



Unit Validation Testing

Prof. Durga Prasad Mohapatra
Professor
Dept.of CSE, NIT Rourkela

Testing Levels

4 Testing Levels

- Software tested at 4 levels:
 - Unit testing
 - Integration testing
 - System testing
 - Regression testing

Test Levels

- **Unit testing**

- Test each module (unit, or component) independently
- **Mostly done by developers of the modules**

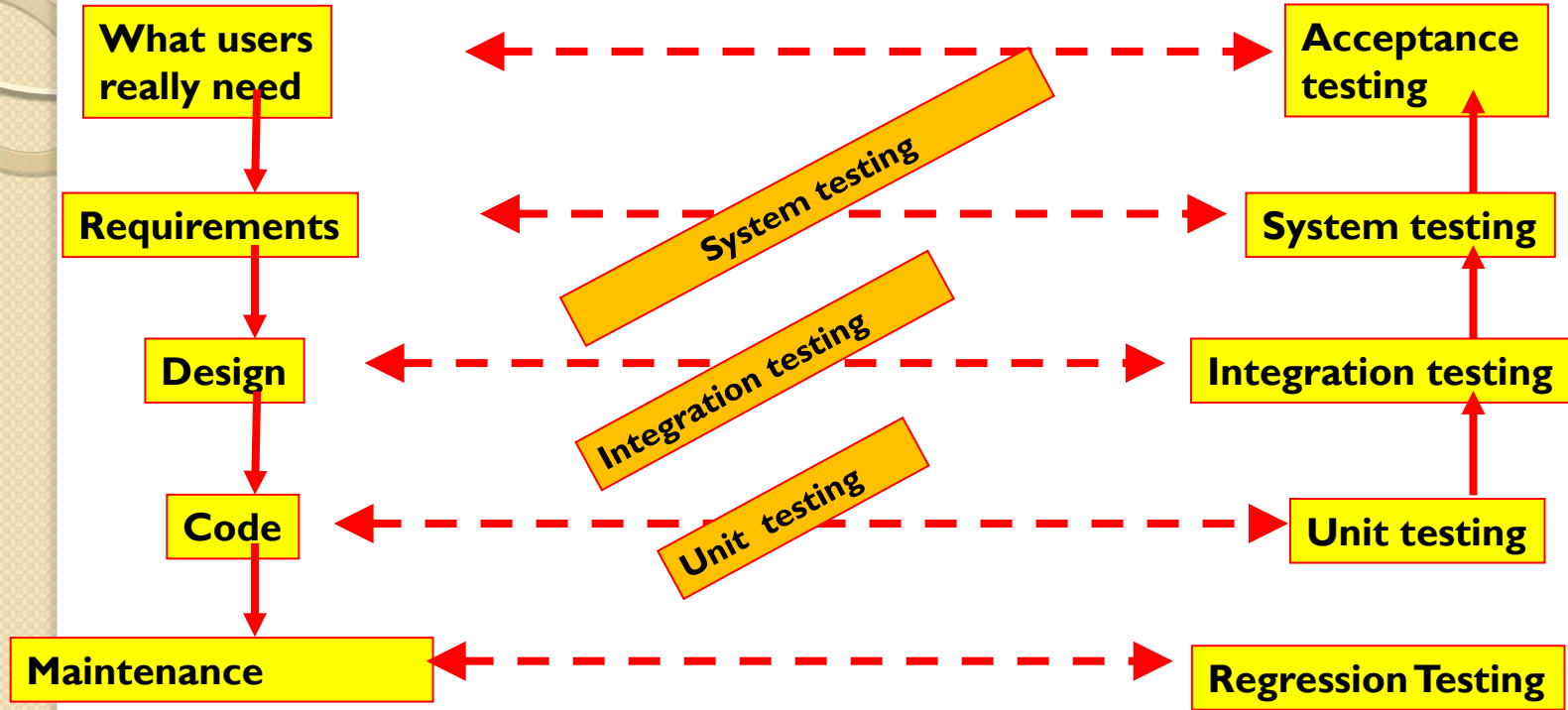
- **Integration and system testing**

- Test the system as a whole
- **Often done by separate testing or QA team**

- **Acceptance testing**

- **Validation of system functions by the customer**

Levels of Testing






Unit testing

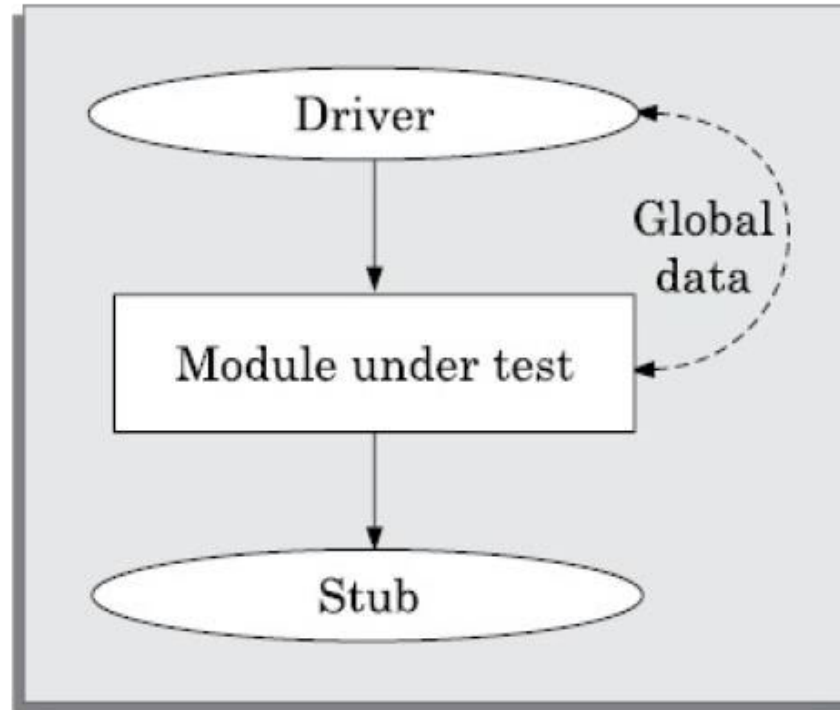
- Unit testing is undertaken after a module has been coded and reviewed.
- This activity is typically undertaken by the coder of the module himself in the coding phase.
- Before carrying out unit testing, the unit test cases have to be designed and the test environment for the unit under test has to be developed.

Driver and stub modules

- In order to test a single module, we need a complete environment to provide all relevant code that is necessary for execution of the module.
- Besides the module under test, the following are needed to test the module:
 - The procedures belonging to other modules that the module under test calls.
 - Non-local data structures that the module accesses.
 - A procedure to call the functions of the module under test with appropriate parameters.

- 
- Modules required to provide the necessary environment (which either call or are called by the module under test) are usually not available until they too have been unit tested.
 - In this context, stubs and drivers are designed to provide the complete environment for a module so that testing can be carried out.
 - The role of stub and driver modules is pictorially shown in the following Figure.

Unit testing with the help of driver and stub modules



Verification versus Validation

- Verification is the process of determining:
 - Whether output of one phase of development conforms to its previous phase.
- Validation is the process of determining:
 - Whether a fully developed system conforms to its SRS document, i.e. the module that has been prepared till now is in conformance with the requirements which were set initially in the SRS or user manual.

