CS 123
Introduction to Software Engineering

07B – Implementation and Integration

DISCS
SY 2013 - 2014

Overview

- Implementation and integration
- Testing during the implementation and integration phase
- Integration testing of graphical user interfaces
- Product testing
- Acceptance testing
- CASE tools for the implementation and integration phase
- CASE tools for the complete software process

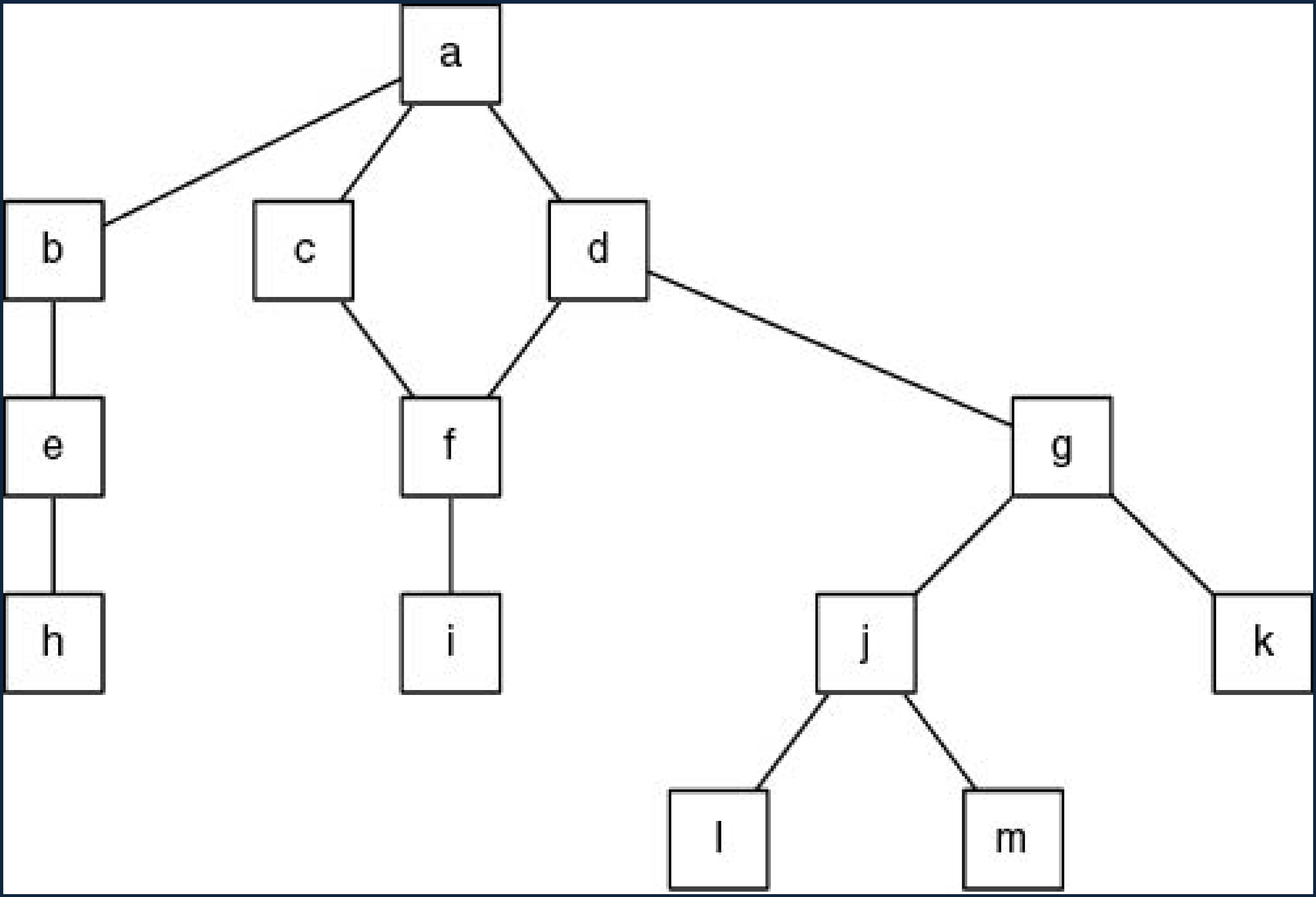
Overview (contd)

