Software Testing

Basics

Important

1. Introduction to Software Testing

Testing is intended to show that a program does what it is intended to do and to discover program defects before it is put into use. Testing is aimed at ensuring that a program:

1. **Does what it is intended to do** (validation)
2. **Discovers program defects** before it is put into use (defect testing)

Testing involves executing the software with artificial data, checking the results for errors, anomalies, or information about non-functional attributes.

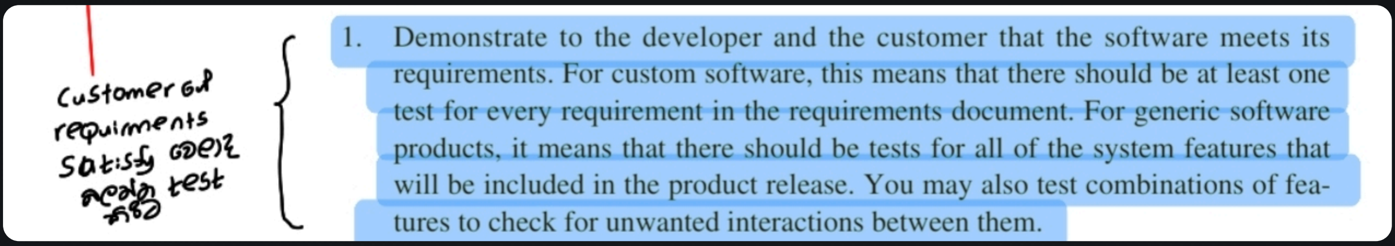
"Testing can only show the presence of errors, not their absence†"

2. Objectives of Software Testing

- **Demonstrate to the developer and customer that the software meets its requirements** (Validation Testing)
- **Find situations where the software behaves incorrectly, undesirably, or does not conform to its specification** (Defect Testing)

3. Types of Testing

3.1 Validation Testing

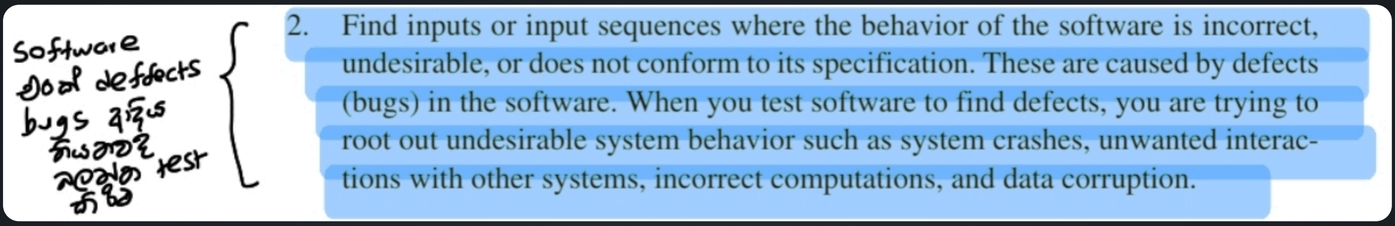


Customer ගේ requirements satisfy වෙනවද බලන්න test කිරීම

1. Demonstrate to the developer and the customer that the software meets its requirements. 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 that will be included in the product release. You may also test combinations of features to check for unwanted interactions between them.

- **Purpose:** Show the system meets its requirements using test cases that reflect expected use.
- **Test Case Design:** Based on normal, expected inputs.
- **Outcome:** Demonstrates correct operation for intended scenarios.
- **Example:** For every requirement, there should be at least one corresponding test case1.

