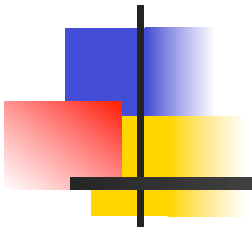
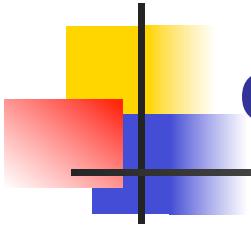


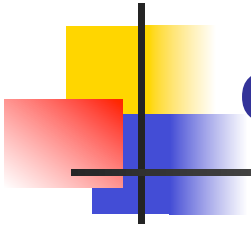
Issues in OO Testing

Chapter 16



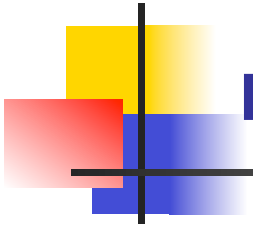
OO context

- OO based on hope that objects could be reused without
  - **Modification**
  - **Additional testing**
    - **Based on notion that objects encapsulate functions and data that belong together**
- Consensus now is that such optimism is unwarranted
  - **OO programs has more severe testing problems than traditional programs**



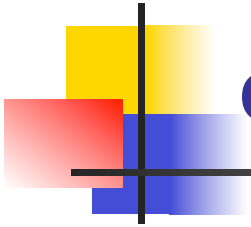
OO context – 2

- Looking to other models that can be combined with OO to ameliorate the problems
  - **Aspect-oriented programs**
    - **Aspect-orientation can be combined with any programming language**



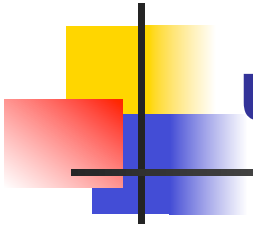
Problems to address

- Levels of testing
  - **What is a unit?**
- Implications of composition strategy of OO
  - **Compare to functional decomposition**
  - **OO programs**
    - **Inheritance, encapsulation and polymorphism**
    - **How can traditional testing be extended?**



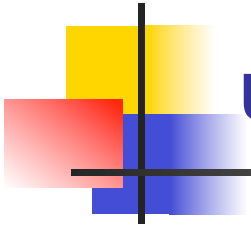
OO unit

- Two definitions
  - **A unit is the smallest program component that can be compiled and executed**
  - **A unit is a program component that would be developed by one person**
    - **Could be a sub-part of one class**



Unit is 1-person development

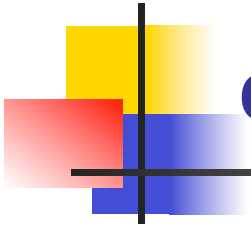
- Traditional testing works well
- Shifts much of the burden of testing to the integration level
- Does not take encapsulation into account
  - **Know about themselves**
  - **Operate on their own**



Unit is compilable & executable

- Can describe behaviour
  - **Model with FSM – Statechart**
  - **Very useful for identifying test cases**

- Integration testing is easier
  - **Integrate by combining already tested classes**
  - **Similar to traditional testing**

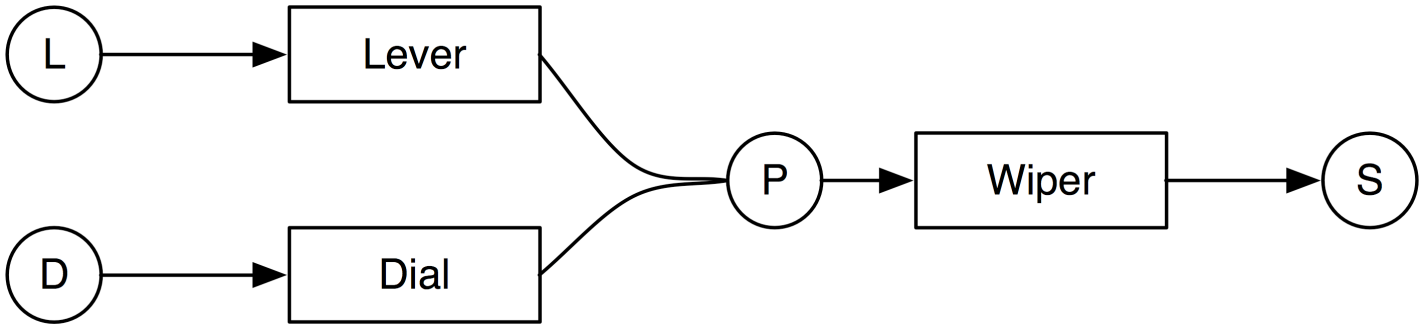


Composition & Encapsulation

- A class may be combined with other unknown classes
  - **Goal of reuse**
  - **Need high cohesion, low coupling**
- Need very good unit testing
- Reality is that burden of testing is still on integration testing

