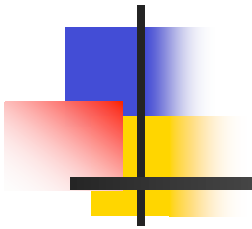
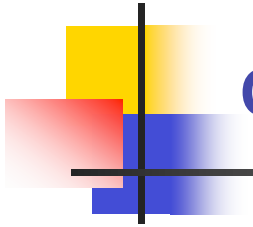


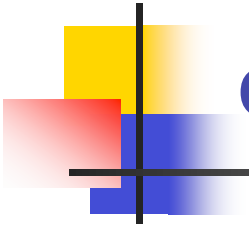
OO Integration Testing

Chapter 18



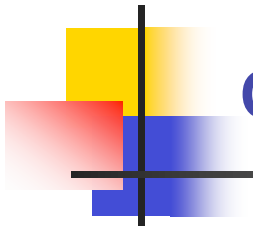
Questions

- What assumption is made for integration testing?
- How does OO unit definition alter integration testing?
- What choices do we have with integration testing?
- What information is needed for integration testing?



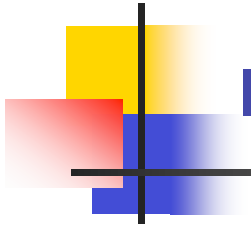
Overview

- Assume unit level testing is complete
- For OO have two choices for unit
  - **For method is a unit**
    - **Need to integrate within the class**
      - Does occur with classes that have multiple designers / implementers
    - **Need to integrate classes**
  - **For class is a unit**
    - **Need to unflatten classes**
    - **Need to remove test methods**
    - **Need to integrate classes**



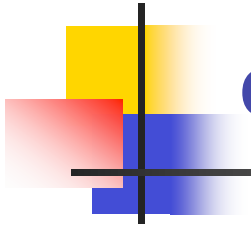
Overview – 2

- Static and dynamic choices
- Address polymorphism statically
  - **Select a test for each polymorphic context**
- Dynamic view is more challenging



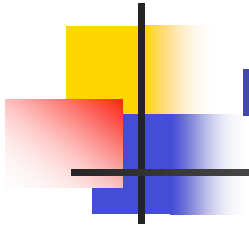
Information environment

- Class definitions
  - **Source text**
- Static model
  - **Inheritance & uses structure**
- Dynamic model
  - **Use cases – scenarios**
  - **Finite state machines – Petri nets**
  - **Class communication – message passing**
  - **Statecharts are not useful**



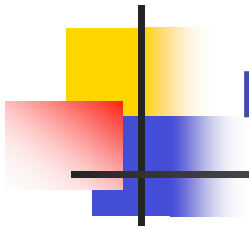
Class communication

- Collaboration diagrams
  - **Annotated call graph – Figure 18.1**
  - **Supports both pair wise and neighbourhood integration strategies**
- Sequence diagrams
  - **Finite state machines with time axis – Figure 18.2**
    - **States**
      - Classes – regular grain
      - Methods – fine grain
    - **Transitions correspond to sending messages**
  - **Close analogy with MM-paths**



Integration types

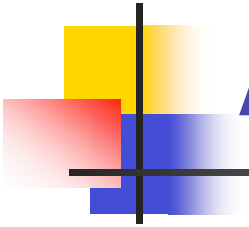
- Pair-wise
  - **Too much extra work with stubs and drivers**
- Neighbourhood
  - **Using collaboration diagrams can cause problems**
    - **Some neighbourhoods may include most classes**
    - **Some neighbourhoods may be only two classes**
    - **Need better definition**
  - **Centers of a graph – Ultra-center**
    - **Minimize maximum distance to other nodes**
      - Analogy with ripples from dropping an object into water
    - **Neighbourhood grows from center**
      - Less stubs
      - Less diagnostic precision



MM-paths (method to message paths)

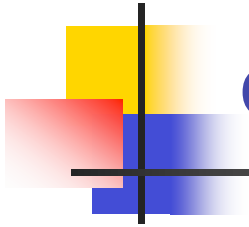
- MM-path in OO
  - **A sequence of method executions linked by messages**
    - **Start at any class**
    - **End at message quiescence**
      - At class that does not send any messages
  - **Largest integration level**
    - **Classes that implement an atomic system function**
    - **Stimulus / response pair of port-level events**