3.2 Defect Testing

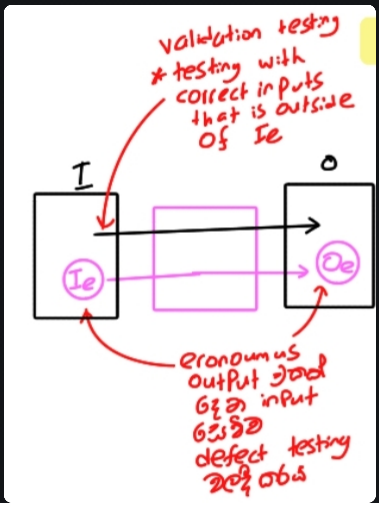


Software එකේ deffects bugs ආදිය තියෙනවද බලන්න test කිරීම

2. Find inputs or input sequences where the behavior of the software is incorrect, undesirable, or does not conform to its specification. These are caused by defects (bugs) in the software. When you test software to find defects, you are trying to root out undesirable system behavior such as system crashes, unwanted interactions with other systems, incorrect computations, and data corruption.

- **Purpose:** Expose faults or bugs using test cases that may be obscure or abnormal.
- **Test Case Design:** Based on unusual, boundary, or invalid inputs.
- **Outcome:** Reveals incorrect, undesirable, or unspecified behavior.

- **Example:** Tests that cause system crashes, incorrect computations, or data corruption1.


validation testing
* testing with correct inputs that is outside of Ie
I
O
Ie
Oe
eronoumus output එකක් දෙන input හොයන defect testing වලදී [illegible]

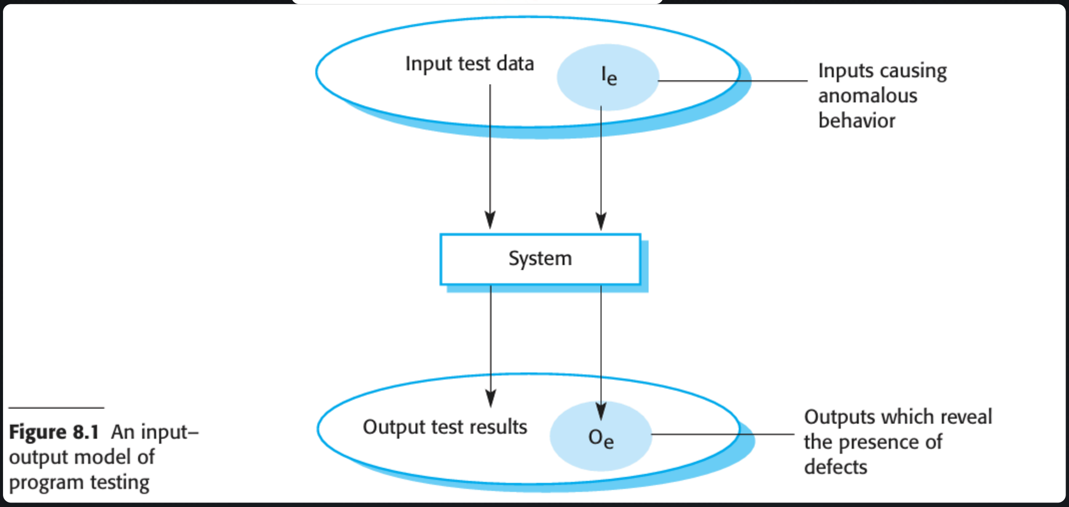

Input test data
I_e
Inputs causing anomalous behavior
System
Output test results
O_e
Outputs which reveal the presence of defects

Figure 8.1 An input–output model of program testing

Note: There is no strict boundary between validation and defect testing. Validation testing may reveal defects, and defect testing may show the system meets requirements.

4. Verification and Validation (V&V)

4.1 Definitions

Term	Question Answered	Focus	Methods
Verification	Are we building the product right?	accordance to specification	Reviews, inspections, static analysis
Validation	Are we building the right product?	Meeting user/customer needs	Dynamic testing, user acceptance tests

- **Verification:** *Static process*; <u>checks if the software meets its stated requirements and design specifications</u>(checking that the software meets its stated functional and non-functional requirements). Involves activities like inspections and reviews, and does not require code execution.

software inspections and reviews

- As well as software testing, the verification and validation process may involve software inspections and reviews.
- Inspections and reviews analyze and check the system requirements, design models, the program source code, and even proposed system tests.
- <u>These are "static" V & V techniques in which you don't need to execute the software to verify it.</u> -> Verification technique
- Figure 8.2 shows that software inspections and testing sup port V & V at different stages in the software process. The arrows indicate the stages in the process where the techniques may be used.