- Integrated environments
- Environments for business applications
- Public tool infrastructures
- Potential problems with environments
- Metrics for the implementation and integration phase
- Air Gourmet Case Study: Implementation and integration phase
- Challenges of the implementation and integration phase

Implementation and Integration Phase

- Up to now: Implementation followed by integration
  - Poor approach
- Better
  - Combine implementation and integration methodically

Product with 13 Modules



Implementation, Then Integration

- Code and test each module separately
- Link all 13 modules together, test product as a whole

Drivers and stubs

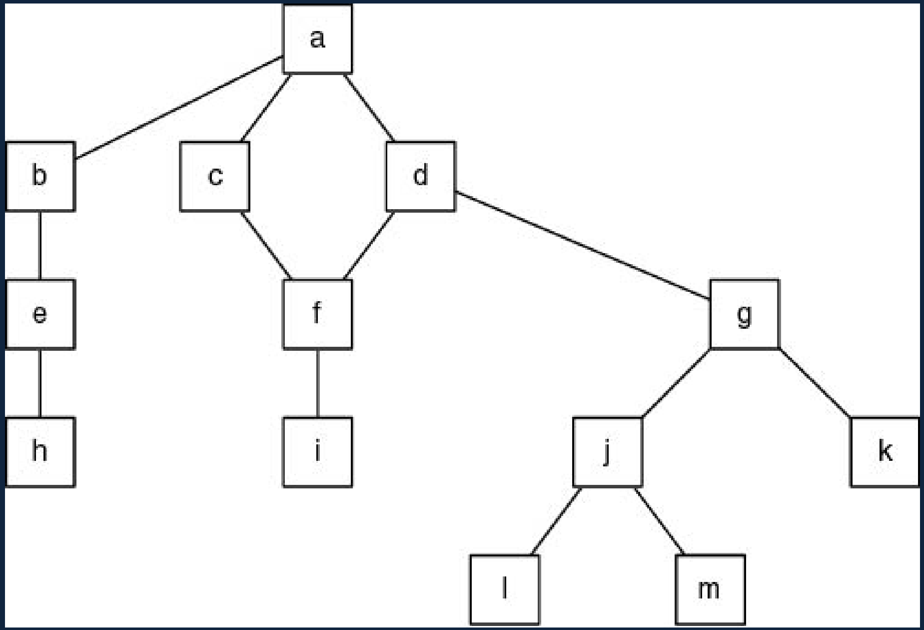
- To test module a, modules b, c, d must be stubs
  - Empty module, or
  - Prints message ("Procedure radarCalc called"), or
  - Returns precooked values from preplanned test cases
- To test module h on its own requires a driver, which calls it
  - Once, or
  - Several times, or
  - Many times, each time checking value returned
- Testing module d requires driver and two stubs

Implementation, Then Integration (contd)

- Problem 1
  - Stubs and drivers must be written, then thrown away after module testing is complete
- Problem 2
  - Lack of fault isolation
  - A fault could lie in *any* of 13 modules or 13 interfaces
  - In a large product with, say, 103 modules and 108 interfaces, there are 211 places where a fault might lie
- Solution to both problems
  - Combine module and integration testing
  - "Implementation and integration phase"

Top-down Implementation and Integration

- If module m1 calls module m2, then m1 is implemented and integrated before m2
- One possible top-down ordering is
  - a, b, c, d, e, f, g, h, i, j, k, l, m
- Another possible top-down ordering is
  - a
  - [a] b, e, h
  - [a] c, d, f, i
  - [a, d] g, j, k, l, m



Top-down Implementation and Integration

- Advantage 1: Fault isolation
  - Previously successful test case fails when mNew is added to what has been tested so far

