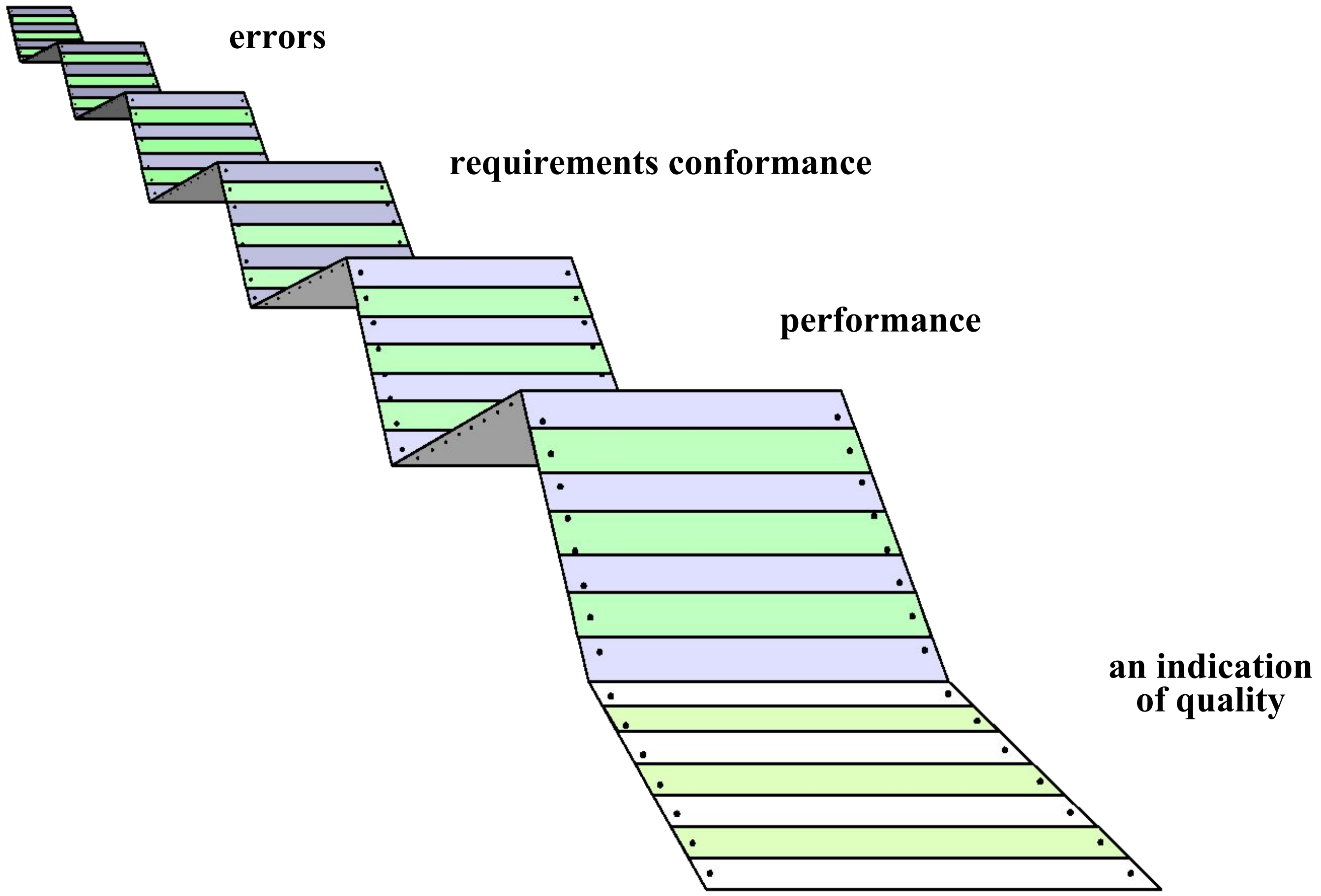
Software Testing

Software Testing

Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

What Testing Shows?



A strategic approach to software testing

To perform effective testing, you should conduct **effective technical reviews**. By doing this, many errors will be eliminated before testing commences.

Testing begins at the **component level** and works "outward" toward the integration of the entire computer-based system.

Different testing techniques are appropriate for different software engineering approaches and at different points in time.

Testing is conducted by the **developer** of the software and (for large projects) an **independent test group**.

Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

V &V

Verification refers to the set of tasks that ensure that software correctly implements a specific function.

Validation refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.

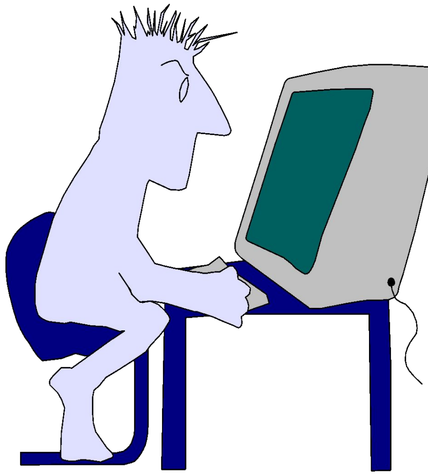
Boehm [Boe81] states this another way:

Verification: "Are we building the product right?"

Validation: "Are we building the right product?"

Who Tests the Software?

developer



independent tester

Understands the system

but, will test "gently"

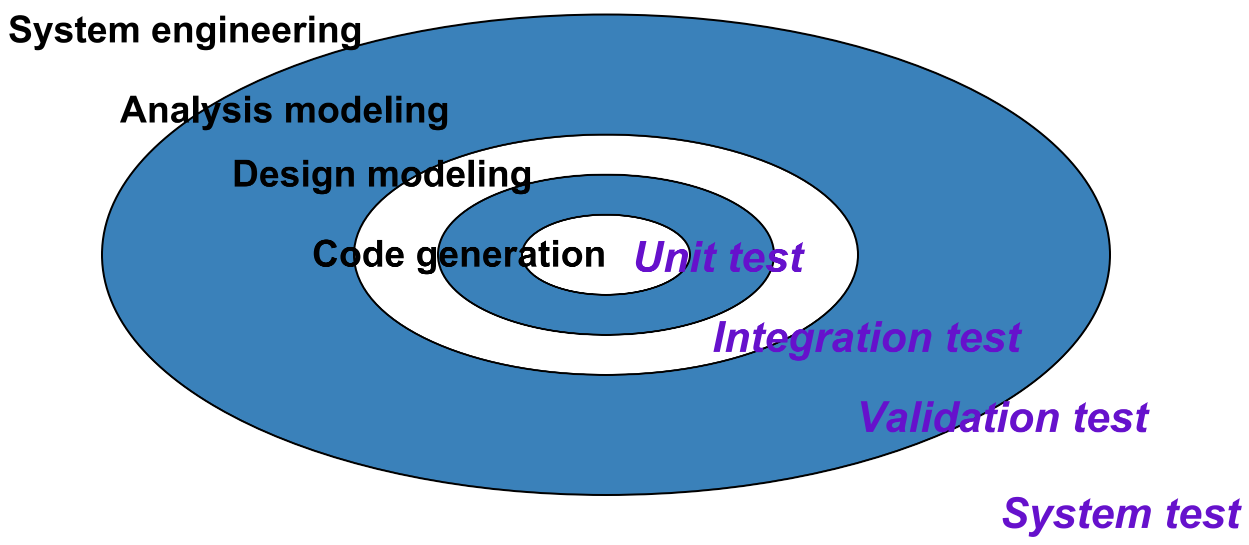
and, is driven by "delivery"

Must learn about the system,

but, will attempt to break it

and, is driven by quality

Software Testing Strategy



Testing Strategy

Begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’