Verification versus Validation

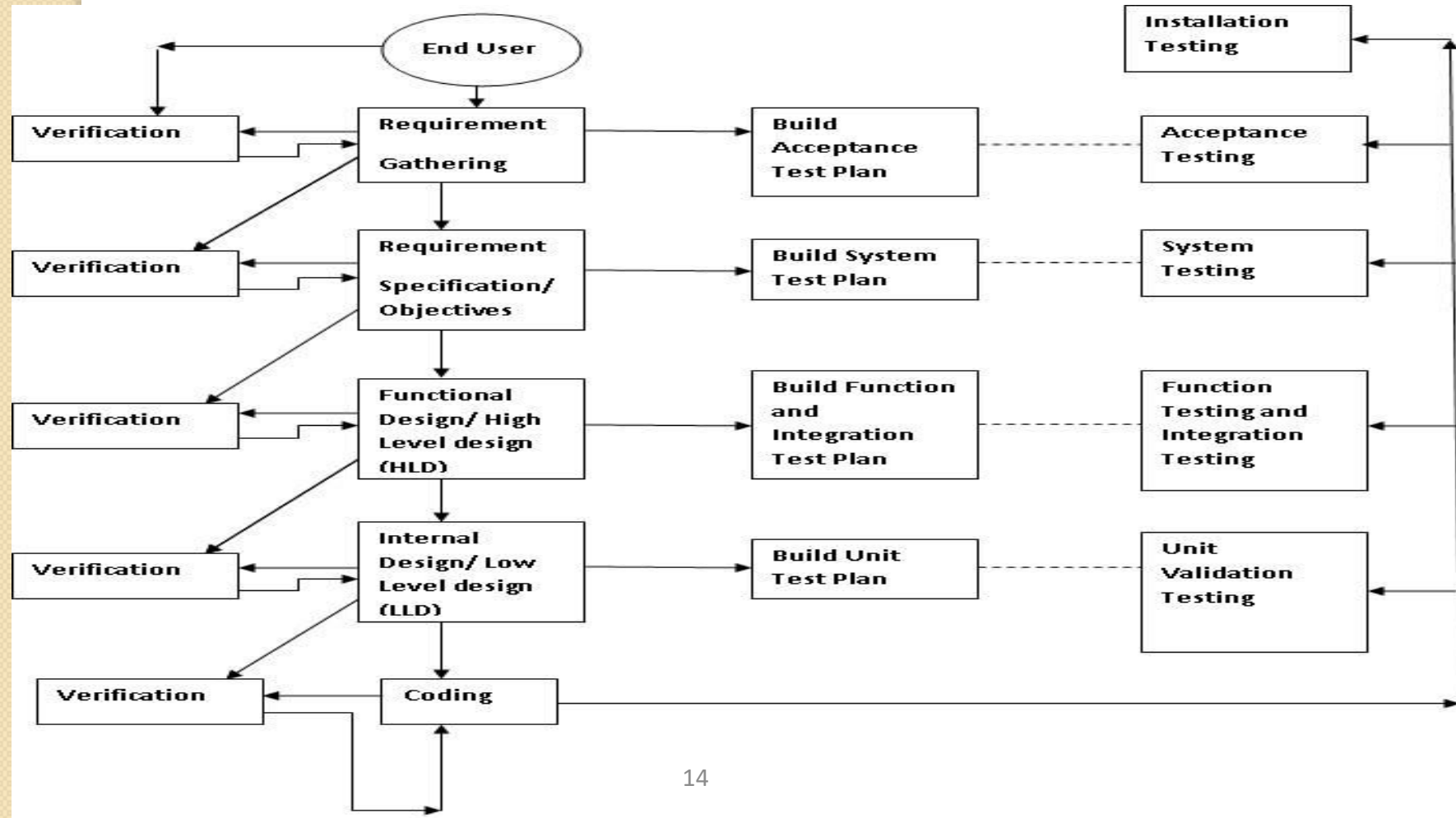
- Verification is concerned with phase containment of errors:
 - Whereas, the aim of validation is that the final product is error free.

Verification and Validation Techniques

- Review
 - Simulation
 - Unit testing
 - Integration testing
- System testing

- Software validation is achieved by a series of black-box tests that demonstrate conformity with requirements.
- Both test plan and test procedure are designed to ensure that
 - Functional requirements are satisfied
 - Behavioral characteristics achieved
 - Performance requirements attained

14



Unit Validation Testing

- Since unit is the smallest building block of the software system, it is the first piece of system to be validated. Before we validate the entire software, units or modules must be validated.
- Unit testing is normally considered an adjunct to the coding step.
- Units must also be validated to ensure that every unit of software has been built in the right manner in conformance with user requirements.

Unit Validation Testing Cont...

- Unit tests ensure that the software meets at least a baseline level of functionality prior to integration and system testing.
- While developing the software, if the developer detects and removes the bug, it offers significant savings of time and costs.
- This type of testing is largely based on **black-box techniques**.

Module

- Software is divided into modules but a module is not an isolated entity.
- The module under consideration might be getting some inputs from another module or the module is calling some other module.
- It means that a module is not independent and cannot be tested in isolation.
- While testing the module, all its interfaces must be simulated if the interfaced modules are not ready at the time of testing the module under consideration.