- Advantage 2: Stubs not wasted
  - Stub expanded into corresponding complete module at appropriate step

Top-down Implementation and Integration

- Advantage 3: Major design flaws show up early
  - Logic modules include decision-making flow of control
    - In the example, modules a, b, c, d, g, j
  - Operational modules perform actual operations of module
    - In the example, modules e, f, h, i, k, l, m
- Logic modules are developed before operational modules

Top-down Implementation and Integration

- Problem 1
  - Reusable modules are not properly tested
  - Lower level (operational) modules are not tested frequently
  - The situation is aggravated if the product is well designed
- Defensive programming (fault shielding)
  - Example:

```
if (x >= 0)
        y = computeSquareRoot (x, errorFlag);
```

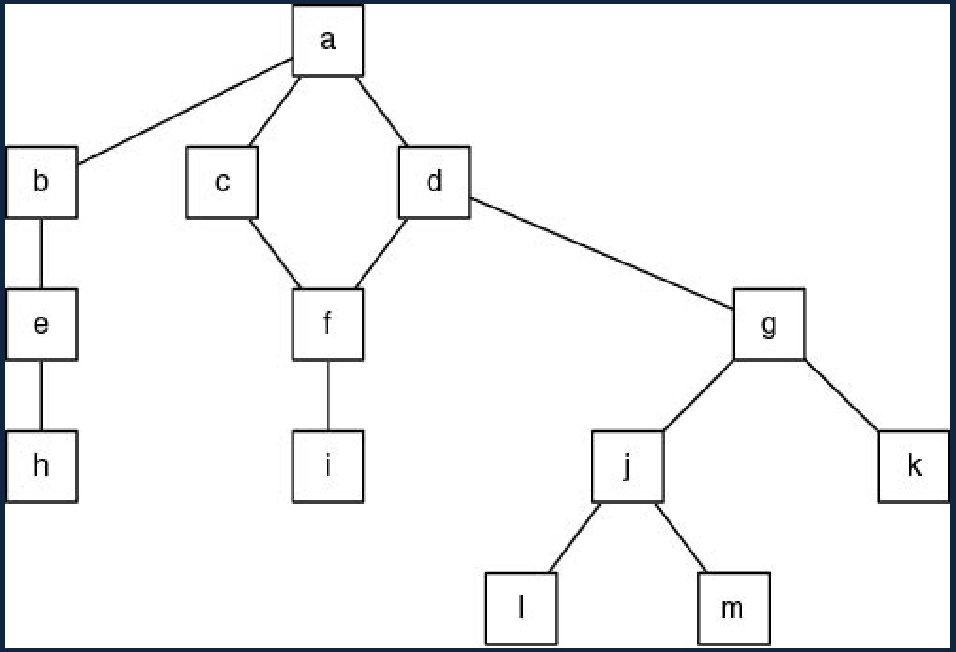
  - Never tested with x < 0
  - Reuse implications

Bottom-up Implementation and Integration

- If module m1 calls module m2, then m2 is implemented and integrated before m1
  - One possible bottom-up ordering is

    l, m, h, i, j, k, e, f, g, b, c, d, a
  - Another possible bottom-up ordering is

    h, e, b
    i, f, c, d
    l, m, j, k, g [d]
    a [b, c, d]