For conventional software

The module (component) is our initial focus

Integration of modules follows

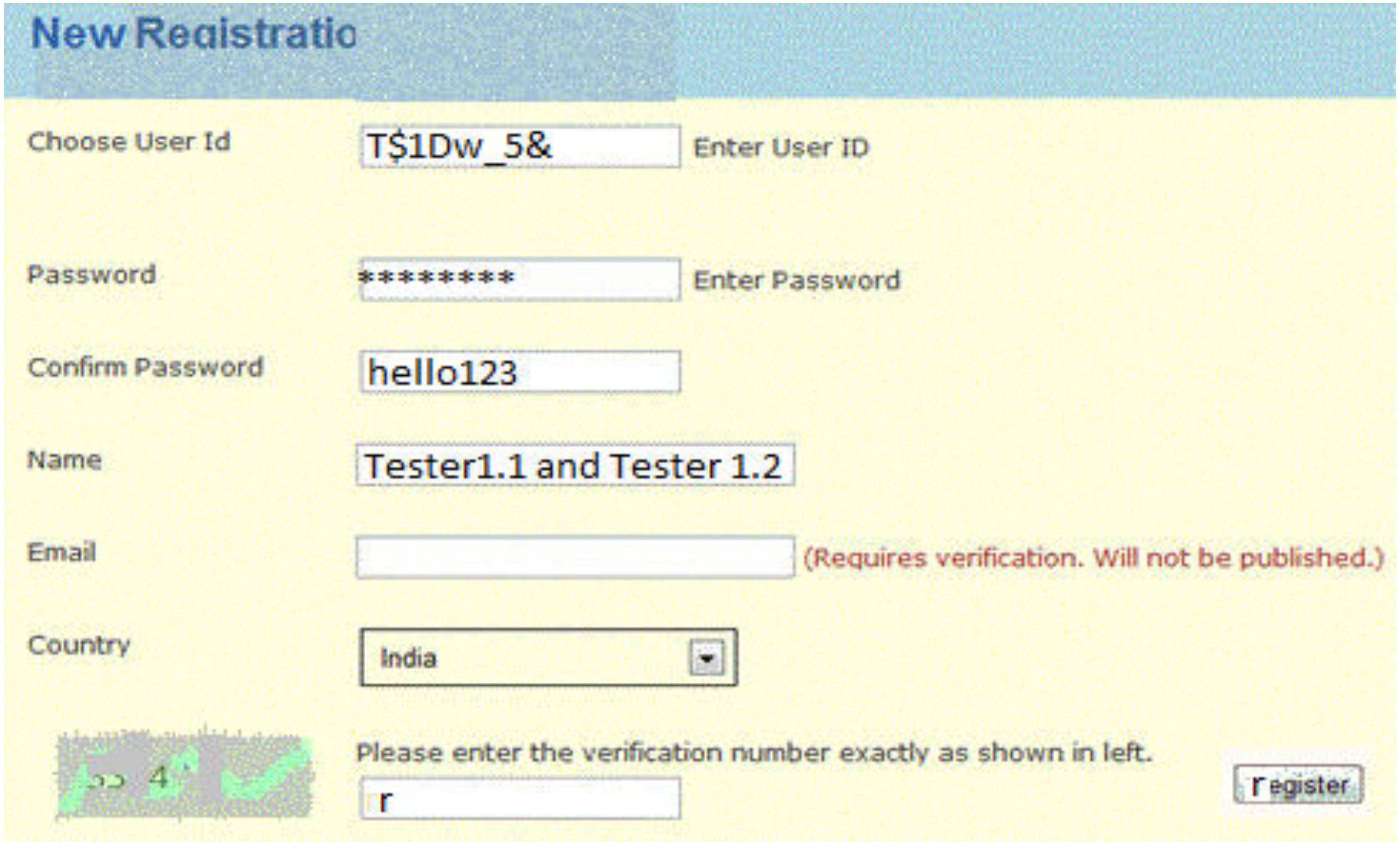
For OO software

Our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Guidelines to a successful testing strategy

- ✓Specify product requirements in a quantifiable manner long before testing commences.
- ✓State testing objectives explicitly.
- ✓Understand the users of the software and develop a profile for each user category.
- ✓Develop a testing plan that emphasizes “rapid cycle testing.”
- ✓Build “robust” software that is designed to test itself
- ✓Use effective technical reviews as a filter prior to testing
- ✓Conduct technical reviews to assess the test strategy and test cases themselves.
- ✓Develop a continuous improvement approach for the testing process.

Can you find out defects on the page shown below?



New Reaistratio

Choose User Id	T$1Dw_5&	Enter User ID
Password	********	Enter Password
Confirm Password	hello123	
Name	Tester1.1 and Tester 1.2	
Email		(Requires verification. Will not be published.)
Country	India	

Please enter the verification number exactly as shown in left.

r

1. *The user Id field accepts special characters.*
2. *Confirm password field does not show content in encrypted mode.*
3. *The name field does not seem to have any validation for number of characters.*
4. *Captcha is not at all readable.*
5. *There is no way user can reload the captcha.*
6. *Register button should be at bottom rather than on side.*
7. *Register button's label has "r" instead of "R".*
8. *There is no cancel button available if user wants to cancel the procedure..*
9. *There is no close button available if user wants to close the page.*
10. *The page title show wrong spelling of Registration.*
11. *Password selection guideline should be provided like the password should be alpha numeric or password strength factor should be present.*
12. *Page title should be New User Registration rather than New Registration.*
13. *Field length and labels should be same for the whole page / form.*
14. *The country field should by default show "Select" rather than selecting a value default.*

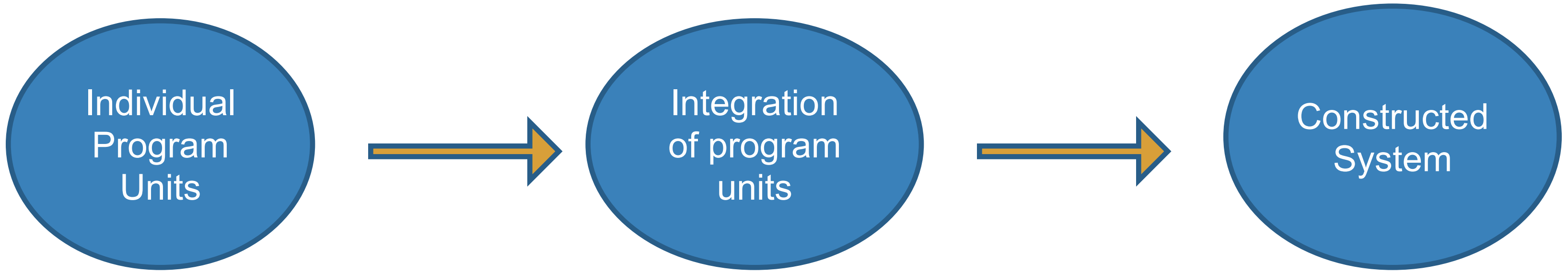
Test Strategies for Conventional Software

Strategies adopted by Industries

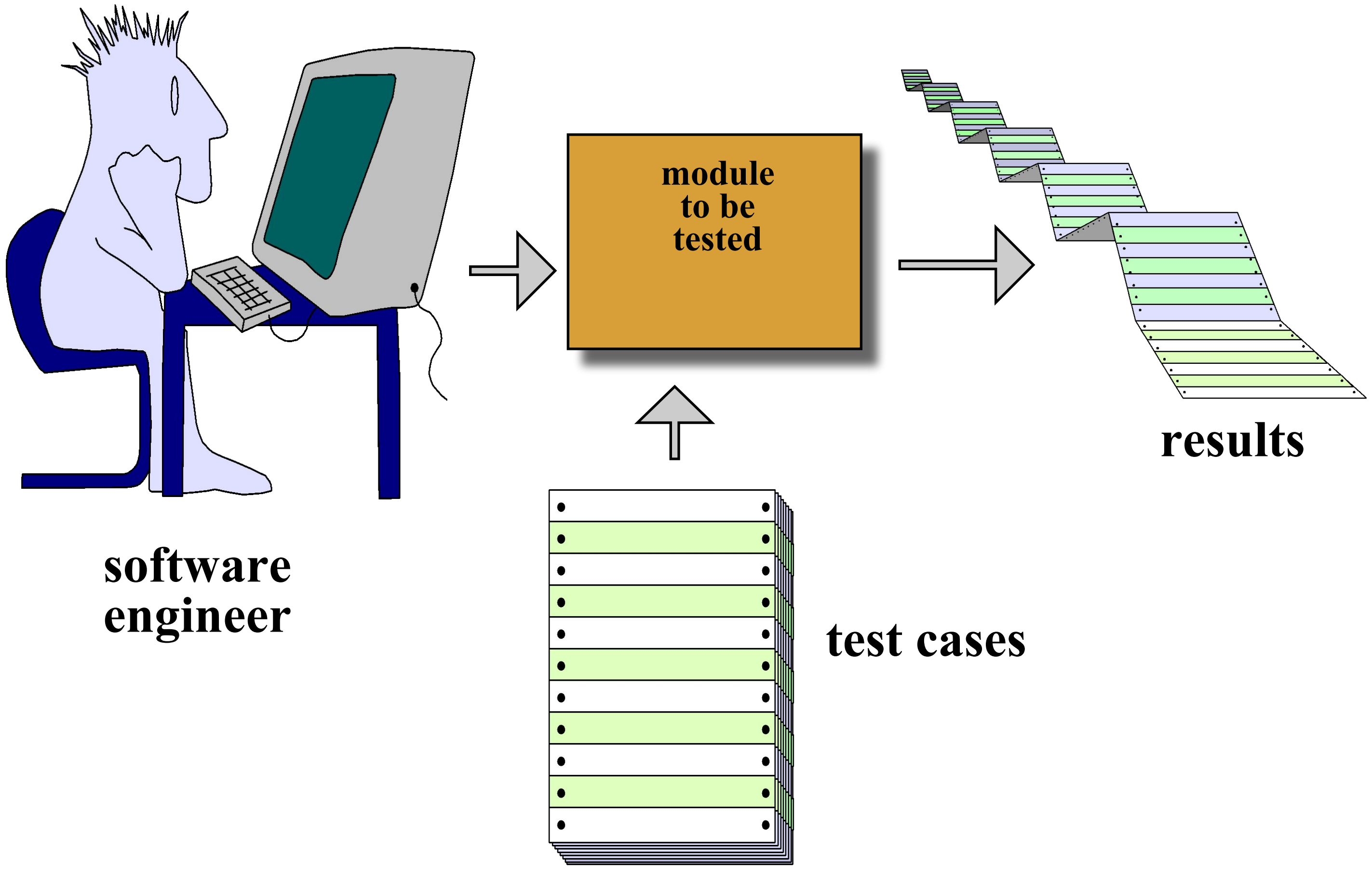
'testing-in-the-large'