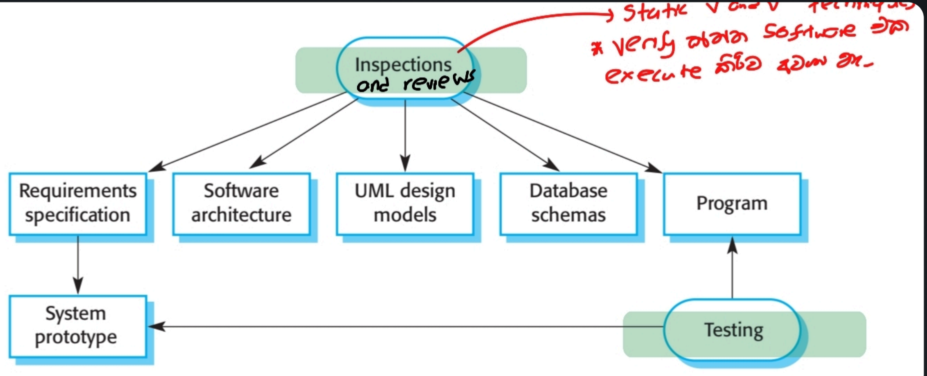
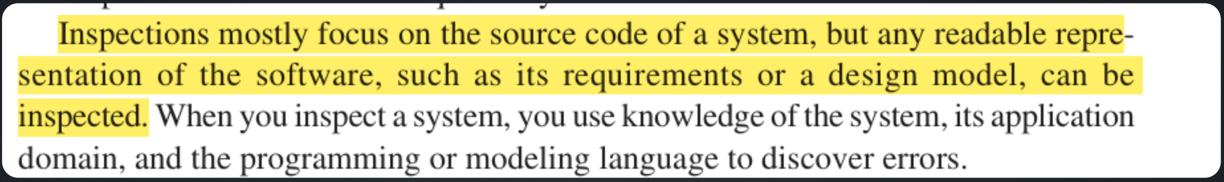


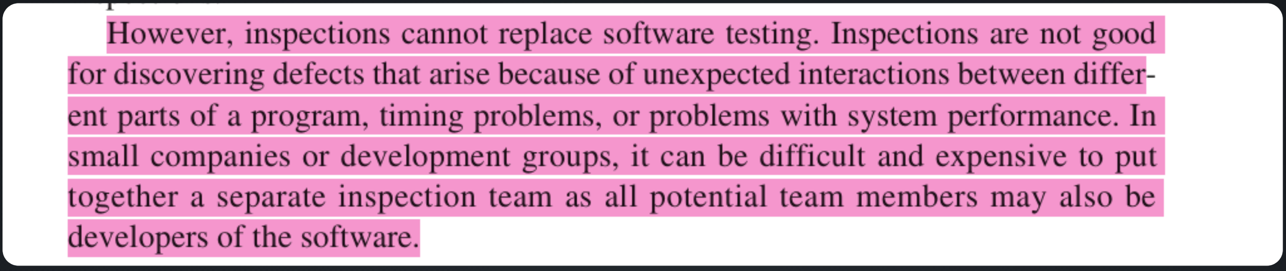
Figure 8.2 Inspections and testing


Inspections mostly focus on the source code of a system, but any readable representation of the software, such as its requirements or a design model, can be inspected. When you inspect a system, you use knowledge of the system, its application domain, and the programming or modeling language to discover errors.

