTING SHOULD MEET WITH REQUIREMENTS



VALIDATION AND DEFECT TESTING

- Validation testing
 - You expect the system to perform correctly using a given set of test cases that reflect the system's expected use (i.e., A successful test shows that the system operates as intended)
 - To demonstrate to the developer and the system customer that the software meets its requirements
- Defect testing
 - To discover faults or defects in the software where its behavior 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



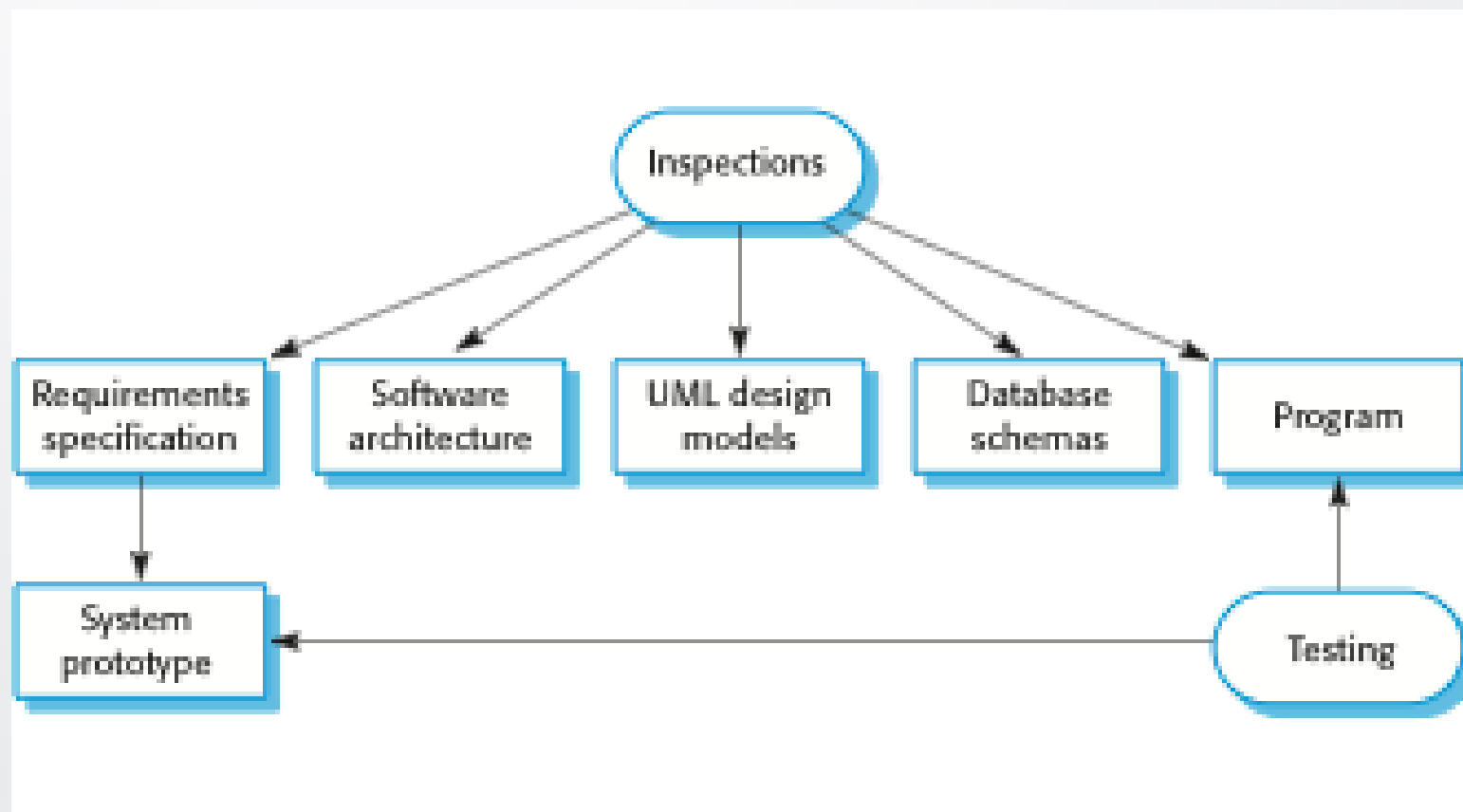
VERIFICATION VS VALIDATION

- 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.

