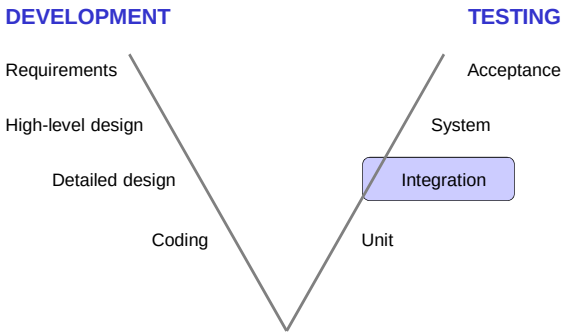
13. Integration testing

Units, modules and functional decomposition
Big Bang, top-down & bottom-up testing
Path based testing
Incremental integration and regression testing
System Requirements Specification based testing

Life cycle model for testing



Integration testing

- Process of examining how the pieces of a system (Modules) work together, especially at the interfaces
- Tests to ensure that the various components of a system
  - Interact correctly
  - Pass data correctly
  - Function cohesively

Test planning issues

- Functional decomposition
  - What are "pieces" of the system (functional sub-assemblies; Modules)?
  - What units should be assembled and tested together as a Module?
- Testing strategies
  - The order of testing
  - Keeping track of faults

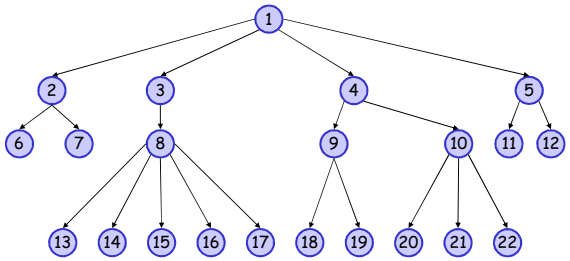
Units, Modules and functional decomposition

- **Unit** – the smallest testable piece of code
  - Has one or several inputs
  - Usually has one output
  - May not have any meaning in context of a specification (e.g. it changes a state of a variable or internal switch)
- **Module** – the smallest testable piece of code for which all inputs and outputs are meaningful at the specification level
  - May be made up of several Units
- Modules can be tested using functional testing; Units may not be

Units, Modules and functional decomposition

- Functional decomposition can be derived from a number of models, for example
  - Context diagram
  - Data flow diagram
  - Entity / relationship model
  - Finite state machine model

Example of a functional decomposition tree

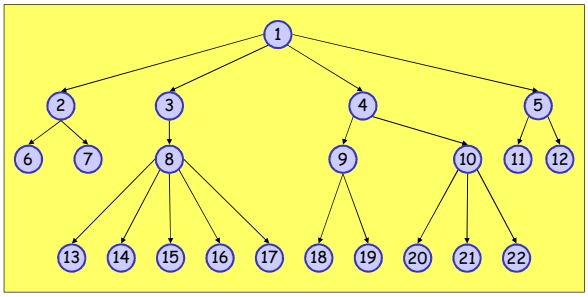


Testing strategies

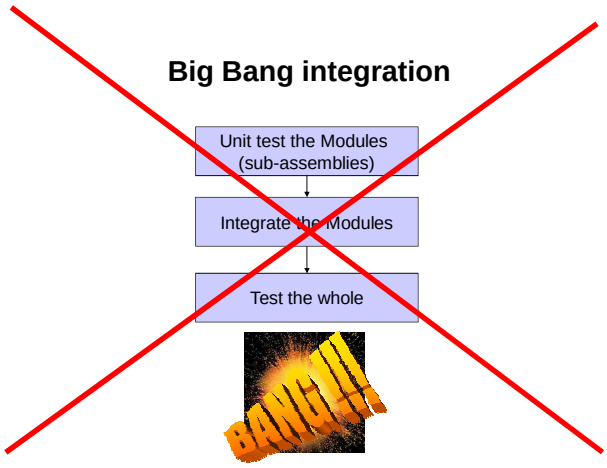
- Big bang integration
- Top-down integration
- Bottom-up integration
- Pair-wise integration
- Neighbourhood integration
- Path-based integration

- All assume that all Modules have been Unit tested

Big Bang integration



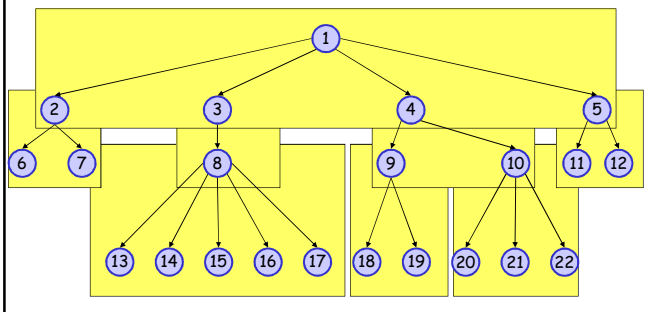
Big Bang integration



Top-down integration

- Begins with the main program Module (the root node of the decomposition tree)
- Any lower level Modules called by the main Module appears as a **stub**
- Carry out testing until satisfactory
- Gradually replace the stubs with the actual code and re-test