SWW– SSD 1

System Specification Diagram

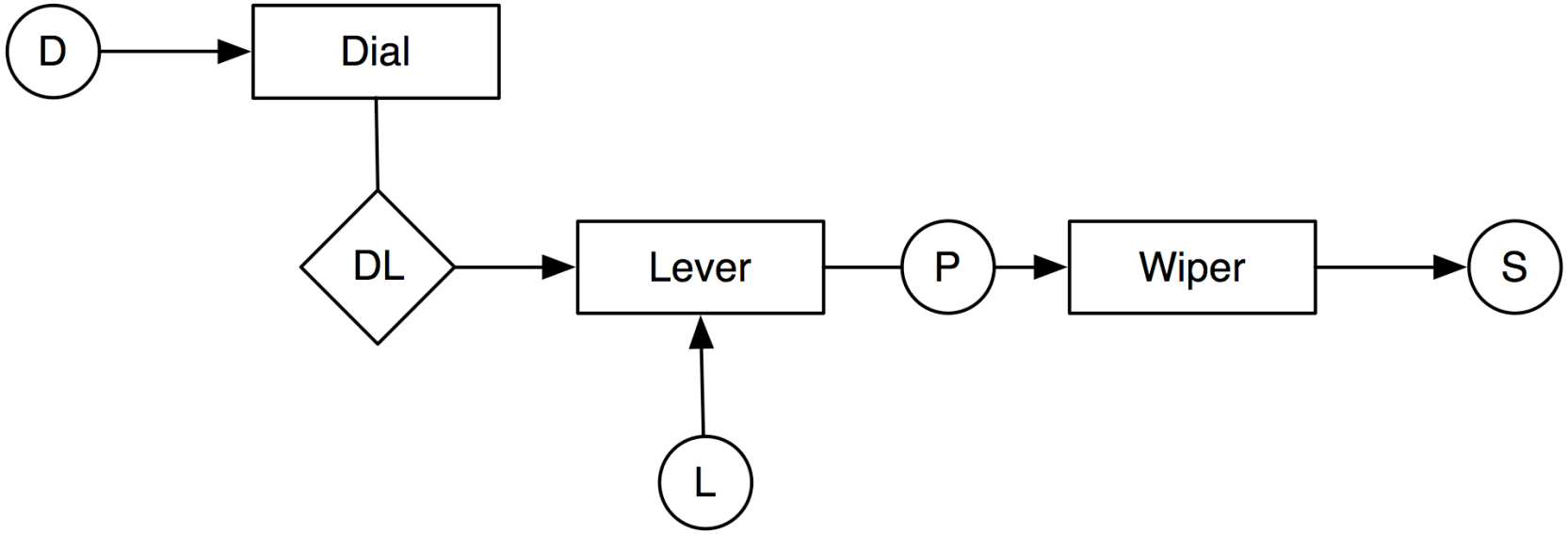


All communication channels are data stream

Channel P is a rough merge of the data streams from Lever and Dial

Low coupling between Wiper Lever and Dial

SWW– SSD 2

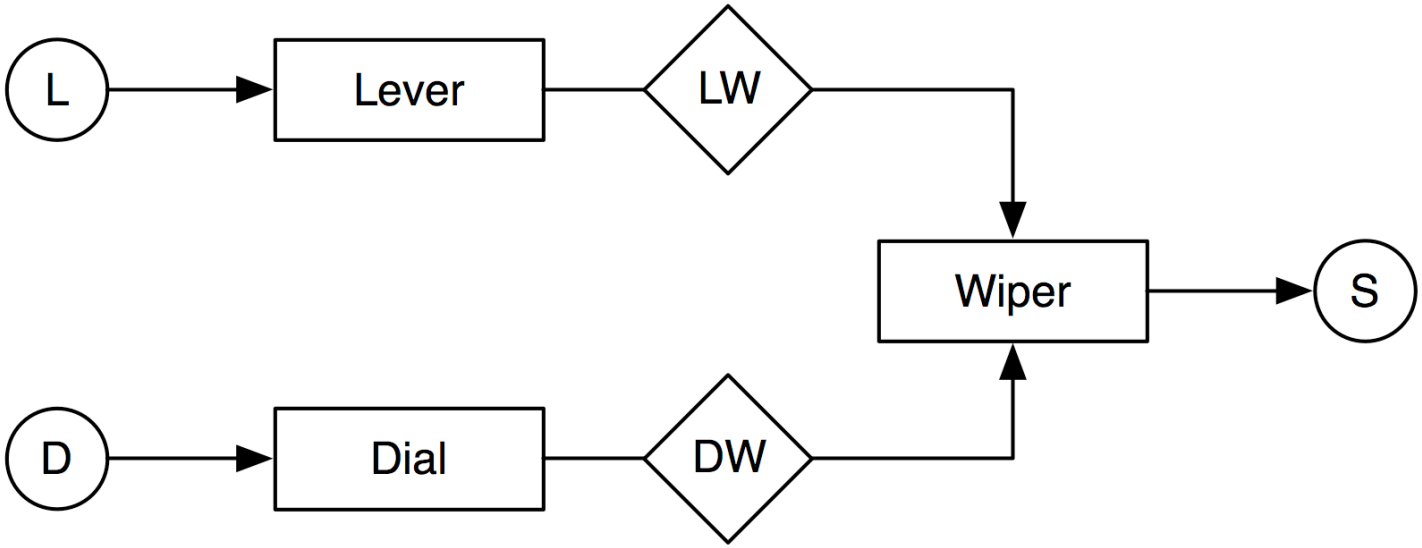


Channel DL is a statevector read of Dial by Lever

High coupling between Lever and Dial

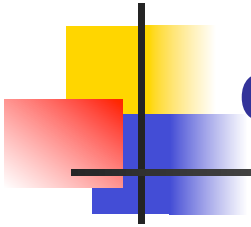
When does Lever read DL?

SWW – SSD 3



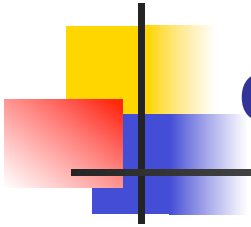
High coupling between
Wiper and Lever
Wiper and Dial

When does Wiper read LW and DW?



Complication of inheritance

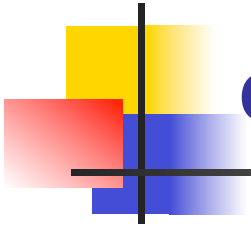
- Unit is more difficult to define when inheritance is involved
  - **Suggestion is to use the flat definition**
    - **Becomes complicated with multiple inheritance**
- Flattening solves inheritance problem
  - **Flattened classes are not a part of the system**
    - **Cannot be certain they are properly tested**



Complication of inheritance – 2

- May not have necessary methods for testing
  - **Can add test methods**
    - **Should they be a part of the delivered system?**