Bottom-up Implementation and Integration

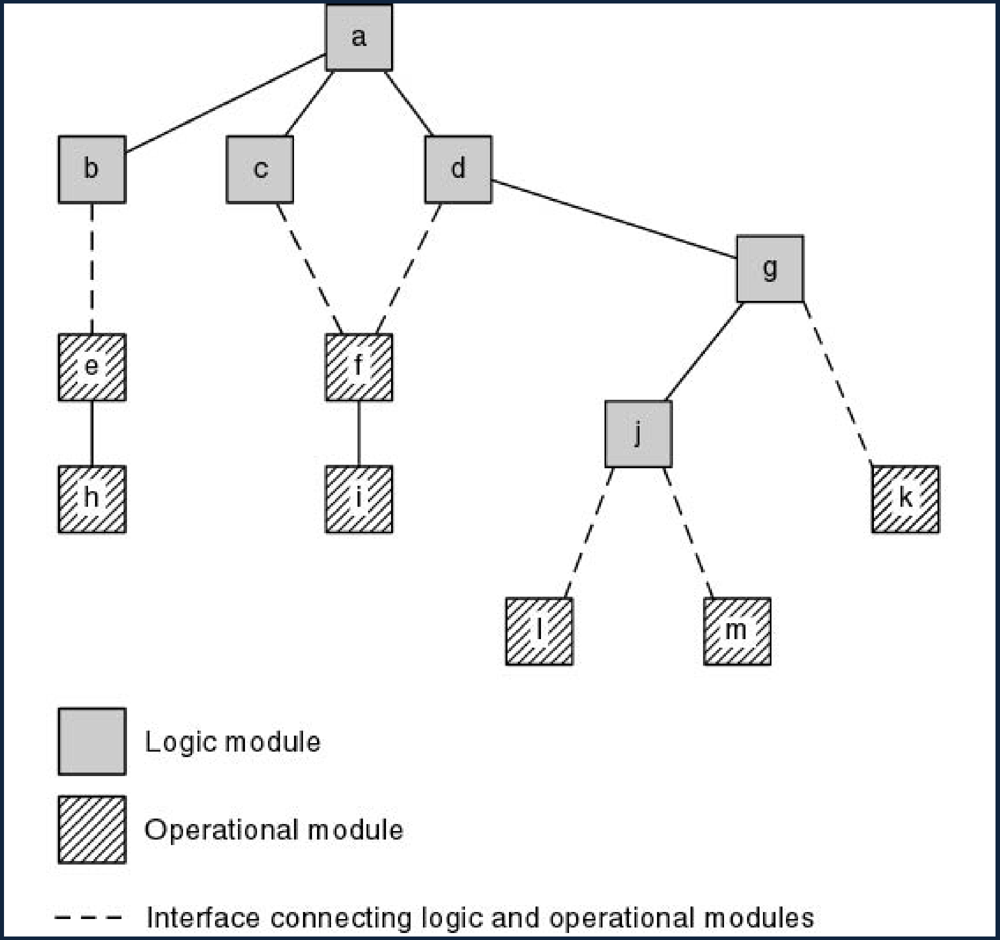
- Advantage 1
  - Operational modules thoroughly tested
- Advantage 2
  - Operational modules are tested with drivers, not by fault shielding, defensively programmed calling modules
- Advantage 3
  - Fault isolation

Bottom-up Implementation and Integration

- Difficulty 1
  - Major design faults are detected late
- Solution
  - Combine top-down and bottom-up strategies making use of their strengths and minimizing their weaknesses

Sandwich Implementation and Integration

- Logic modules are implemented and integrated top-down
- Operational modules are implemented and integrated bottom-up
- Finally, the interfaces between the two groups are tested



Sandwich Implementation and Integration (contd)

- Advantage 1
  - Major design faults are caught early
- Advantage 2
  - Operational modules are thoroughly tested
  - They may be reused with confidence
- Advantage 3
  - There is fault isolation at all times

Summary

Approach	Strengths	Weaknesses
Implementation then integration (Section 15.1)	—	No fault isolation Major design faults show up late
Top-down implementation and integration (Section 15.1.1)	Fault isolation Major design faults show up early	Potentially reusable modules are not adequately tested
Bottom-up implementation and integration (Section 15.1.2)	Fault isolation Potentially reusable modules are adequately tested	Major design faults show up late
Sandwich implementation and integration (Section 15.1.3)	Fault isolation Major design faults show up early Potentially reusable modules are adequately tested	—

Object-Oriented Implementation and Integration

- Object-oriented implementation and integration
  - Almost always sandwich implementation and integration
  - Objects are integrated bottom-up
  - Other modules are integrated top-down