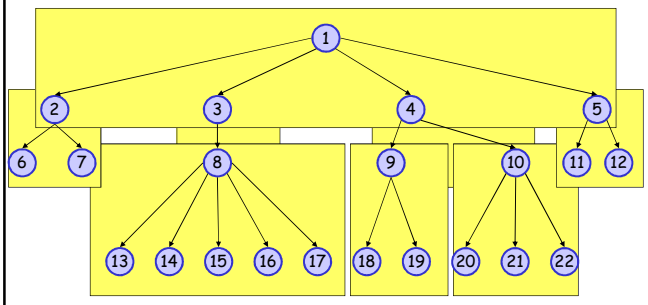
Top-down integration



Bottom-up integration

- Start with the Modules which are leaf nodes of the decomposition tree
- For each set of leaf nodes, code a **driver** at the level of the parent node of these nodes
- Carry out testing until satisfactory
- Gradually replace the drivers with the actual code and re-test

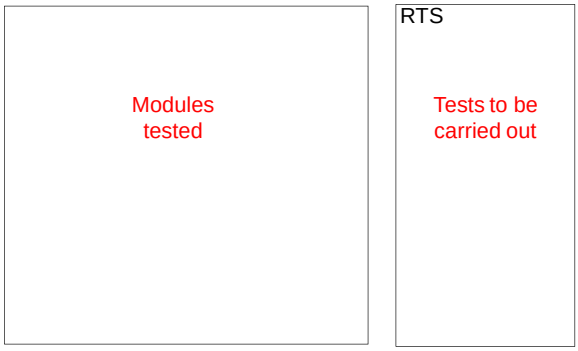
Bottom-up integration



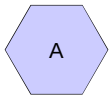
Incremental integration and regression testing

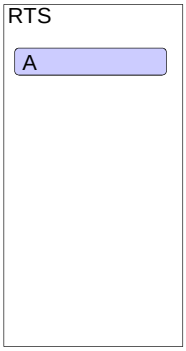
- Start with one Module
  - Root, if using a top - down scheme
  - Leaf, if using a bottom - up scheme
- Start a Regression Test Set (RTS) to hold tests
- As each further Module is added, use tests to check that
  - The new Module works alone
  - The new Module perform its function when connected to established working Modules
  - Adding the Module did not break anything else

Incremental integration and regression testing



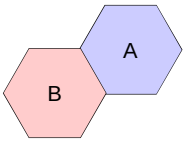
Incremental integration and regression testing

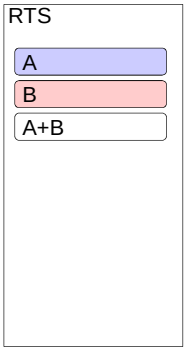




Module A arrives, along with its unit tests

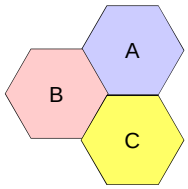
Incremental integration and regression testing





Module B is added; it has its set of unit tests

Incremental integration and regression testing

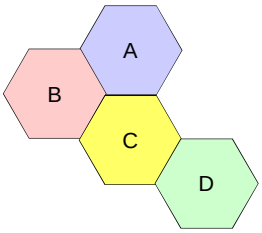


RTS

- A
- B
- A+B
- C
- A+B+C

Module C is added together with its set of unit tests

Incremental integration and regression testing



RTS

- A
- B
- A+B
- C
- A+B+C
- D
- A+B+C+D

Module D is added together with its set of unit tests

Benefits

- We **always** have a system that works
  - we just keep adding functionality
- It is relatively easy to find and fix problems
  - If everything works until we add Module X and then Module A stops working, the problem is likely to only a limited number of causes
- On the completion, we have a complete Regression Test Set
  - RTS is invaluable for future system maintenance and further enhancements

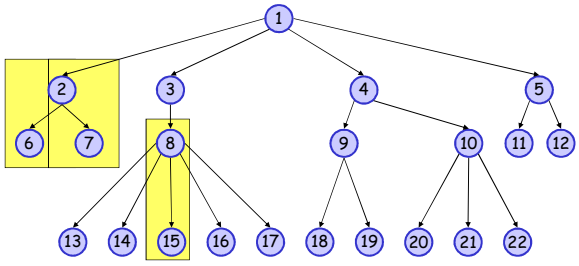
Top-down or bottom-up?