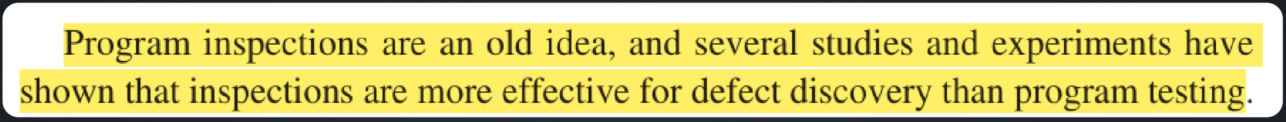
Advantages of software inspection over dynamic testing

They complement dynamic testing and offer unique benefits in defect detection and quality assurance.

1. **No Error Masking**
   - Testing can obscure errors: An initial error may produce unexpected outputs, making it difficult to distinguish subsequent errors from side effects.
   - Inspections avoid this by analyzing code/documentation statically, enabling simultaneous discovery of multiple defects in a single session.
2. **Cost-Effective for Incomplete Systems**
   - Inspections require no test harnesses or executable code, reducing overhead for partially developed systems.
   - Testing incomplete systems often demands additional infrastructure (e.g., stubs, drivers), increasing development costs.
3. **Broader Quality Assessment**
   - Inspections evaluate compliance with standards, portability, maintainability, and algorithmic efficiency.
   - They identify poor programming practices (e.g., inefficient algorithms, unclear documentation) that testing alone cannot address



However, inspections cannot replace software testing. Inspections are not good for discovering defects that arise because of unexpected interactions between different parts of a program, timing problems, or problems with system performance. In small companies or development groups, it can be difficult and expensive to put together a separate inspection team as all potential team members may also be developers of the software.

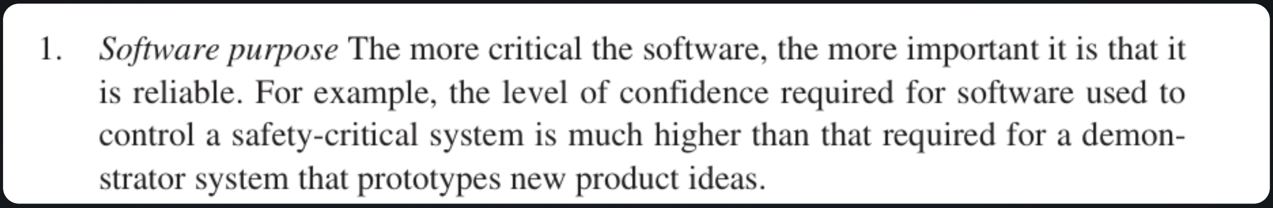


Program inspections are an old idea, and several studies and experiments have shown that inspections are more effective for defect discovery than program testing.

- **Validation:** *Dynamic process*; ensures the software meets the customer's expectations and actual needs, going beyond the specification. Involves executing the software and checking its behavior. Validation is a more general process. Validation is essential because, as I discussed in Chapter 4, statements of requirements do not always reflect the real wishes or needs of system customers and users.

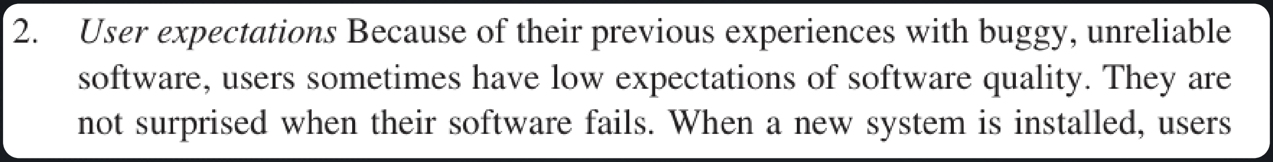
4.2 goals of V&V

- Both V&V processes aim to establish confidence that the software is **fit for purpose**.
- The required level of confidence depends on:
  - **Software purpose** (e.g., safety-critical vs. prototype)