'testing-in-the-small' to 'testing-in-the-large'

All industries takes incremental view of testing,

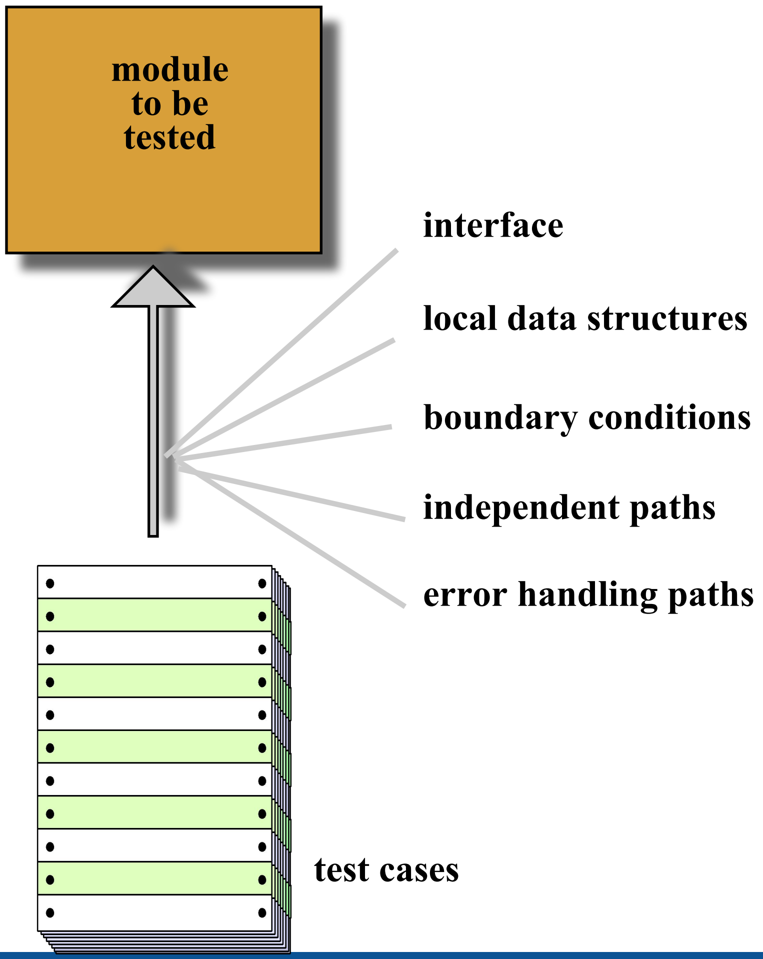


Unit Testing

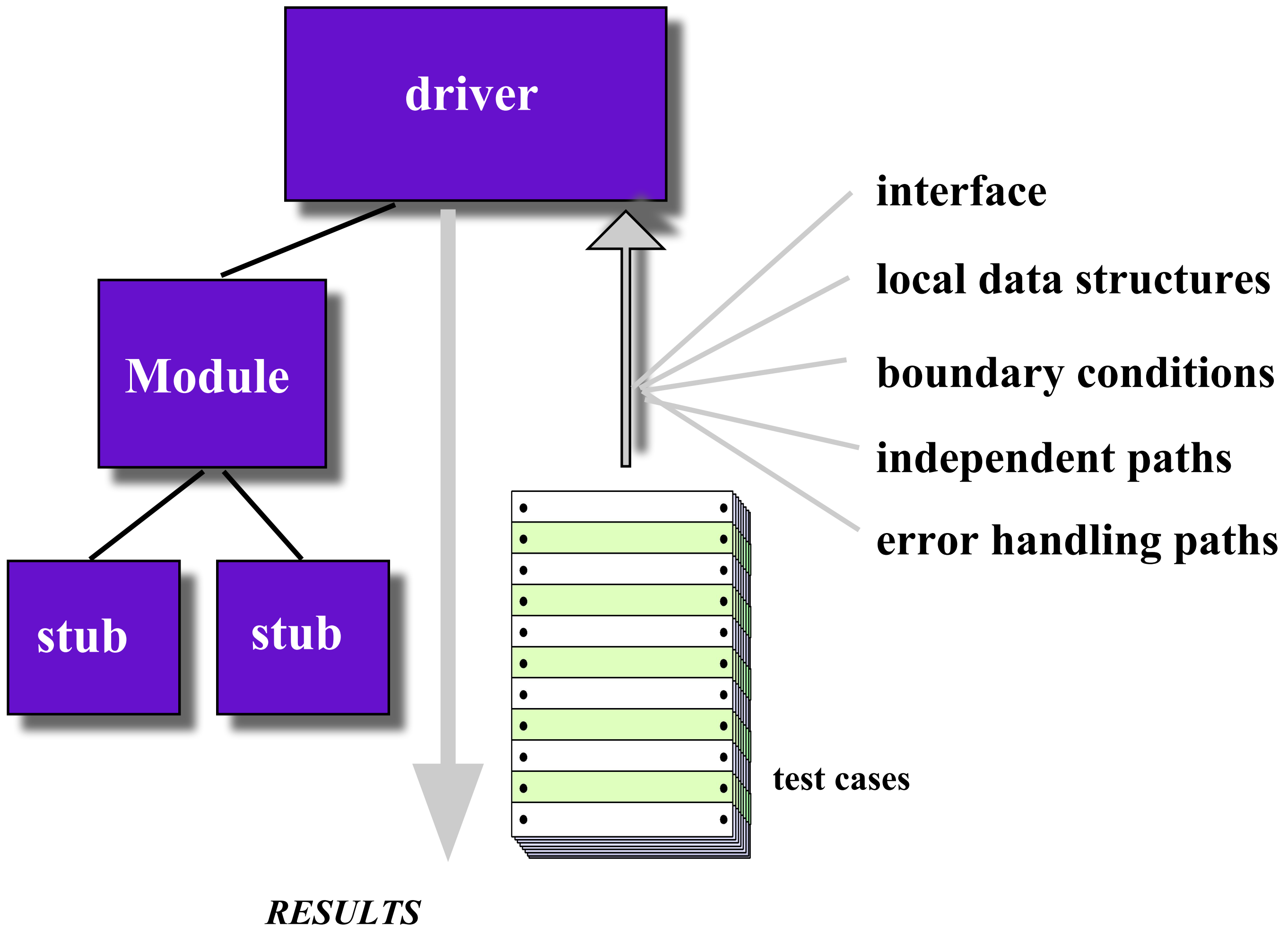


Unit Test Consideration

Checks the Information Flow



Unit Test Procedures

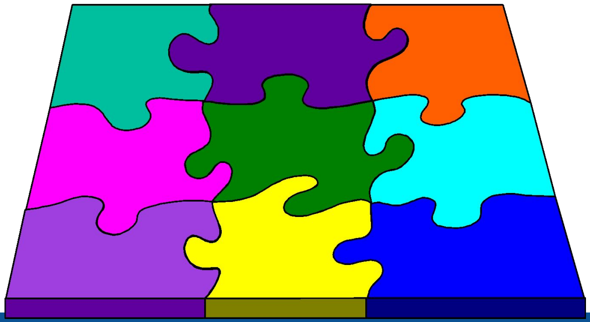


Integration Testing

Systematic technique for constructing the software architecture while at the same time conducting test to uncover errors associated with interfacing.

