SOFTWARE TESTING-II

Professor Sudip Misra

Department of Computer Science & Engineering
Indian Institute of Technology, Kharagpur
http://cse.iitkgp.ac.in/~smisra/

Mutation Testing

- The software is first tested:
  - using an initial testing method based on white-box strategies we already discussed.
- After the initial testing is complete,
  - mutation testing is taken up.
- The idea behind mutation testing:
  - make a few arbitrary small changes to a program at a time.

Mutation Testing

- Each time the program is changed,
  - it is called a **mutated program.**
  - the change is called a **mutant.**
- A mutated program:
  - tested against the full test suite of the program.
- If there exists at least one test case in the test suite for which:
  - a mutant gives an incorrect result,
  - then the mutant is said to be dead.

Mutation Testing

- If a mutant remains alive:
  - even after all test cases have been exhausted,
  - *the test suite is enhanced to kill the mutant.*
- The process of generation and killing of mutants:
  - *can be automated by predefining a set of primitive changes that can be applied to the program.*

Mutation Testing

- The primitive changes can be:
  - altering an arithmetic operator,
  - changing the value of a constant,
  - changing a data type, etc.
- A major disadvantage of mutation testing:
  - computationally very expensive,
  - a large number of possible mutants can be generated.

Testing Levels

- Unit testing:
  - test the functionalities of a single module or function.
- Integration testing:
  - test the interfaces among the modules.
- System testing:
  - test the fully integrated system against its functional and non-functional requirements.

Unit Testing

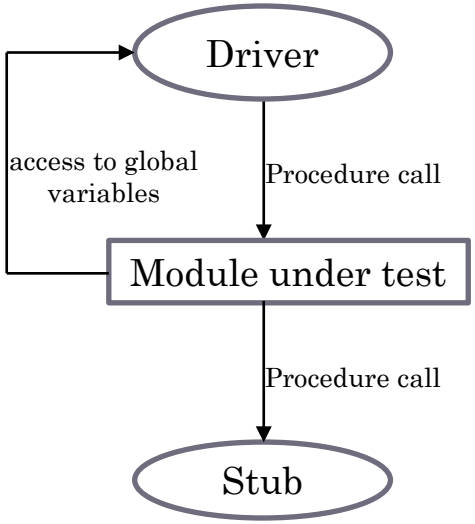
- Involves testing a single isolated module
- Modules in a program are not isolated, they interact with each other.
  - Possible interactions:
    - *calling procedures in other modules*
    - *receiving procedure calls from other modules*
    - *sharing variables*
- For unit testing we need to isolate the module we want to test, we do this using two things
  - drivers and stubs

Drivers and Stubs

- Driver: A program that calls the interface procedures of the module being tested and reports the results
  - A driver simulates a module that calls the module currently being tested
- Stub: A program that has the same interface as a module that is being used by the module being tested, but is simpler.
  - A stub simulates a module called by the module currently being tested
  - **Mock objects**: Create an object that mimics only the behavior needed for testing

Drivers and Stubs

- Driver and Stub should have the same interface as the modules they replace
- Driver and Stub should be simpler than the modules they replace