INSPECTIONS AND TESTING

- Software inspections are concerned with analysis of the static system representation to discover problems (static verification)
 - May be supplement by tool-based document and code analysis.
- Software testing is concerned with exercising and observing product behaviour (dynamic verification)
 - The system is executed with test data and its operational behaviour is observed.

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**.

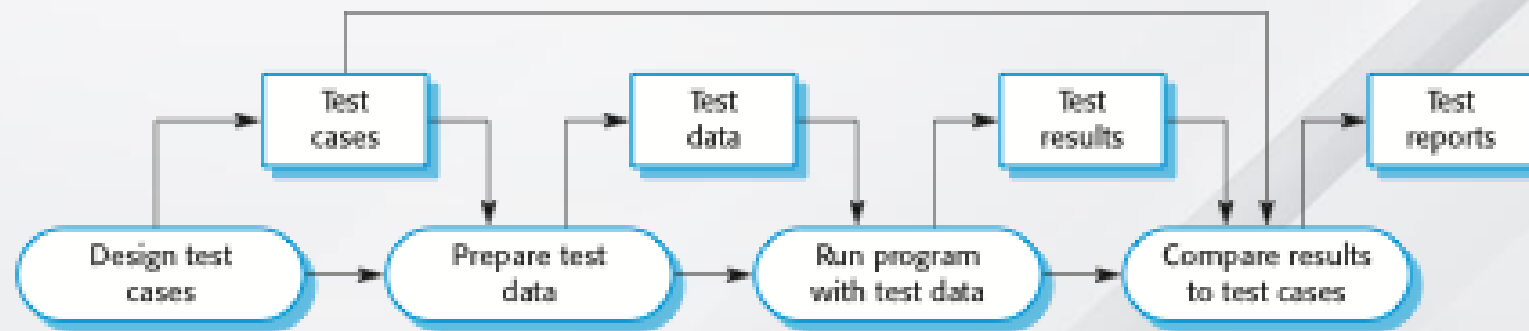
ADVANTAGES OF INSPECTIONS

- 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.
- 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



STAGES OF TESTING (WHEN)

- **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.

DEVELOPMENT TESTING

- Development testing includes all testing activities that are carried out by the team developing the system.
 - **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.

OBJECT CLASS TESTING

- Complete test coverage of a class involves
 - Testing all operations associated with an object
 - Checking all object attributes
 - Exercising the object in all possible states.
- Inheritance makes it more difficult to design object class tests as the information to be tested is not localised.

THE WEATHER STATION OBJECT INTERFACE

WeatherStation

identifier

reportWeather ()

reportStatus ()

powerSave (instruments)

remoteControl (commands)

reconfigure (commands)

restart (instruments)

shutdown (instruments)

WEATHER STATION TESTING

- Need to define test cases for “reportWeather”, calibrate, test, startup and shutdown.
- Using a state model, identify sequences of state transitions to be tested and the event sequences to cause these transitions
- For example:
 - Shutdown -> Running-> Shutdown
 - Configuring-> Running-> Testing -> Transmitting -> Running
 - Running-> Collecting-> Running-> Summarizing -> Transmitting -> Running

UNIT TEST EFFECTIVENESS

- The test cases should show that the component that you are testing **does what it is supposed to do**.
- If there are **defects** in the component, these should be revealed by test cases.
- This leads to 2 types of unit test case:
 - The first of these should reflect **normal operation** of a program and should show that the component works as expected.
 - The other kind of test case should be 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.