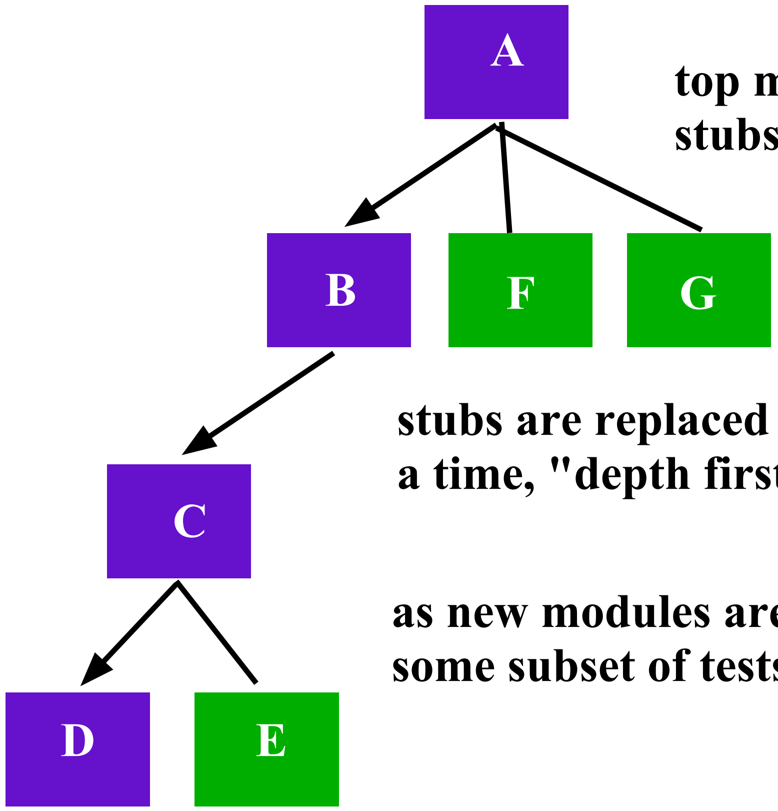
Options:

- The "big bang" approach
- An incremental construction strategy



Incremental Integration Strategies

Top-Down Integration



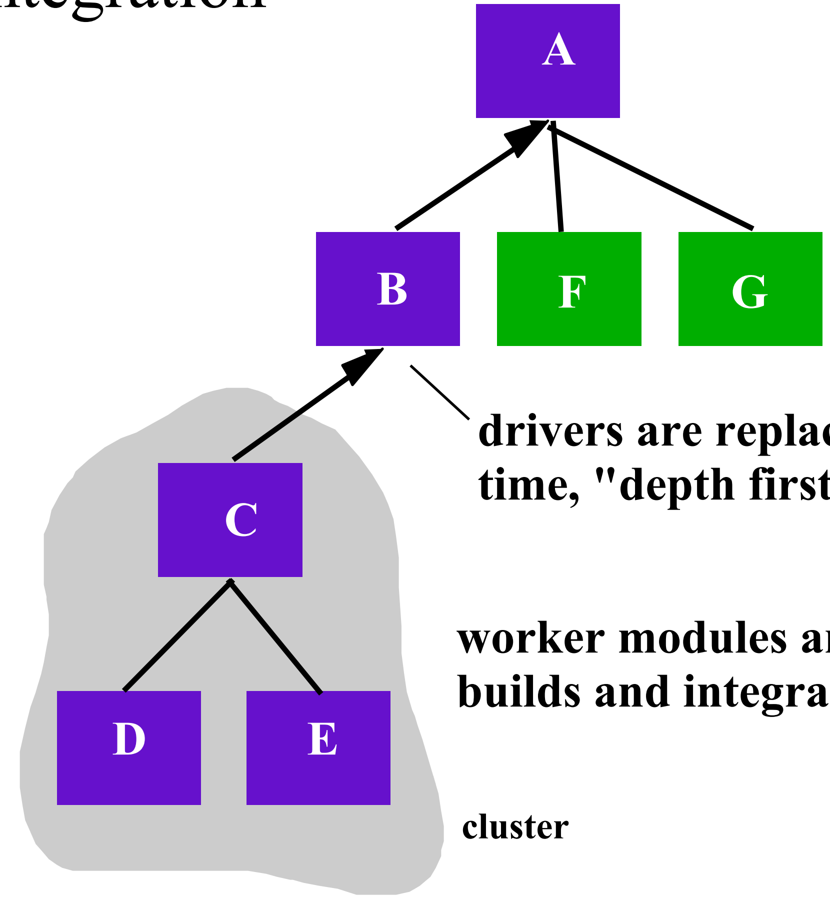
top module is tested with stubs

stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

Incremental Integration Strategies

Bottom-Up Integration



drivers are replaced one at a time, "depth first"

worker modules are grouped into builds and integrated

cluster

Regression Testing

Re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.

Can be done manually

Regression test suite have three classes of test cases

- ✓A representative sample of tests that will exercise all software functions.
- ✓Additional tests that focus on software functions that are likely to be affected by the change.
- ✓Tests that focus on the software components that have been changed.

Smoke Testing

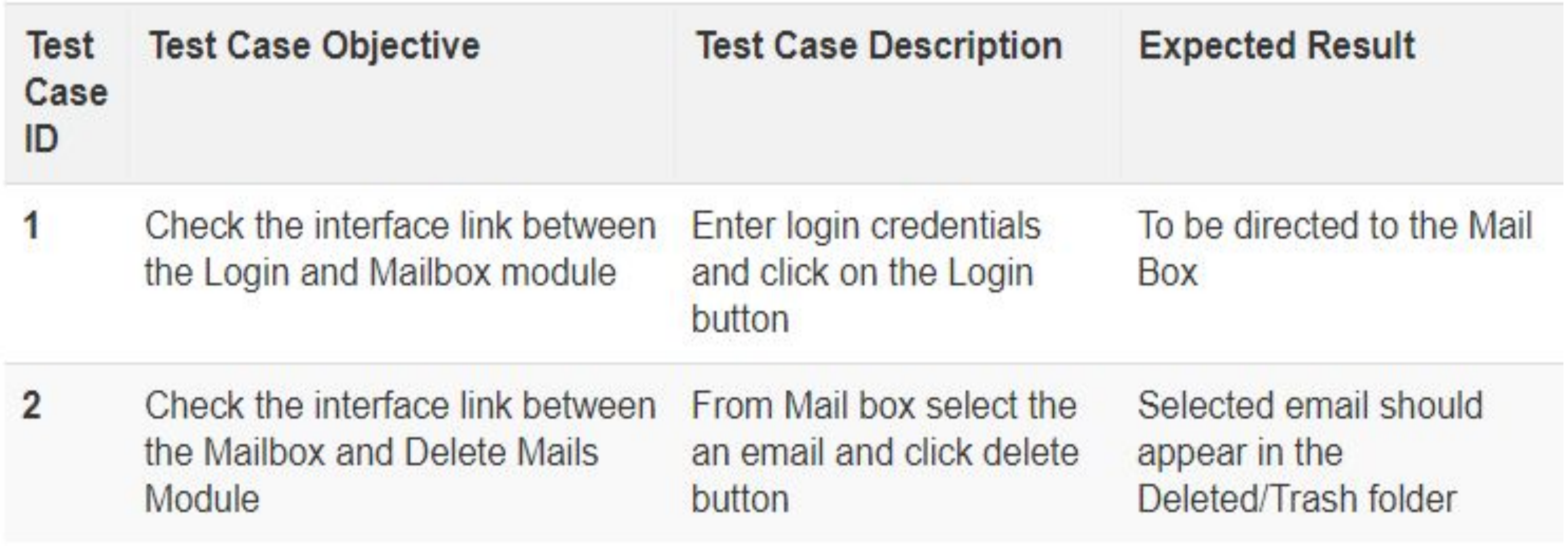
1. Software components that have been translated into code are integrated into a *build*. *A build includes all data files, libraries, reusable modules, and engineered* components that are required to implement one or more product functions.

2. A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "showstopper" errors that have the highest likelihood of throwing the software project behind schedule.

3. The build is integrated with other builds, and the entire product (in its current form) is smoke tested daily. The integration approach may be top down or bottom up.

Example Integration Testing

Application has 3 modules say 'Login Page', 'Mail box' and 'Delete mails' and each of them are integrated logically.



Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mail box select the an email and click delete button	Selected email should appear in the Deleted/Trash folder

Benefits of Smoke Testing

- ✓Integration Risk is minimized
- ✓The quality of the end product is improved
- ✓Error diagnosis and correction are simplified
- ✓Progress is easier to assess

Critical Modules

(1) Addresses several software requirements

(2) Has a high level of control (resides relatively high in the program structure)

(3) Is complex or error prone

(4) Has definite performance requirements

Test Scenarios For Gmail Application

1. Verify that a newly received email is displayed as highlighted in the Inbox section.
2. Verify that a newly received email has correctly displayed sender emailId or name, mail subject and mail body(trimmed to single line).
3. Verify that on clicking the newly received email, user is navigated to email content.
4. Verify that the email contents are correctly displayed with the desired source formatting.
5. Verify that any attachments are attached to the email and is downloadable.
6. Verify that the attachments are scanned for viruses before download.
7. Verify that all the emails marked as read are not highlighted.
8. Verify that all the emails read as well as unread have a mail read time appended at the end on the email list displayed in the inbox section.
9. Verify that count of unread emails is displayed alongside 'Inbox' text in left sidebar of GMail.
10. Verify that unread email count increases by one on receiving a new email.
11. Verify that unread email count decreases by one on reading an email (marking email as read).
12. Verify that email recipients in cc are visible to all user.
13. Verify that email recipients in bcc are not visible to user.
14. Verify that all received emails get piled up in the 'Inbox' section and gets deleted in cyclic fashion based on the size availability.
15. Verify that email can be received from non-gmail emailIds like - yahoo, hotmail etc.

Validation Testing

Focuses on Software Requirement

After the Unit testing and Integration testing

Checking for the conformity with requirements

Validation Test criteria

Test cases for

- ✓Functional requirements are satisfied
- ✓Behavioural characteristics are achieved
- ✓All contents are accurate and properly presented
- ✓All performance requirements are attained
- ✓Documentation is correct
- ✓Usability and other requirements are met

After testing

2. The function or performance characteristics conforms to specification and is accepted
3. A deviation from specification is uncovered and a deficency list is created.

Alpha and Beta Testing

Focus is on customer usage

Alpha

- ✓ Conducted at the developers site by a representative group of end users.
- ✓ Conducted in a controlled environment

Beta

- ✓ Conducted at one or more end-user sites
- ✓ Customer records all problems that encountered and reports to the developers

Characteristics of a testable software

Operability—it operates cleanly

Observability—the results of each test case are readily observed

Controllability—the degree to which testing can be automated and optimized

Decomposability—testing can be targeted

Simplicity—reduce complex architecture and logic to simplify tests

Stability—few changes are requested during testing

Understandability—of the design

What is a Good Test?

A good test has a high probability of finding an error

A good test is not redundant.

A good test should be “best of breed”

A good test should be neither too simple nor too complex

Internal and External Views of Testing

Black Box Testing

Examine fundamental aspect of a system with little regard for the internal logical structure of the software

White Box Testing

Close examination of procedural details. Logical paths through the software and collaborations between components are tested.

White Box Testing

Also called Glass box testing

Test cases that derives

1. Guarantee that all independent paths within a module have been exercised at least once
2. Exercise all logical decisions on their true and false sides
3. Execute all loops at their boundaries and with in their operational bounds
4. Exercise internal data structures to ensure their validity

Basis Path Testing

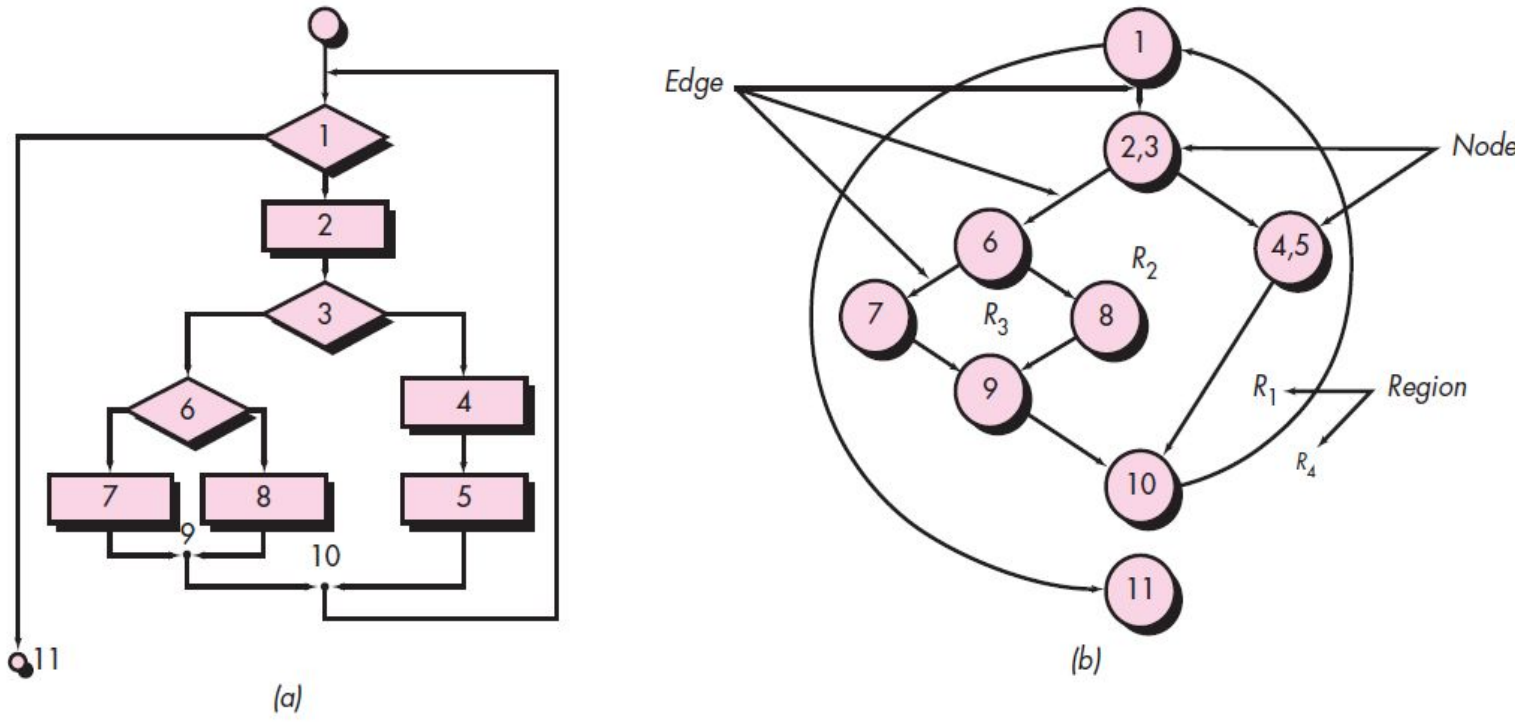
White box testing technique
Proposed by Tom McCabe.

Derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.

Flow Graph Notation

Called as Program Graph

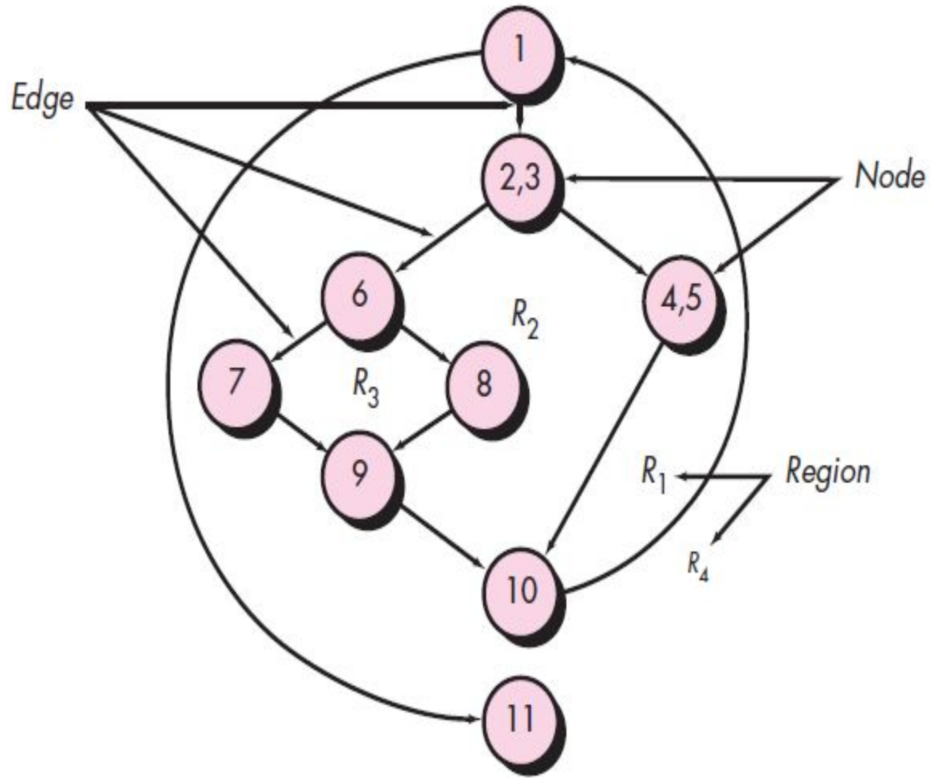
Depicts logical control flow using the notation



Independent Program Paths

Any path through the program that introduces at least one new set of processing statements or a new condition.