See Figures 18.3, 18.4, 18.5



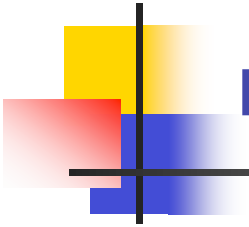
Atomic System Function

- An MM-path
  - **Also called stimulus-response path**
  - **Begins with an input port event**
  - **Ends with an output port event**
- Begin and end at event quiescence
- Addresses event-driven nature of OO programs
- At the boundary of integration and system testing



OO-calendar analysis

- Directed graph makes it possible to be analytical in choosing test case
- How many test cases are there
  - **Cyclomatic complexity is 23**
  - **Lower bound could be 3 test cases**
    - **Start at each of the three statements in routine testIt**
  - **Depends upon choice of test cases, which could miss leap year related cases**
  - **Need to cover every message**
    - **The 13 decision-table based identified in decision table testing (Table 7.16) would give a good integration test suite**
    - **Look for test cases to cover every message in Figure 18.3**

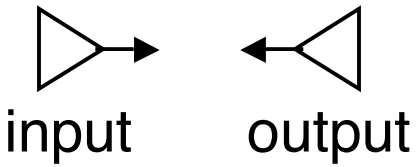


Data flow testing

- MM-paths, like DD-paths, are insufficient
- Data values add complexity
  - **Come from inheritance**
  - **Come from stages of message passing**
- Program graphs are basis but are too simple
  - **Need event and message driven Petri nets**

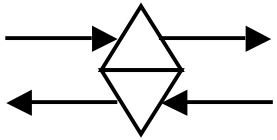
Event & Message driven Petri nets (EMDPN)

- P – set of port events

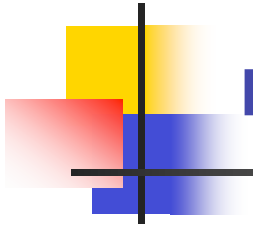


- D – set of data places

- M – message send/return places
  - **Output for sender**
  - **Input for receiver**

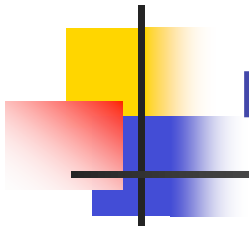


- T – set of transitions
  - **Represent a method execution path**



EMDPN – 2

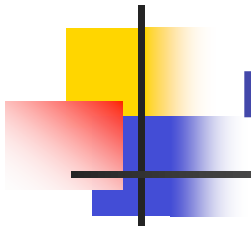
- In – set of edges to transitions
  - **(P ∪ D ∪ M) ↔ T**
    - **It is a relation between places and transitions**
    - **If deterministic then it is a function from places to transitions**
- Out – set of edges from transitions
  - **T ↔ (P ∪ D ∪ M)**



Message send/receive places

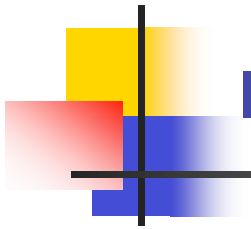
- Capture notion of **interobject** messages
  - **They are an output of a method execution path in the sending object**
  - **They are an input to a method execution path in the receiving object**
  - **The return is an output of a method execution path in the receiving object**
  - **The return is an input to a method execution path in the sending object**

See Figure 18.7



DU-paths

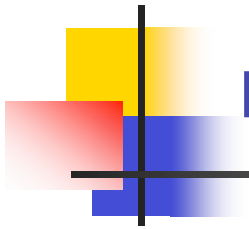
- Define / use paths
  - **Focus on connectivity**
  - **Ignore types of nodes**



Inheritance-induced data flow

- Begins with a data place
- Ends with a data place
- Data places alternate with isA transitions
  - **isA transitions are degenerate execution paths**
    - **Implement inheritance**

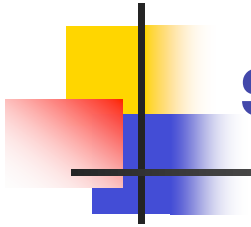
See Figure 18.8



Message-induced data flow

- Set of transitions
  - **Start with defining transition**
    - **Variable is defined in the module execution path**
  - **End with use transition**
    - **Variable Is used in the module execution path**
- Can be definition clear or not definition clear

See Figure 18.9
&
Section 18.3.3 for an example path



Slices

- Useful if executable
  - **Difficult to do in OO environment**
- Can be used for desk checking for fault location