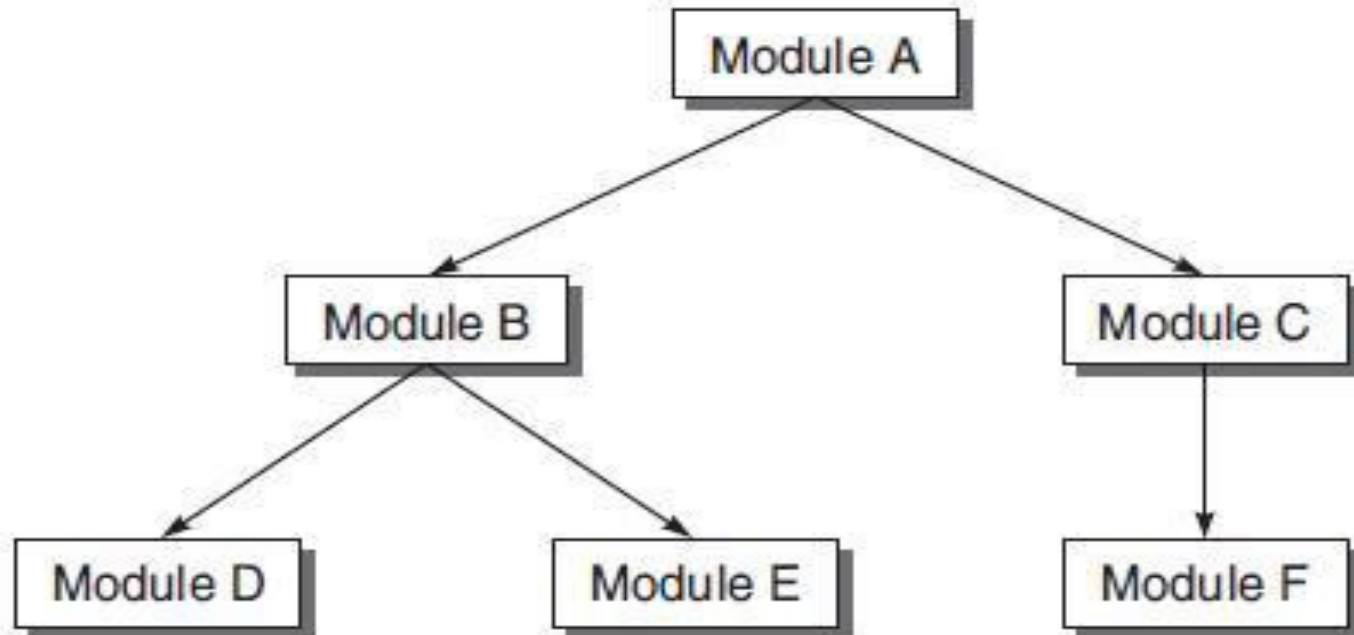
Drivers

- Software test drivers are programs which simulate the behaviors of software components (or modules) that are the control modules of a under test module.
- Control the test cases.
- A driver module should contain the non-local data structures accessed by the module under test.
- Additionally, it should also have the code to call the different functions of the unit under test with appropriate parameter values for testing.
- The driver module may print or interpret the results produced by the module under testing.

Example

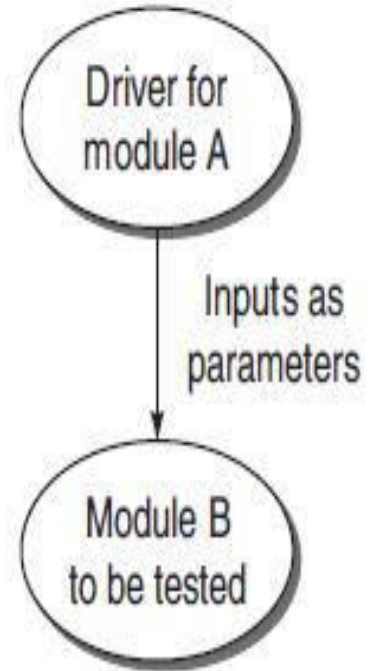
- For example, see the design hierarchy of the modules in Figure 1.
- Suppose module B is under test. In the hierarchy, module A is a superordinate of module B.
- Suppose module A is not ready and B has to be unit tested.

Design Hierarchy of an example system



Example

- A driver module is needed which will simulate module A in the sense that it passes the required inputs to module B and acts as main program for module B.



Drivers

- Test driver is supporting the code and data used to provide an environment for testing a part of a system in isolation.
- In fact, it drives the unit being tested by creating necessary 'inputs' required for the unit and then invokes the unit.