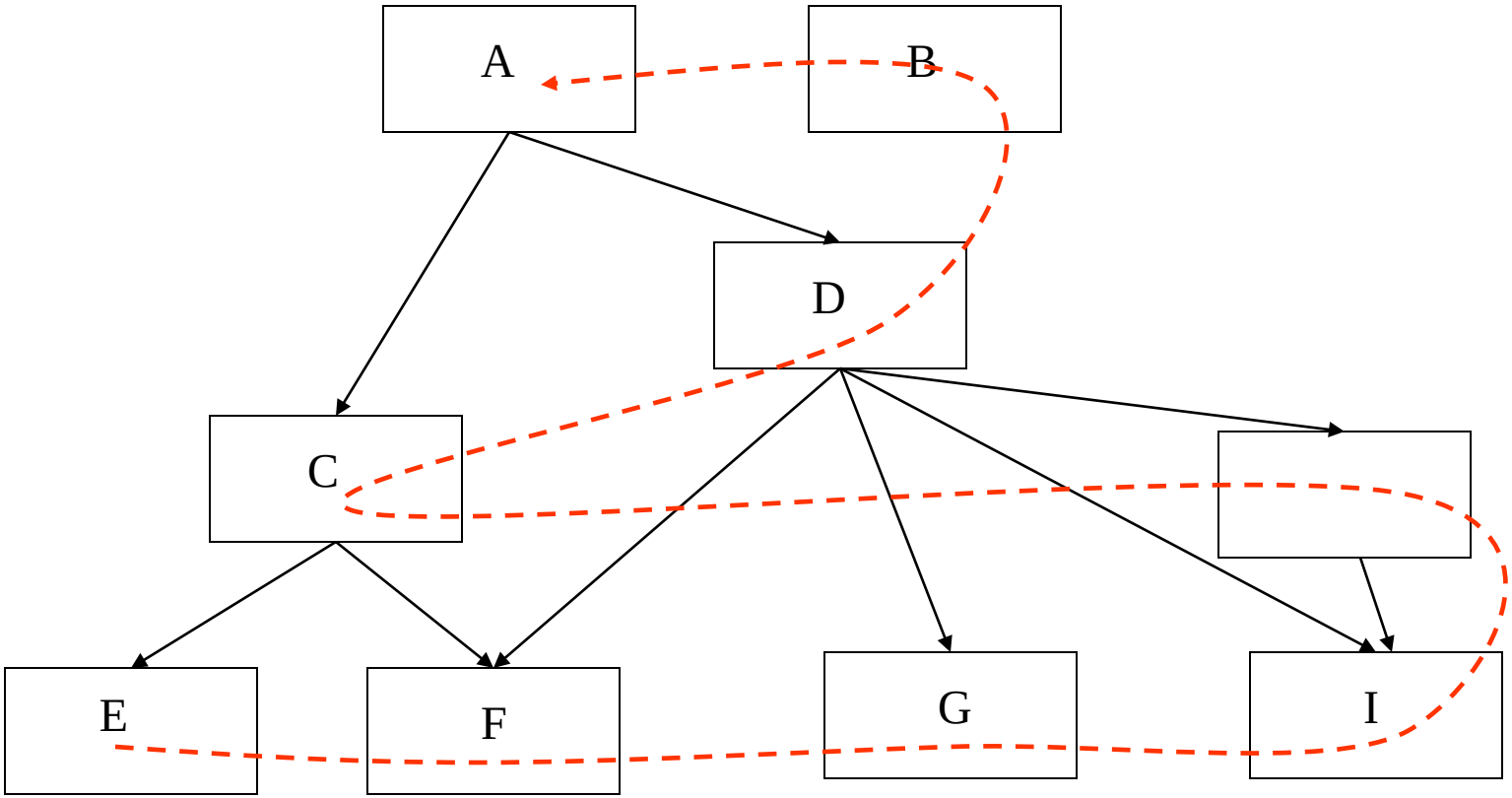
Integration Testing

- Integrated collection of modules tested as a group or partial system
- Integration plan specifies the order in which to combine modules into partial systems
- Different approaches to integration testing
  - Bottom-up
  - Top-down
  - Big-bang
  - Sandwich

Bottom-Up Integration

- Only terminal modules (i.e., the modules that do not call other modules) are tested in isolation
- Modules at lower levels are tested using the previously tested higher level modules
- Non-terminal modules are not tested in isolation
- Requires a module driver for each module to feed the test case input to the interface of the module being tested
  - However, stubs are not needed since we are starting with the terminal modules and use already tested modules when testing modules in the lower levels