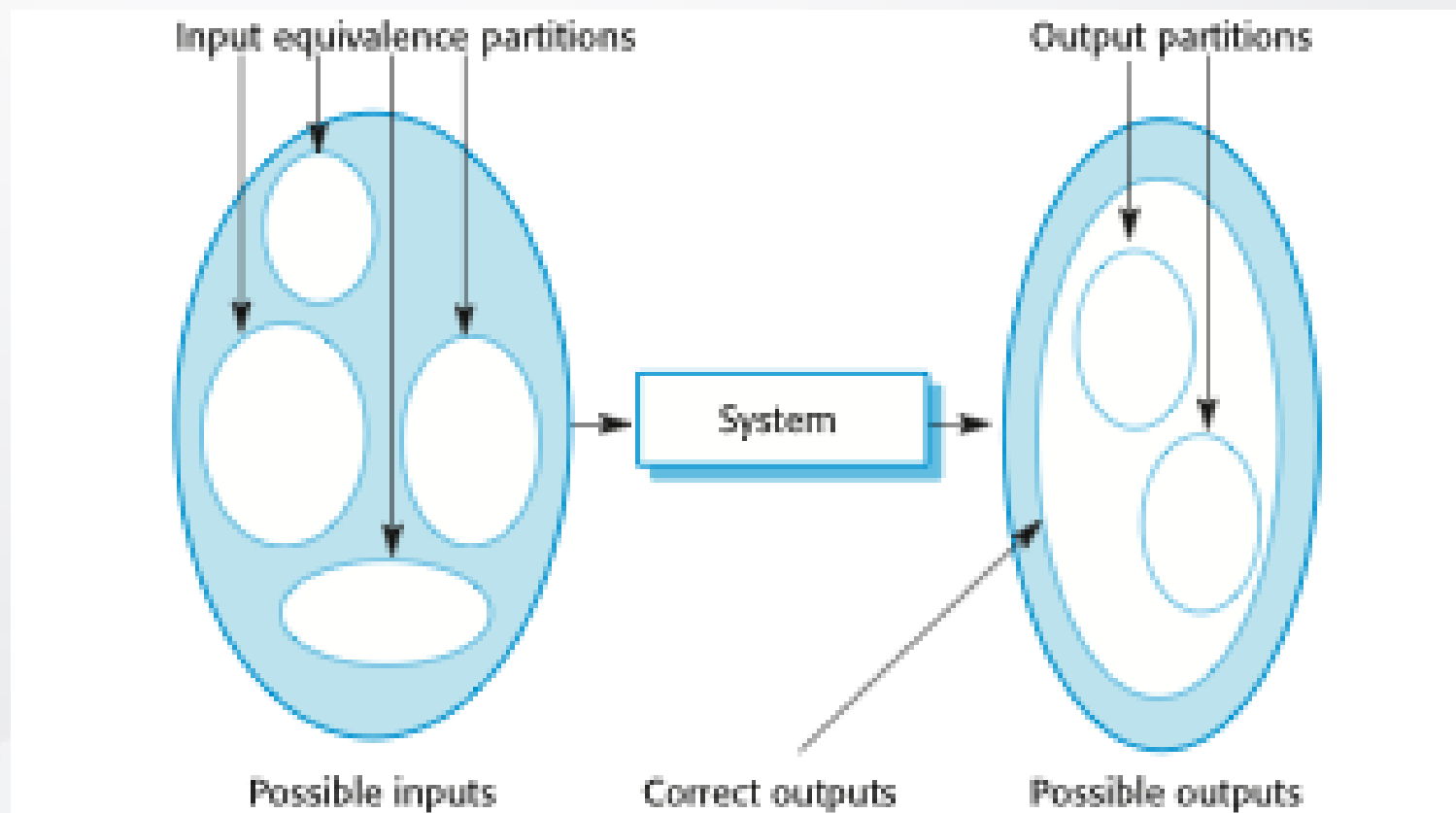
TESTING STRATEGIES

- 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.**

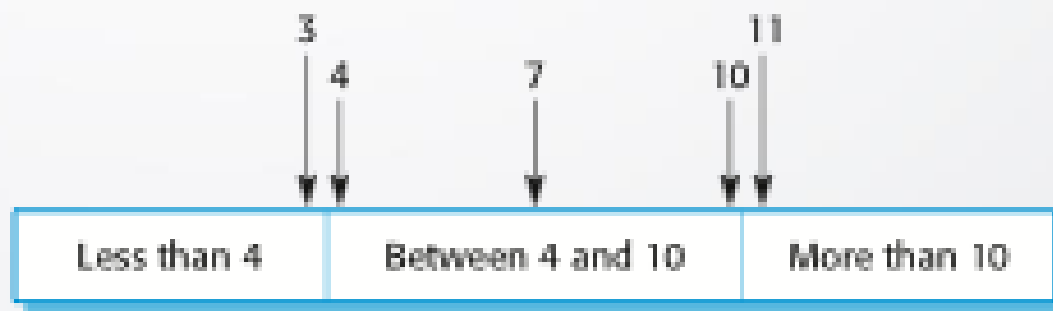
PARTITION TESTING

- 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 PARTITIONING



EQUIVALENCE PARTITIONS



Number of input values



Input values

TESTING GUIDELINES (SEQUENCES)

- Test with sequences of zero length
- Test software with sequences which have only a single value.
- Use sequences of different sizes in different tests.
- Derive tests so that the first, middle and last elements of the sequence are accessed.