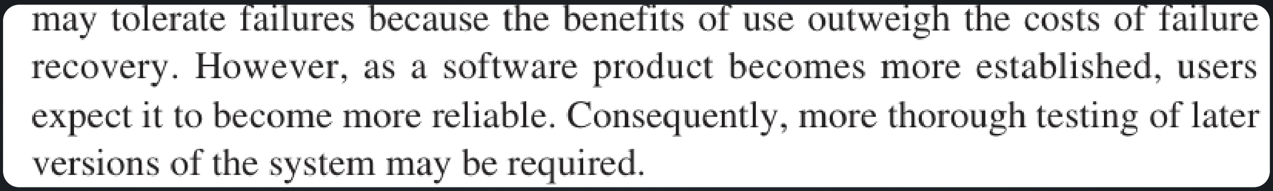


1. *Software purpose* The more critical the software, the more important it is that it is reliable. For example, the level of confidence required for software used to control a safety-critical system is much higher than that required for a demonstrator system that prototypes new product ideas.

  - **User expectations**

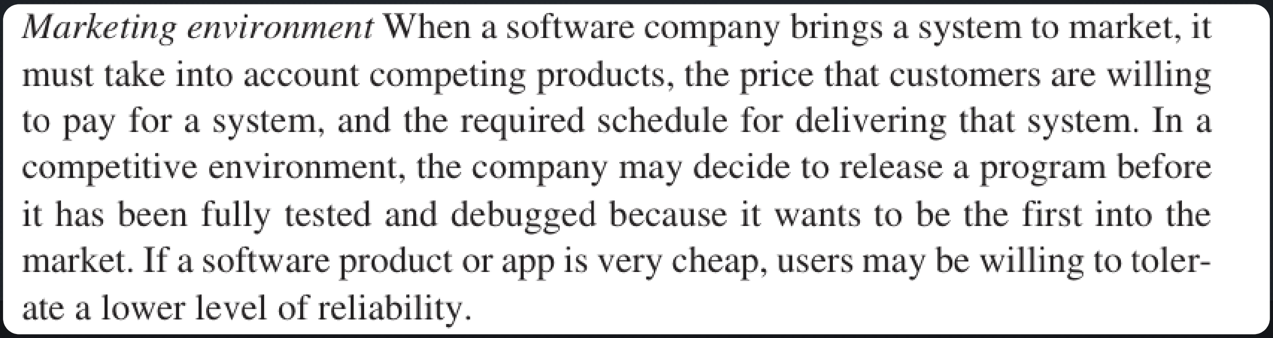


2. *User expectations* Because of their previous experiences with buggy, unreliable software, users sometimes have low expectations of software quality. They are not surprised when their software fails. When a new system is installed, users



may tolerate failures because the benefits of use outweigh the costs of failure recovery. However, as a software product becomes more established, users expect it to become more reliable. Consequently, more thorough testing of later versions of the system may be required.

  - **Market environment**



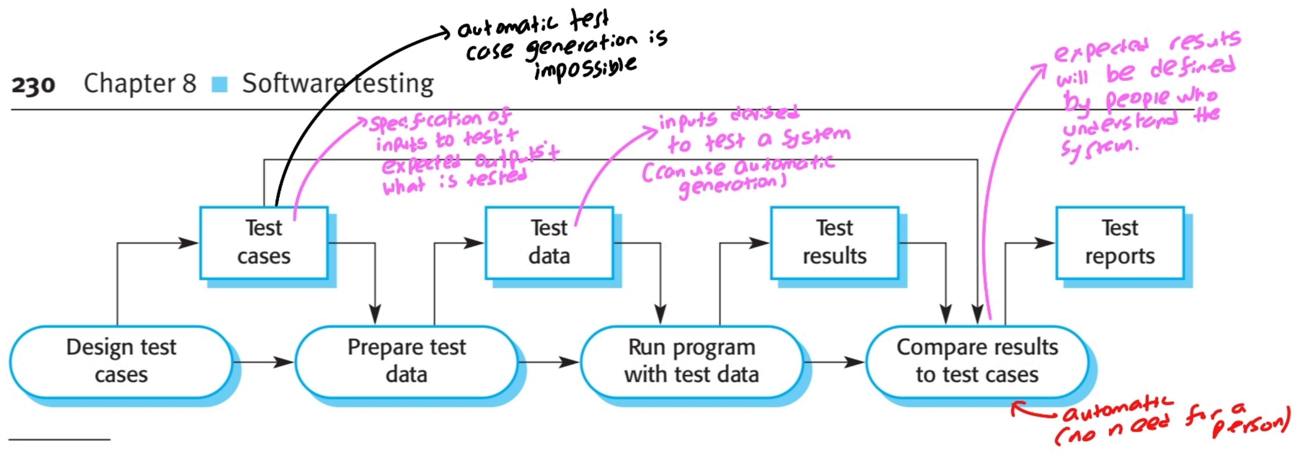
Marketing environment When a software company brings a system to market, it must take into account competing products, the price that customers are willing to pay for a system, and the required schedule for delivering that system. In a competitive environment, the company may decide to release a program before it has been fully tested and debugged because it wants to be the first into the market. If a software product or app is very cheap, users may be willing to tolerate a lower level of reliability.