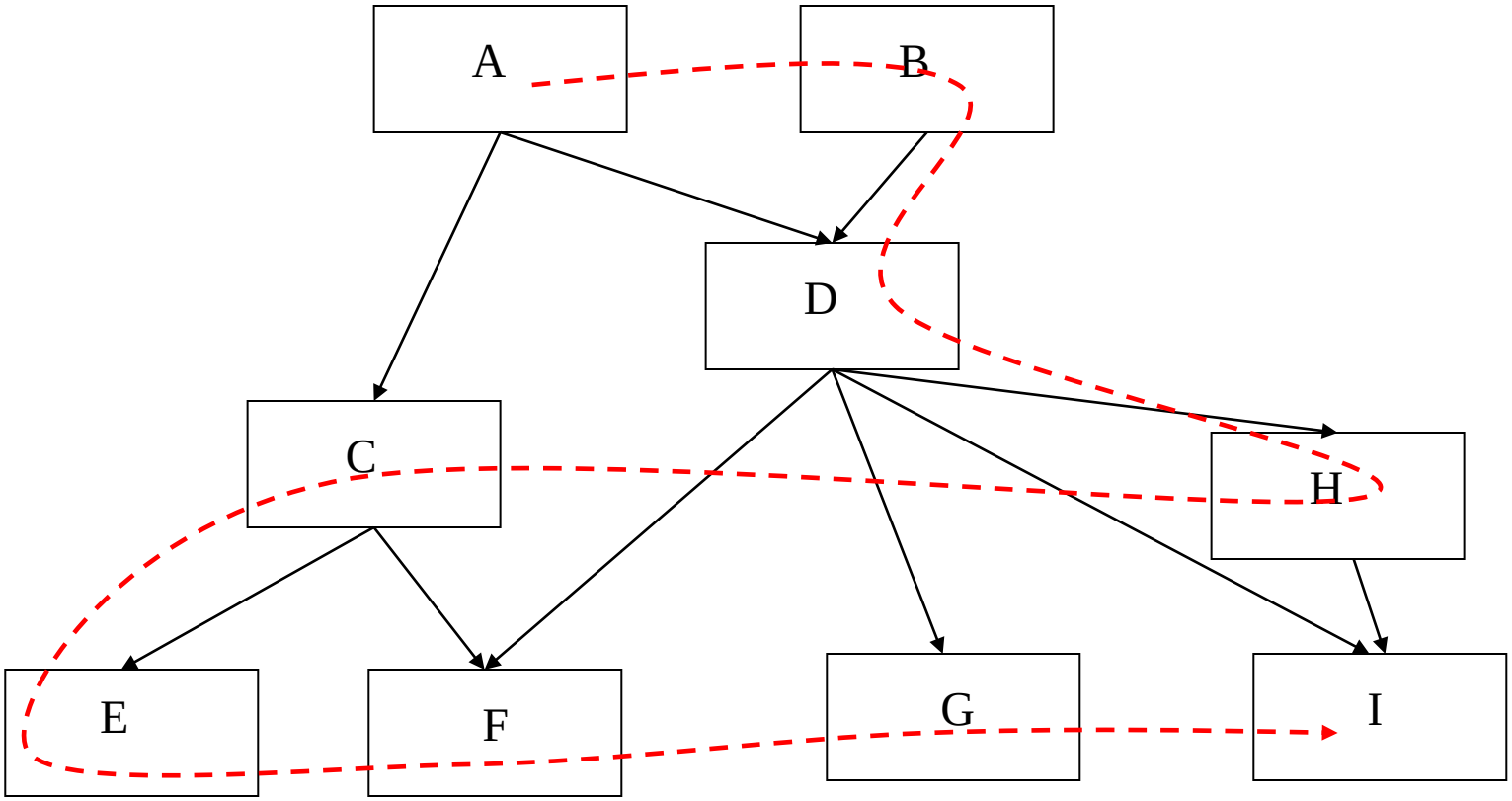
Bottom-Up Integration



Top-down Integration

- Only modules tested in isolation are the modules which are at the highest level
- After a module is tested, the modules directly called by that module are merged with the already tested module and the combination is tested
- Requires stub modules to simulate the functions of the missing modules that may be called
  - However, drivers are not needed since we are starting with the modules which is not used by any other module and use already tested modules when testing modules in the higher levels

Top-down Integration



Other Approaches to Integration

- Sandwich Integration
  - Compromise between bottom-up and top-down testing
  - Simultaneously begin bottom-up and top-down testing and meet at a predetermined point in the middle
- Big Bang Integration
  - Every module is unit tested in isolation
  - After all of the modules are tested they are all integrated together at once and tested
  - No driver or stub is needed
  - However, in this approach, it may be hard to isolate the bugs!
  - this technique is used only for very small systems.