Path 1: 1 – 11
Path 2: 1-2-3-4-5-10-1-11
Path 3: 1-2-3-6-8-9-10-1-11
Path 4: 1-2-3-6-7-9-10-1-11



Cyclomatic Complexity

Provides a quantitative measure of the logical complexity of a program

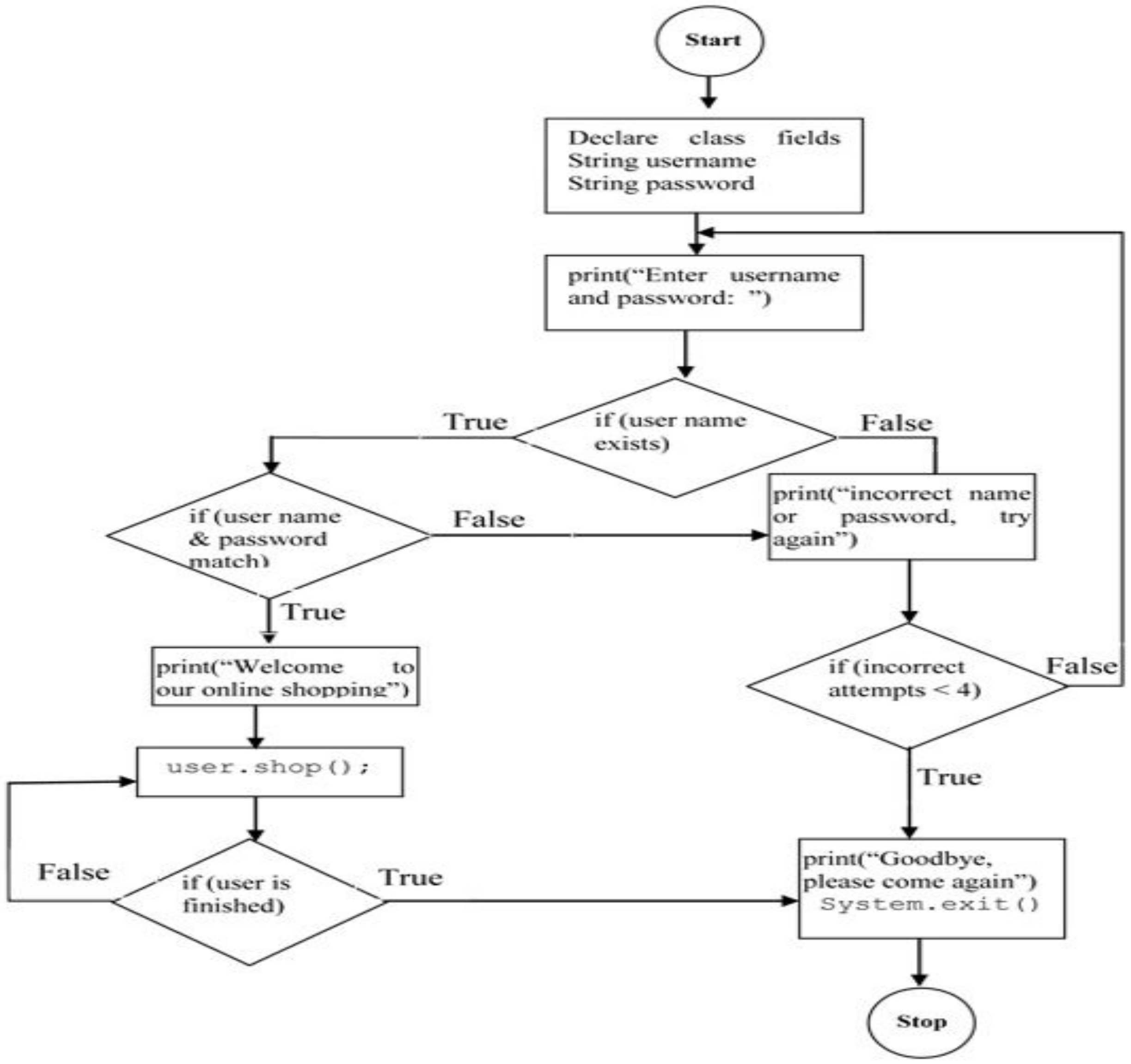
In the basis path test context, the cyclomatic complexity defines the number of independent paths in the basis set of a program

Can be computed,

1. The number of regions of the flow graph
2. $V(G) = E - N + 2$
3. $V(G) = P + 1$ (P is the predicate nodes)

Deriving Test Cases

- Using the design or code as a foundation, draw a corresponding flow graph.
- Determine the cyclomatic complexity of the resultant flow graph.
- Determine a basis set of linearly independent paths.
- Prepare test cases that will force execution of each path in the basis set


Start
Declare class fields
String username
String password
print("Enter username and password: ")
if (user name exists)
True
False
if (user name & password match)
False
print("incorrect name or password, try again")
True
print("Welcome to our online shopping")
user.shop();
if (incorrect attempts < 4)
False
True
False
if (user is finished)
True
print("Goodbye, please come again")
System.exit()
Stop

```
If A=100 then
    if B > C then
        A = B
    else
        A = C
    end if
End if
Print A
Print B
Print C
```

Control Structure Testing

Condition Testing

Dataflow Testing

Loop Testing

Condition Testing

Proposed by Tai[89]

E1 < relational operator> E2

Data Flow Testing

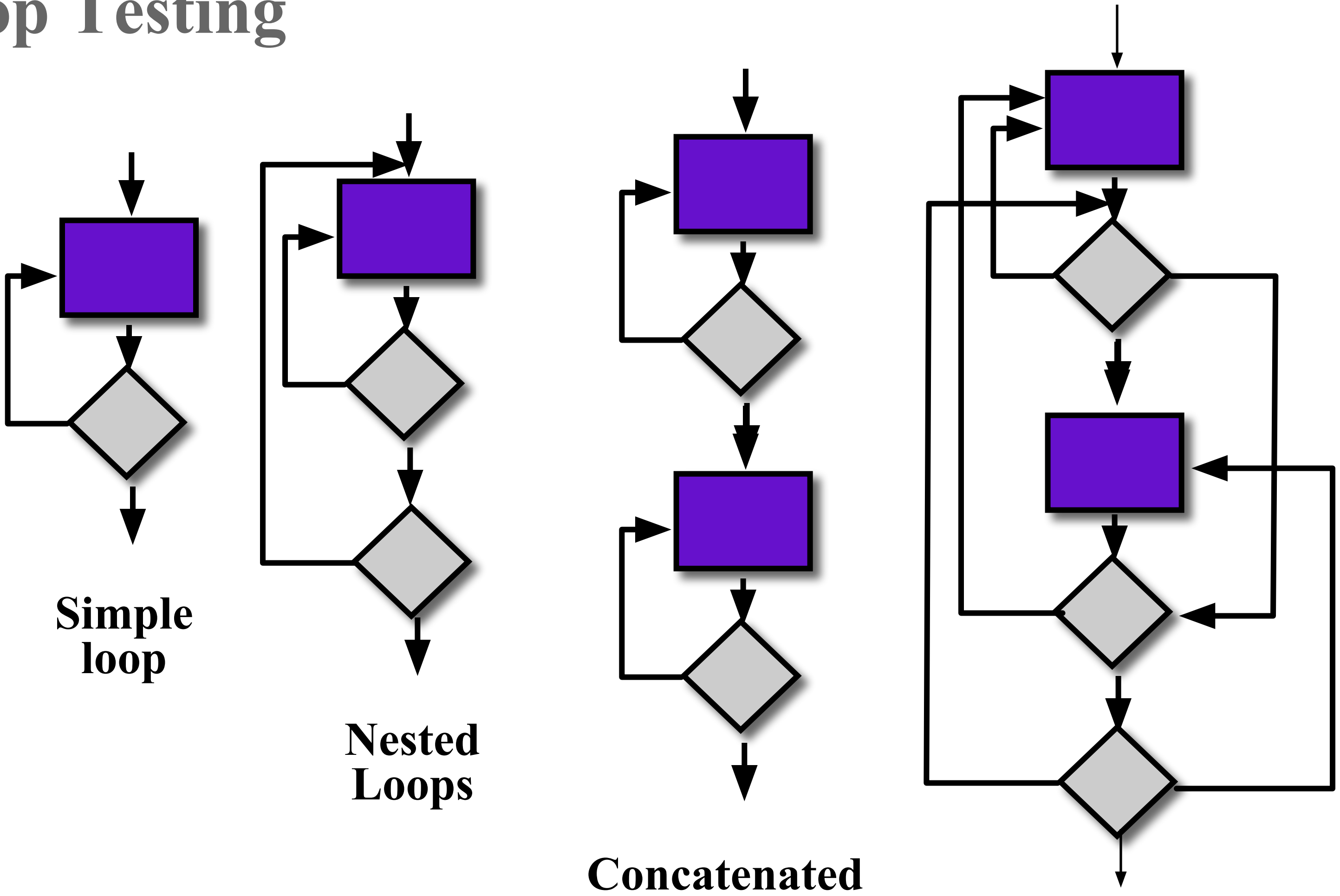
Selects the paths of a program according to the locations of the definitions and uses of variables in the program