In this chapter, I focus on testing and testing processes. I discuss reviews and inspections in more detail in Chapter 24 (Quality Management).

Traditional testing process

- Figure 8.3 is an abstract model of the traditional testing process, as <u>used in plan driven development.</u>

230 Chapter 8 Software testing



automatic test case generation is impossible
specification of inputs to test + expected outputs + what is tested
inputs devised to test a system (can use automatic generation)
expected results will be defined by people who understand the system.
Test cases
Test data
Test results
Test reports
Design test cases
Prepare test data
Run program with test data
Compare results to test cases
automatic (no need for a person)

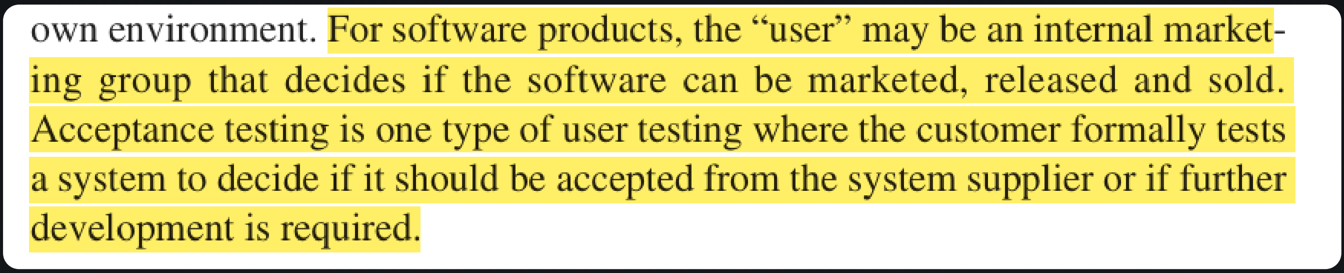
Figure 8.3 A model of the software testing process

Stages of Testing

Typically, a commercial software system has to go through three stages of testing

1. **Development Testing**
    - Conducted by developers during system creation.
    - Includes unit, component, and system testing.
    - Primarily focuses on defect testing.

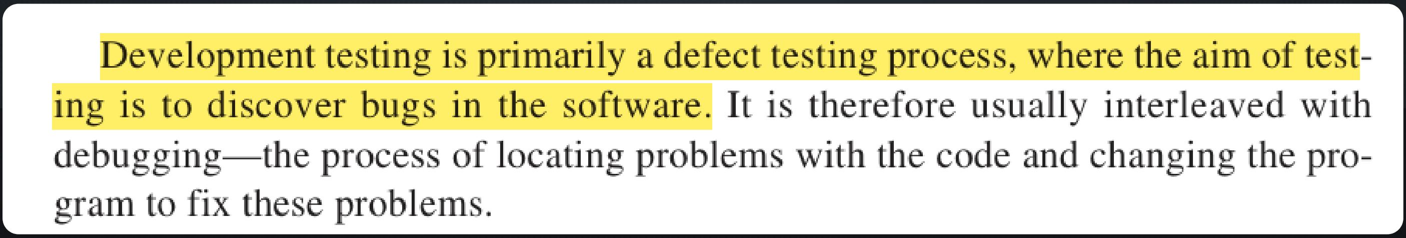
2. **Release Testing**
   - Performed by a separate team before the system is released.
   - Aims to validate the system against requirements of the system stakeholders.
   - Often uses black-box (functional) testing.
3. **User Testing**
   - Conducted by actual users in their environment.
   - Includes alpha, beta, and acceptance testing.
   - Essential for uncovering issues not found in controlled environments.