Management Issues during Implementation and Integration

- Example
  - Design document used by Team 1 (who coded module m1) shows m1 calls m2 passing 4 parameters
  - Design document used by Team 2 (who coded module m2) states clearly that m2 has only 3 parameters
- Solution
  - The implementation and integration. process must be run by SQA

Testing during Implem. and Integration Phase

- Product testing
  - Performed to ensure that the product will not fail its acceptance test
- Failing the acceptance test is a disaster for management
  - The client may conclude that the developers are incompetent
  - Worse, the client may believe that the developers tried to cheat
- The SQA team must ensure that the product passes its acceptance test

Integration Testing of Graphical User Interfaces

- GUI test cases:
  - Mouse clicks
  - Key presses
  - And so on
- Cannot be stored in the usual way
- We need special CASE tools
- Examples:
  - QAPartner
  - XRunner

Strategy for Product Testing

- The SQA team must try to approximate the acceptance test
- Black box test cases for the product as a whole
- Robustness of product as a whole
  - Stress testing (under peak load)
  - Volume testing (e.g., can it handle large input files?)

Strategy for Product Testing (contd)

- Check all constraints
  - Timing constraints
  - Storage constraints
  - Security constraints
  - Compatibility
- Review all documentation to be handed over to the client
- Verify the documentation against the product
- The product (software plus documentation) is now handed over to the client organization for acceptance testing

Acceptance Testing

- The client determines whether the product satisfies its specifications
- Performed by
  - The client organization, or
  - The SQA team in the presence of client representatives, or
  - An independent SQA team hired by the client

Acceptance Testing (contd)

- Four major components of acceptance testing
  - Correctness
  - Robustness
  - Performance
  - Documentation
- (Precisely what was done by developer during product testing)

CASE tools for the Implem. and Integ. Phase

- Bare minimum:
  - Version control tools
  - Examples:
    - *rcs*, *sccs*, PCVS, SourceSafe

CASE Tools for the Complete Software Process

- A large organization needs an environment
- A medium-sized organization can probably manage with a workbench
- A small organization can usually manage with just tools.

Integrated Environments

- Usual meaning of “integrated”
  - User interface integration
  - Similar “look and feel”
  - Most successful on the Macintosh
- There are also other types of integration

Process Integration

- Environment supports one specific process
- Subset: Technique-based environment
  - Formerly: "method-based environment"
  - Supports specific technique
  - Examples:
    - Structured systems analysis
    - Petri nets
  - Usually comprises
    - Graphical support for specification, design phases
    - Data dictionary
    - Some consistency checking
    - Some management support
  - Support and formalization of manual processes
  - Examples:
    - Analyst/Designer, Rose, Rhapsody (for Statecharts)

Technique-Based Environments (contd)

- Advantage of technique-based environment
  - Forced to use specific method, correctly
- Disadvantages of technique-based environment
  - Forced to use specific method
  - Inadequate support of programming-in-the-many

Environments for Business Application

- The emphasis is on ease of use
  - GUI
  - Standard forms for input, output
  - Code generator
    - Detailed design is lowest level of abstraction
    - Expect productivity rise
- Examples:
  - Foundation, Bachman Product Set

Public Tool Infrastructure

- PCTE—Portable common tool environment
  - NOT an environment
  - An infrastructure for supporting CASE tools (similar to the way an operating system provides services for user products)
  - Adopted by ECMA (European Computer Manufacturers Association)
- Example implementations:
  - IBM, Emeraude

Potential Problems with Environments

- No one environment is ideal for all organizations
  - Each has its strengths and weaknesses
- Warning 1
  - Choosing the wrong environment can be worse than no environment
  - Enforcing a wrong technique is counterproductive
- Warning 2
  - Shun environments below CMM level 3
  - We cannot automate a nonexistent process
  - A CASE tool or CASE workbench is fine

Challenges of the Implem. and Integration Phase

- Management issues are paramount here
  - Appropriate CASE tools
  - Test case planning
  - Communicating changes to all personnel
  - Deciding when to stop testing