Each statement will be assigned with a statement number

DEF(S) = {X | statement S contains a definition of X}

USE(S) = {X | statement S contains a use of X}

Loop Testing



Simple Loops

N- maximum number of allowable passes through the loop

- Skip the loops entirely
- Only one pass through the loop
- Two passes through the loop
- M passes through the loop where m<n
- N-1, N, N+1 passes through the loop

Nested Loops and Concatenated Loops

Nested Loops

Start at the innermost loop. Set all outer loops to their minimum iteration parameter values.

Test the min+1, typical, max-1 and max for the innermost loop, while holding the outer loops at their minimum values.

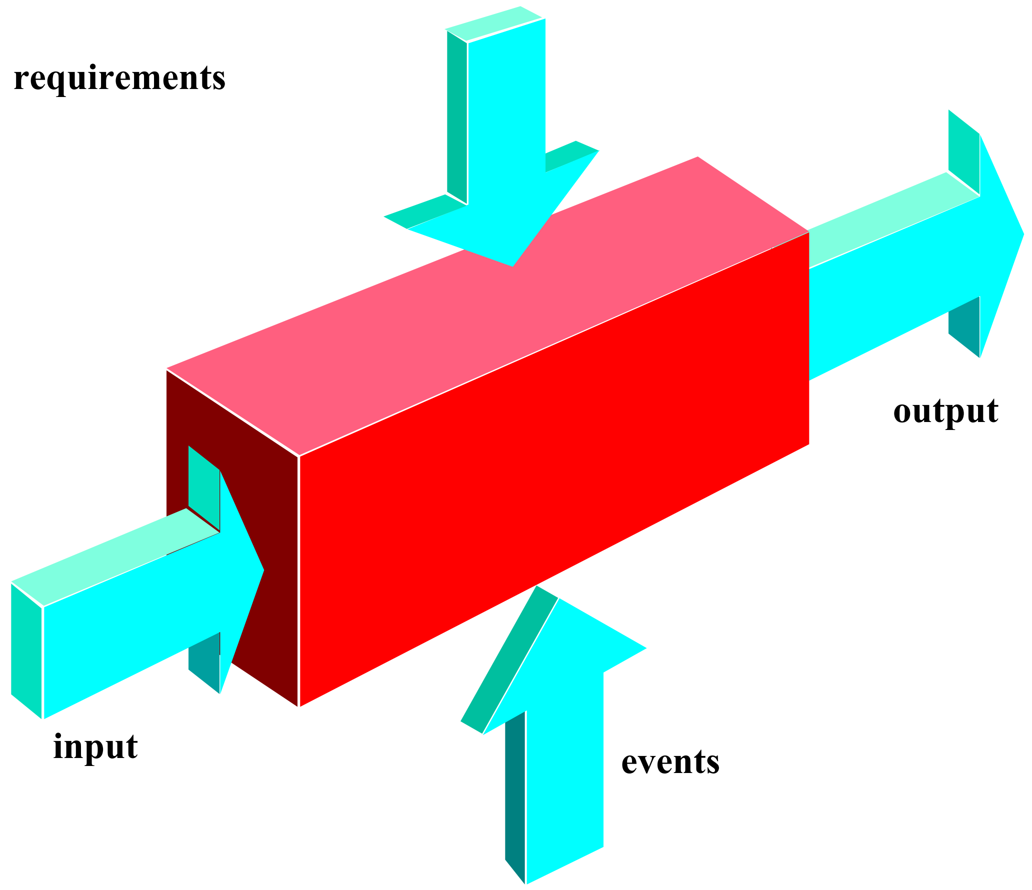
Move out one loop and set it up as in step 2, holding all other loops at typical values. Continue this step until the outermost loop has been tested.

Concatenated Loops

```
If the loops are independent of one another
   then treat each as a simple loop
   else* treat as nested loops
endif*
```

for example, the final loop counter value of loop 1 is used to initialize loop 2.

Black Box Testing



Black Box Testing

Attempt to find errors in the following categories

1. Incorrect or missing functions
2. Interface errors
3. Errors in data structures or database access
4. Behaviour or performance error
5. Initialization or termination error

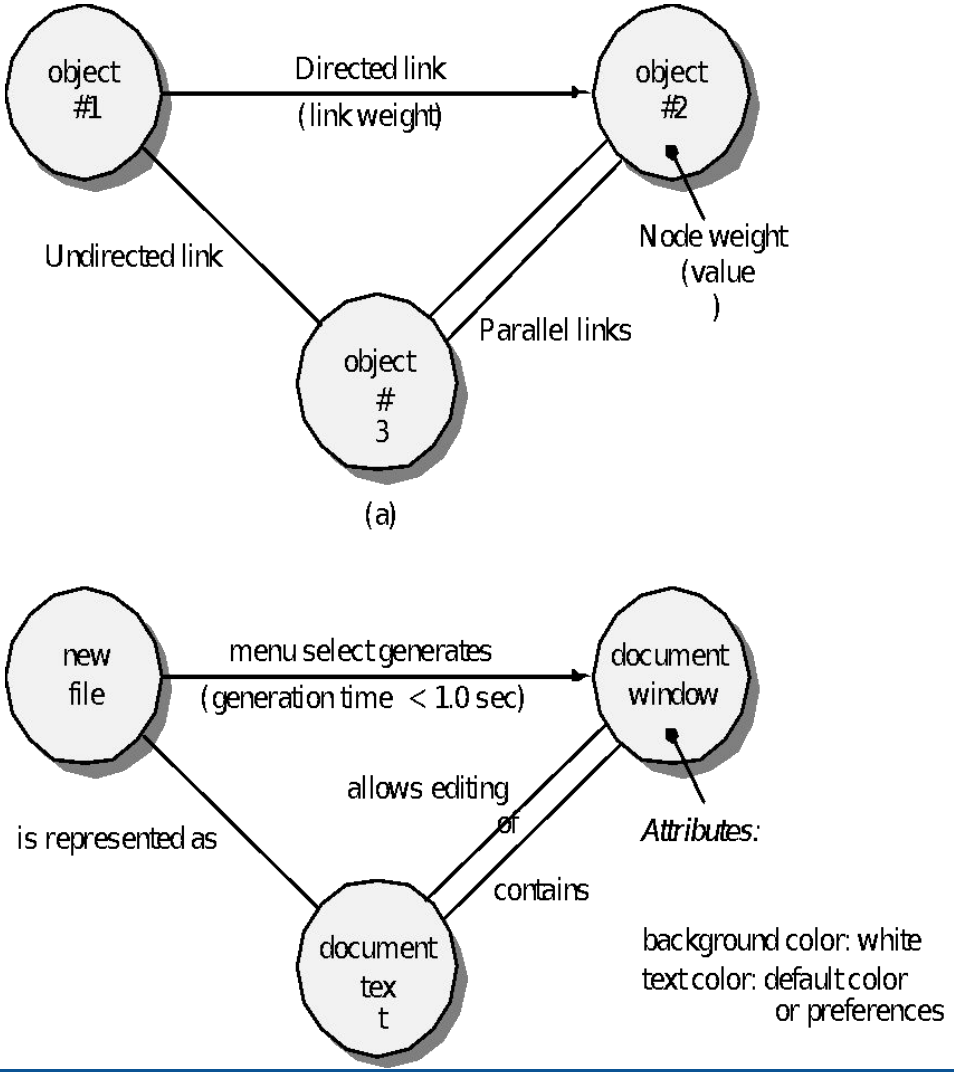
Will Answer ..