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.

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.

INTERFACE TESTING FOR COMPONENT TESTING

✧ Objectives are to detect faults due to:

- 1. Interface errors
- 2. Invalid assumptions about interfaces.

✧ Interface types

- **Parameter interfaces** Data passed from one method or procedure to another.
- **Shared memory interfaces** Block of memory is shared between procedures or functions.
- **Procedural interfaces** Sub-system encapsulates a set of procedures to be called by other sub-systems.
- **Message passing interfaces** Sub-systems request services from other sub-systems

POSSIBLE 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 **behavior** 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.

INTERFACE TESTING GUIDELINES

- ✧ Design tests so that parameters to a called procedure are at the extreme ends of their ranges.
- ✧ Always test pointer parameters with null pointers.
- ✧ Design tests which cause the component to fail.
- ✧ Use stress testing in message passing systems.
- ✧ In shared memory systems, vary the order in which components are activated.

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:
 - 1. components are **compatible**,
 - 2. **interact correctly** and
 - 3. transfer the **right data** at the **right time** across their interfaces.
- ✧ System testing tests the emergent behavior of a system.

USE-CASE TESTING FOR SYSTEM 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.

TESTING POLICIES

✧ **Exhaustive system testing is impossible** so testing policies which define the required system test coverage may be developed.

✧ Examples of testing policies:

- All system functions that are accessed through **menus** should be tested.
- **Combinations of functions** (e.g. text formatting) that are accessed through the same menu must be tested.
- Where user input is provided, all functions must be tested with both **correct and incorrect input**.

REGRESSION TESTING

- ✧ Regression testing is testing the system to check that changes have not 'broken' **previously working code**.
- ✧ In a manual testing process, regression testing is expensive but, with automated testing, it is simple and straightforward. All tests are rerun every time a change is made to the program.
- ✧ Tests must run 'successfully' before the change is committed.

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 convince the supplier of the system that it is good enough for use.
- ✧ Release testing, therefore, must show that the system **delivers its specified functionality, performance and dependability**, and that it **does not fail** during normal use.

RELEASE TESTING AND SYSTEM TESTING

- Release testing is a form of system testing.
- Important differences:
 - **release testing**
 - Done by: **Separate team** that has not been involved in the system development.
 - Objective: **Validation** testing
 - **System testing:**
 - Done by: **Development team**
 - Objective: **Defect 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.
 - The reason for this is that influences from the **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.
Primarily for custom systems.

LATEST ADVANCEMENTS IN SOFTWARE ENGINEERING

THE AI-ENGINEERED SOFTWARE LIFECYCLE

- AI is moving beyond simple code completion to become a collaborative partner in the entire development process.
- Developers are shifting from **writing** boilerplate code to **reviewing, verifying, and integrating** AI-generated code.

AI IN REQUIREMENTS & DESIGN

- **Tools like ChatGPT:** Can help draft user stories, technical specifications, and even architectural diagrams from natural language prompts.

GENERATIVE AI (GENAI) FOR CODING

- **AI Assistants (e.g., GitHub Copilot, Gemini Code Assist, Amazon Q)** are now integral.
- They perform **real-time code generation, refactoring, and documentation.**
- **GitHub Copilot, Amazon CodeWhisperer, etc.:** Act as pair programmers, suggesting whole lines and functions.

AI IN TESTING & DEBUGGING

- **Automated Test Case Generation:**
 - AI analyzes code to create comprehensive test suites.
- **Intelligent Root Cause Analysis:**
 - AI inspects logs and metrics to pinpoint the likely cause of a failure.
- AI-powered tools (e.g., **Testim, AppliTools**) automate **visual testing** and **test script generation**, reducing test fragility and maintenance time.

AI AGENTS & SWE-BENCH

- AI systems are increasingly capable of autonomously tackling complex, multi-step tasks (e.g., fixing a bug across a large codebase) as demonstrated by performance improvements on benchmarks like **SWE-Bench**.

SUMMARY

- 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.
- 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.
- 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.
- Latest advancements in Software Engineering



Thank You