own environment. For software products, the "user" may be an internal marketing group that decides if the software can be marketed, released and sold. Acceptance testing is one type of user testing where the customer formally tests a system to decide if it should be accepted from the system supplier or if further development is required.

8.1. Development testing

- Development testing includes all testing activities that are carried out by the team developing the system. The tester of the software is usually the programmer who developed that software.
- Some development processes use programmer/tester pairs where each programmer has an associated tester who develops tests and assists with the testing process.
- For critical systems, a more formal process may be used, with a separate testing group within the development team. This group is responsible for developing tests and maintaining detailed records of test results.

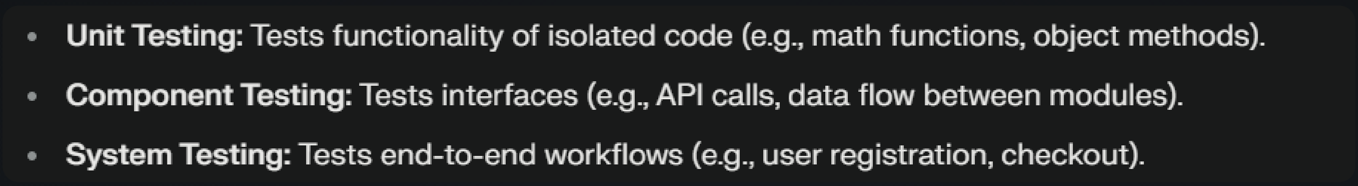


Development testing is primarily a defect testing process, where the aim of testing is to discover bugs in the software. It is therefore usually interleaved with debugging—the process of locating problems with the code and changing the program to fix these problems.

Stages of dev testing

> ### 1. Unit Testing
>
> - **Focus:** Testing individual units (methods, classes) in isolation.
> - **Example:**
>   - *Function:* A method calculating the sum of two numbers.
>
>     ```
>     def add(a, b):
>         return a + b
>     ```
>
>   - *Test Cases:*
>   - `add(2, 3)` → 5 (valid input).
>   - `add(-1, 5)` → 4 (negative numbers).
>   - `add(0, 0)` → 0 (edge case).
>
> ### 2. Component Testing
>
> - **Focus:** where several individual units are integrated to create composite components. Component testing should focus on testing the component interfaces that provide access to the component functions.
> - **Example:**
>   - *Component:* User login module combining input validation and database checks.
>   - *Test Cases:*
>   - Valid username/password → Success message.
>   - Invalid password → Error message.
>   - Empty input → Prompt to fill fields.
>   - *Interface Check:* Ensure the validator correctly passes data to the database component.
>
> ### 3. System Testing

- **Focus:** 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 com ponent interactions.
- **Example:**
  - *System:* E-commerce checkout process.
  - *Test Scenario:*
    1. User adds items to cart.
    2. Enters payment details.
    3. Completes purchase.
  - *Check:* Inventory updates, payment confirmation email sent, order history updated.



- **Unit Testing:** Tests functionality of isolated code (e.g., math functions, object methods).
- **Component Testing:** Tests interfaces (e.g., API calls, data flow between modules).
- **System Testing:** Tests end-to-end workflows (e.g., user registration, checkout).

1.Unit testing

8.2. Test-driven development

8.3. Release testing

8.4. User testing