1. How is functional validity tested?
2. How is system behavior and performance tested?
3. What classes of input will make good test cases?
4. Is the system particularly sensitive to certain input values?
5. How are the boundaries of a data class isolated?
6. What data rates and data volume can the system tolerate?
7. What effect will specific combinations of data have on system operation?

Graph Based Methods

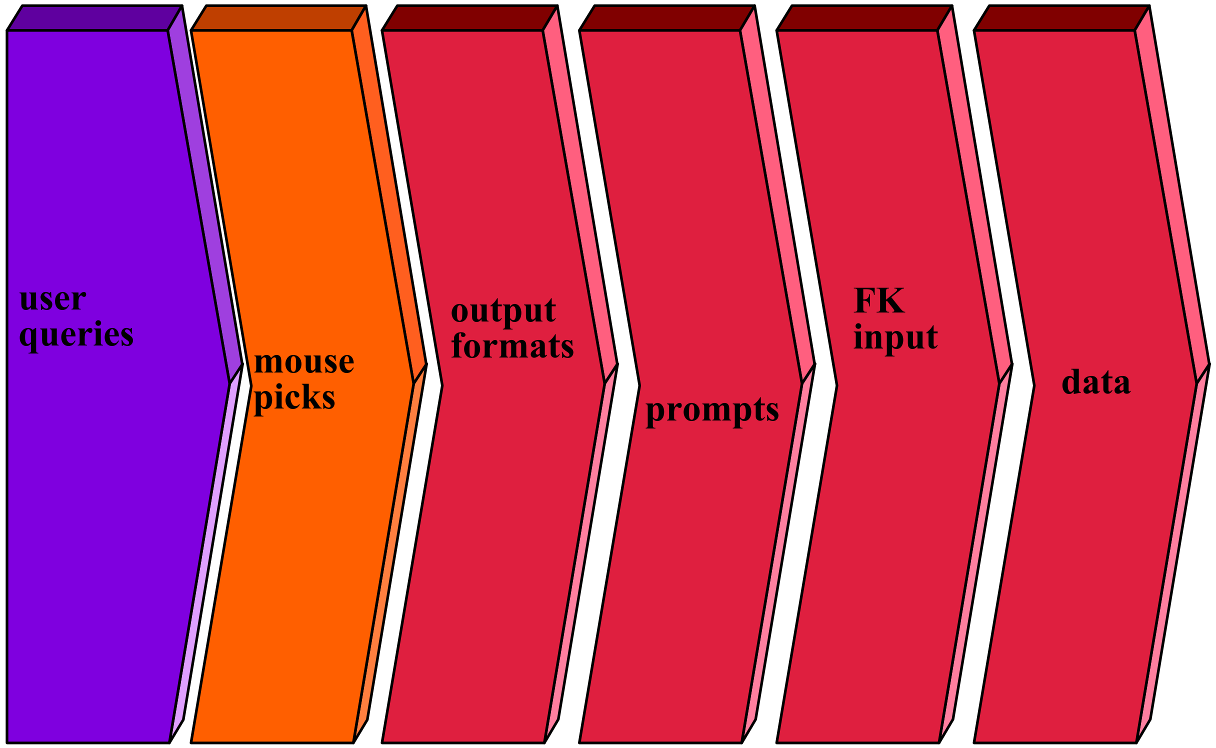
To understand the objects that are modeled in software and the relationships that connect these objects

In this context, we consider the term "objects" in the broadest possible context. It encompasses data objects, traditional components (modules), and object-oriented elements of computer software.



Equivalence Partitioning

Divides the input domain of a program into classes of data from which test cases can be derived



Sample Equivalence classes

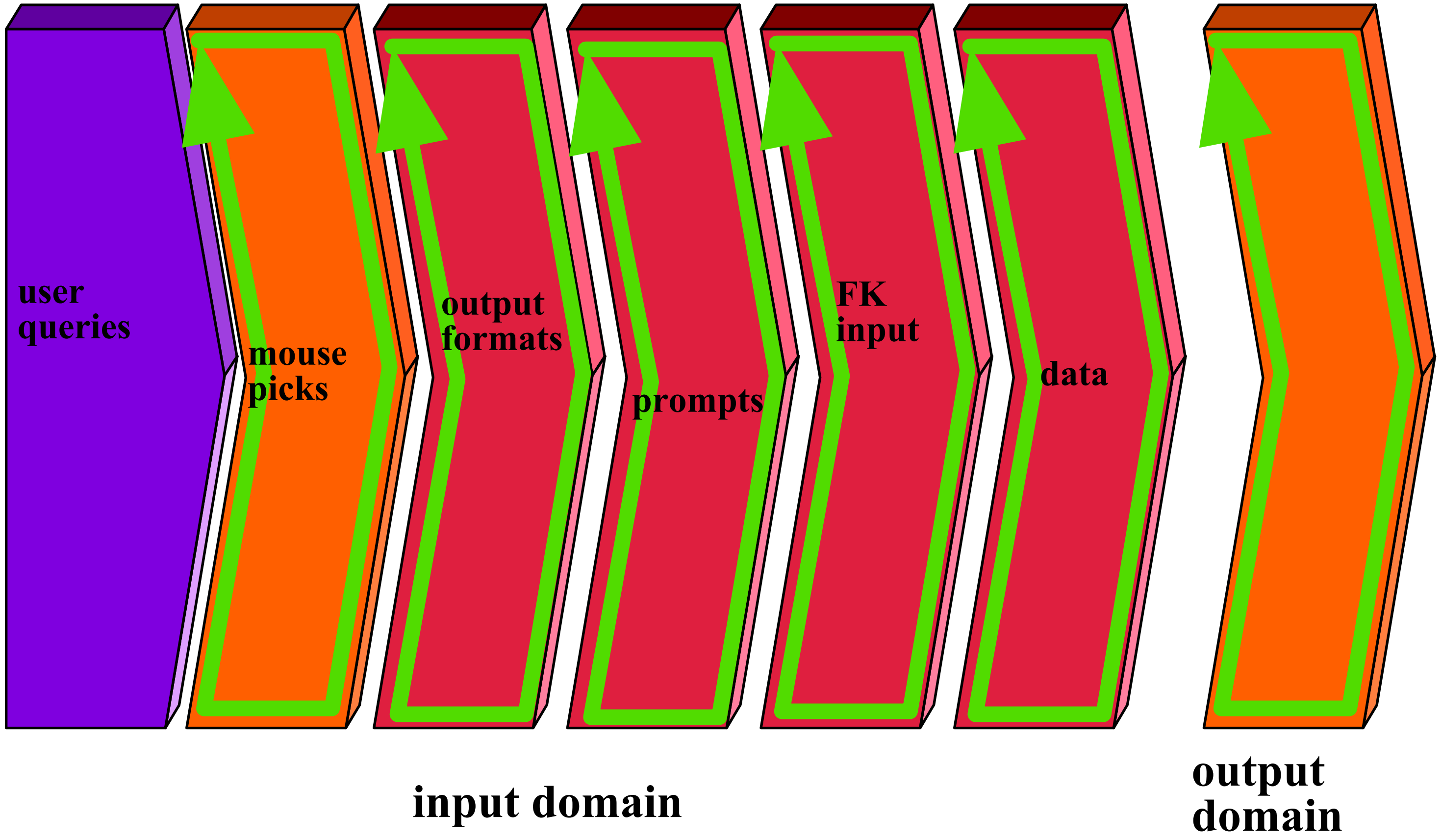
Valid data

- **user supplied commands**
- **responses to system prompts file names**
- **computational data**
  - **physical parameters**
  - **bounding values**
  - **initiation values**
- **output data formatting**
- **responses to error messages**
- **graphical data (e.g., mouse picks)**

Invalid data

- **data outside bounds of the program**
- **physically impossible data**
- **proper value supplied in wrong place**

Boundary Value Analysis



Guidelines for BVA

1. If an input condition specifies a range bounded by values *a and b, test cases* should be designed with values *a and b and just above and just below a and b.*

2. If an input condition specifies a number of values, test cases should be developed that exercise the minimum and maximum numbers. Values just above and below minimum and maximum are also tested.

3. Apply guidelines 1 and 2 to output conditions. For example, assume that a temperature versus pressure table is required as output from an engineering analysis program. Test cases should be designed to create an output report that produces the maximum (and minimum) allowable number of table entries.

4. If internal program data structures have prescribed boundaries (e.g., a table has a defined limit of 100 entries), be certain to design a test case to exercise the data structure at its boundary.

Object Oriented Testing Methods

Conventional test case design, which is driven by an input-process-output view of software

OO testing focuses on designing appropriate sequences of operations to exercise the states of the class.