      - Analogous to having instrumented program text
    - **Test methods need to be tested !!! …**

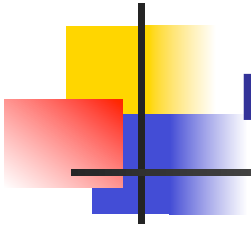
See Figures 16.2 & 16.3



Complication of polymorphism

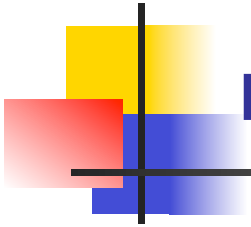
- Testing with different objects
  - **Redundant tests on inherited methods**
    - **Lose hoped for economies**

- Similarly testing polymorphism introduces redundant testing



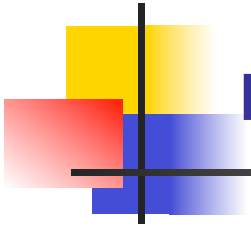
Levels of testing – Methods are units

- Four levels
  - **Method**
    - **Unit testing**
  - **Class**
    - **Intraclass integration testing**
  - **Integration**
    - **Interclass integration testing**
  - **System**
    - **At port level – same as traditional testing**



Levels of testing – Classes are units

- Three levels
  - **Class**
    - **Unit testing**
  - **Integration**
    - **Interclass testing**
  - **System**
    - **At port level**



Dataflow testing

- Need analogue to dataflow testing of units in traditional programs
  - **Use a revised Petri net definition to handle method calls between classes**
    - **See Chapter 18, OO-integration testing**