Drivers

- A test driver may take inputs in the following form and call the unit to be tested:

1. It may hard-code the inputs as parameters of the calling unit.
2. It may take the inputs from the user.
3. It may read the inputs from a file.

Drivers

- A test driver provides the following facilities to a unit to be tested:
 1. Initializes the environment desired for testing.
 2. Provides simulated inputs in the required format to the units to be tested.

Drivers

- Projects where commercial drivers are not available, specialized drivers need to be developed.
- This happens mainly in defense projects where projects are developed for a special application.

Stubs

- The module under testing may also call some other module which is not ready at the time of testing.
- Therefore, these modules need to be simulated for testing.
- In most cases, dummy modules instead of actual modules, which are not ready, are prepared for these subordinate modules.
- These dummy modules are called *stubs*.

Stub

- A stub is a dummy procedure that has the same I/O parameters as the function called by the unit under test but has a highly simplified behaviour.
- For example, a stub procedure may produce the expected behaviour using a simple table look up mechanism.

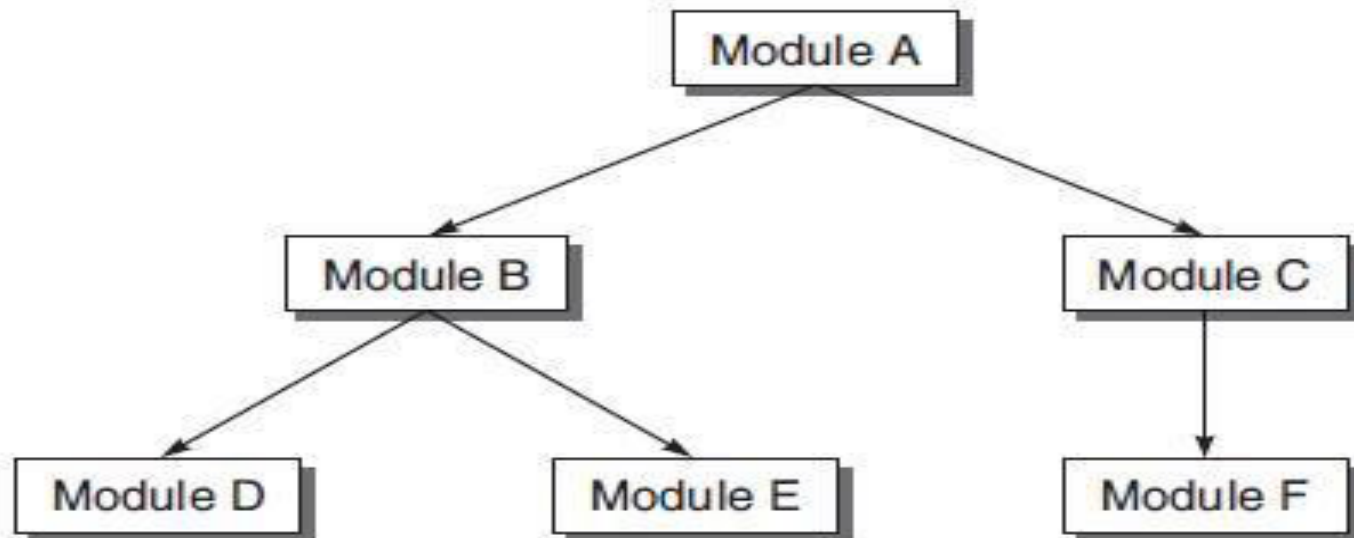
Stubs

- Thus, a stub can be defined as a piece of software that works similar to a unit which is referenced by the unit being tested, but it is much simpler than the actual unit.
- A stub works as a 'stand-in' for the subordinate unit and provides the minimum required behavior for that unit.