- Pros – discussed above
- Cons
  - Suits well the Waterfall development model, but may not be a most natural method for other models
  - Testing effort may be large due to the need to write stubs and drivers

Pair-wise integration

- Similar to top-down or bottom-up integration
- Uses real code instead of stubs / drivers

Pair-wise integration



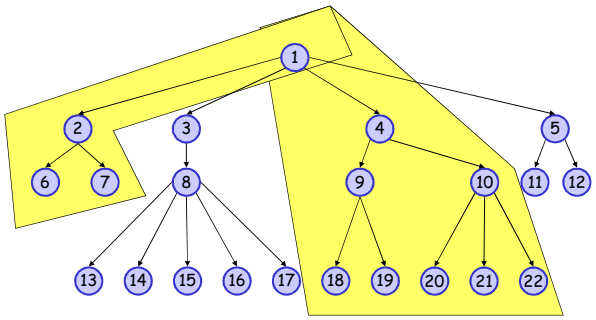
Pair-wise integration

- Reduces testing effort (no need to write stubs / drivers)
- If not done carefully, it may deteriorate into Big Bang Testing

Neighbourhood integration

- Typical top-down or bottom-up strategy testing selects nodes in the **breadth-first** order
- Neighbourhood integration selects nodes in the **depth-first** order
- Avoids the need to write stubs / drivers
- If not done with care, it may deteriorate into Medium Bang Testing

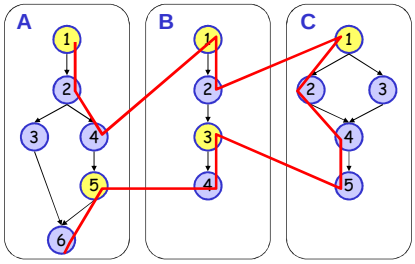
Example of a functional decomposition tree



Path-based integration

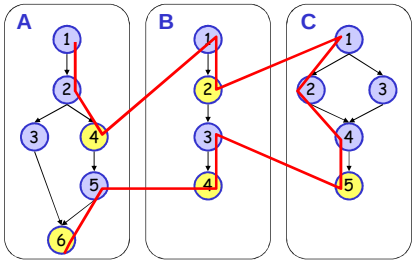
- A **module execution path** is a sequence of statements that begins with a source node and ends with a sink node
- A **message** is a programming language mechanism by which one unit transfers control to another unit
  - Examples: procedure call, function references
- An **MM-path** is an interleaved sequence of module execution paths and messages

MM-paths - example



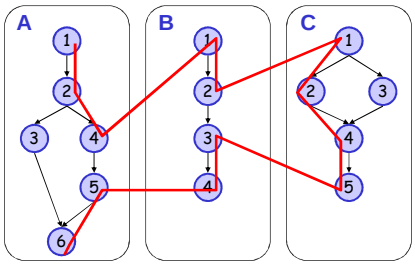
Source nodes
A: 1,5
B: 1,3
C: 1

MM-paths - example



Sink nodes
A: 4,6
B: 2,4
C: 5

MM-paths - example

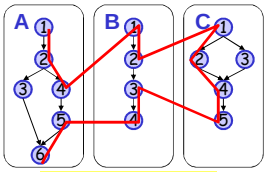


Module Execution paths:
A1: 1,2,3,6
A2: 1,2,4
A3: 5,6
B1: 1,2
B2: 3,4
C1: 1,2,4,5
C2: 1,3,4,5

MM-path graph

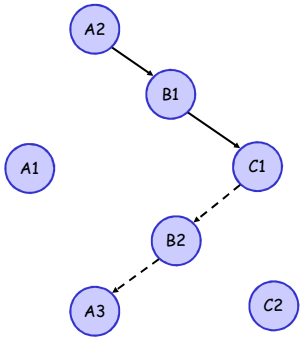
Module Execution Paths:
A1: 1,2,3,6
A2: 1,2,4
A3: 5,6
B1: 1,2
B2: 3,4
C1: 1,2,4,5
C2: 1,3,4,5

MM-path graph is the directed Graph in which nodes are Module Execution Paths and edges correspond to messages and returns from one unit to another



MM-path graph

Module Execution Paths:
A1: 1,2,3,6
A2: 1,2,4
A3: 5,6
B1: 1,2
B2: 3,4
C1: 1,2,4,5
C2: 1,3,4,5



Path-based integration testing

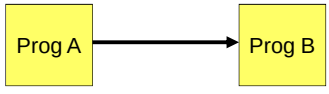
- Path-based integration proceeds along MM-paths
- The principles of the incremental integration should be applied here as well
- Path based testing combines merits of functional and structural testing
  - Functional: actions with inputs and outputs
  - Structural: the method of path identification
- More effort is needed to identify MM-paths
- But no need to write stubs and drivers

Test cases

- Based on functional specification / design
- Generated using functional specification methods

Prior to testing …

- Check interfaces between the Modules
- Interfaces are major places where things go wrong



(A,B,C,D,E) (A,B,D,C)

Prior to testing …

- Check for
  - Number of parameters
  - Whether parameters are required or optional
  - Order of parameters
  - Parameter type / data format
  - Length
  - Defintion
- Some incompatibilities may be detected by a compiler

Next lecture

System testing

Further reading

- Craig and Jaskiel “Systematic Software Testing, relevant sections of Chapter 4
- Other books on testing have good coverage of the Integration Testing

Homework

- Given a functional decomposition of the program NextDate in the form of pseudo-code
  - Draw a functional decomposition graph
  - List MM-paths
  - Draw the MM-path graph for one of the paths

  The pseudo-code can be found on the Module web page
- Ensure that you are familiar with the following software modelling methods:
  - Context diagram
  - Data flow diagram
  - Entity / relationship model
  - Finite state machine model