Phased versus Incremental Integration Testing

- Integration can be incremental or phased.
- In incremental integration testing,
  - only one new module is added to the partial system each time.
- In phased integration,
  - a group of related modules are added to the partially integrated system each time

Phased versus Incremental Integration Testing

- Phased integration requires less number of integration steps:
  - compared to the incremental integration approach.
- However, when failures are detected,
  - it is easier to debug if using incremental testing
    - since errors are very likely to be in the newly integrated module.

System Testing

- System testing follows the integration phase
  - testing the system as a whole
- Test cases can be constructed based on the requirements specifications
  - main purpose is to assure that the system meets its requirements
- Manual testing
  - Somebody uses the software on a bunch of scenarios and records the results
  - Use cases and use case scenarios in the requirements specification would be very helpful here
  - manual testing is sometimes unavoidable: usability testing

System Testing

- *Alpha testing* is performed within the development organization
- *Beta testing* is performed by a select group of friendly customers
- *Acceptance Testing*
  - System testing performed by the customer to determine whether he/she should accept the delivery of the system.
  - Formally last phase of testing !

System Testing

- During system testing, in addition to functional tests *performance tests* are performed.
  - To check whether the system meets non-functional requirements given in the SRS

System Testing

- Types of performance testing
  - Stress testing
  - Volume testing
  - Configuration testing
  - Compatibility testing
  - Recovery testing
  - Maintenance testing
  - Documentation testing
  - Usability testing

System Testing

- Stress Testing
  - Also known as endurance testing
  - push system to extreme situations and see if it fails
  - designed to impose a range of abnormal and even illegal input conditions
  - If an operating system is supposed to support 15 multiprogrammed jobs,
    - the system is stressed by attempting to run 15 or more jobs simultaneously.

System Testing

- Volume Testing
  - Addresses handling large amounts of data in the system:
    - whether data structures (e.g. queues, stacks, arrays, etc.) are large enough
    - Fields, records, and files are stressed to check if their size can accommodate all possible data volumes.
- Configuration Testing
  - Analyse system behaviour:
    - in various hardware and software configurations specified in the requirements
    - sometimes systems are built in various configurations for different users
    - for instance, a minimal system may serve a single user,
      - other configurations for additional users.

System Testing

- Compatibility Testing
  - These tests are needed when the system interfaces with other systems:
    - check whether the interface functions as required.
  - If a system is to communicate with a large database system to retrieve information:
    - a compatibility test examines speed and accuracy of retrieval.

System Testing

- Recovery Testing
  - These tests check response to:
  - presence of faults or to the loss of data, power, devices, or services
    - subject system to loss of resources
      - check if the system recovers properly.
- Maintenance Testing
  - Diagnostic tools and procedures:
    - help find source of problems.
    - It may be required to supply
      - memory maps
      - diagnostic programs
      - traces of transactions
      - circuit diagrams, etc.

System Testing

- Documentation Testing
  - Check that required documents exist and are consistent:
    - user guides,
    - maintenance guides,
    - technical documents
  - Sometimes requirements specify:
    - format and audience of specific documents
    - documents are evaluated for compliance
- Usability Testing
  - All aspect of user interfaces are tested:
    - Display screens
    - Report formats
    - Navigation and selection problems

Regression Testing

- Does not belong to either unit test, integration test, or system test.
  - In stead, it is a separate dimension to these three forms of testing.
- Regression testing is the running of test suite:
  - after each change to the system or after each bug fix
  - *ensures that no new bug has been introduced due to the change or the bug fix.*
- Regression tests assure:
  - the new system's performance is at least as good as the old system
  - always used during phased system development.

THANK YOU