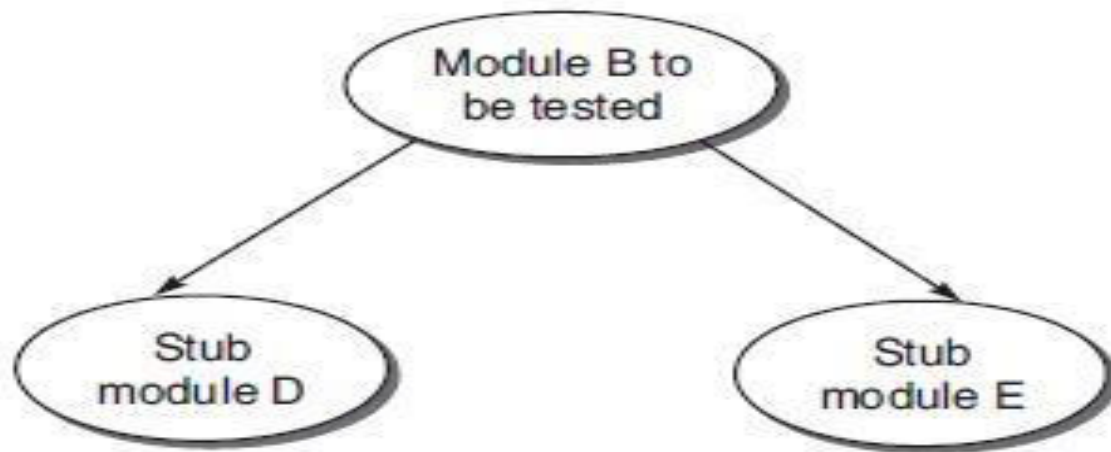
Stubs

- For example, consider Figure 1 again. Module B under test needs to call module D and module E. But they are not ready.
- So there must be some skeletal structure in their place so that they act as dummy modules in place of the actual modules.
- Therefore, stubs are designed for module D and module E, as shown in Figure 3.

Example



Example



Stubs

- Stubs have the following characteristics:

1. Stub is a place holder for the actual module to be called. Therefore, it is not designed with the functionalities performed by the actual module. It is a reduced implementation of the actual module.

Stubs

2. It does not perform any action of its own and returns to the calling unit (which is being tested).
3. We may include a display instruction as a trace message in the body of stub. The idea is that the module to be called is working fine by accepting the input parameters.

Stubs

4. A constant or null must be returned from the body of stub to the calling module.
5. Stub may simulate exceptions or abnormal conditions, if required.

Benefits of Designing Stubs and Drivers

1. Stubs allow the programmer to call a method in the code being developed, even if the method does not have the desired behavior yet.
2. By using stubs and drivers effectively, we can cut down our total debugging and testing time by testing small parts of a program individually, helping us to narrow down problems before they expand.
3. Stubs and drivers can also be an effective tool for demonstrating progress in a business environment.

- However, drivers and stubs represent overheads also. Overhead of designing them may increase the time and cost of the entire software system.

- Therefore, they must be designed simple to keep overheads low.

- Stubs and drivers are generally prepared by the developer of the module under testing.

- Developers use them at the time of unit verification.

- But they can also be used by any other person who is validating the unit.

Example

Consider the following program:

```
main()
{
    Int  a, b, c, sum, diff, mul;
    scanf("%d %d %d", &a, &b, &c);
    sum = calsum(a,b,c);
    diff = caldiff(a,b,c);
    mul = calmul(a,b,c);
    printf("%d %d %d", sum, diff, mul);
}

calsum(int x, int y, int z)
{
    int d;
    d = x + y + z;
    return(d);
}
```

Example

- (a) Suppose `main()` module is not ready for the testing of `calsum()` module. Design a driver module for `main()`.
- (b) Modules `caldiff()` and `calmul()` are not ready when called in `main()`. Design stubs for these two modules.

Solution

Solution

(a) **Driver for main() module:**

```
main()
{
    int a, b, c, sum;

    scanf("%d %d %d", &a, &b, &c);
    sum = calsum(a,b,c);
    printf("The output from calsum module is %d", sum);
}
```

Solution

(b) **Stub for caldiff() Module**

```
caldiff(int x, int y, int z)
{
    printf("Difference calculating module");
    return 0;
}
```

Stub for calmul() Module

```
calmul(int x, int y, int z)
{
    printf("Multiplication calculation module");
    return 0;
}
```


Summary

- Discussed different levels of testing.
- Explained Verification vs. Validation.
- Presented V & V activities.
- Discussed unit validation testing.
- Explained the concept of Drivers and Stubs with an example.



References

1. Rajib Mall, Fundamentals of Software Engineering, (Chapter – 10), Fifth Edition, PHI Learning Pvt. Ltd., 2018.
2. Naresh Chauhan, Software Testing: Principles and Practices, (Chapter – 7), Second Edition, Oxford University Press, 2016.



Thank You