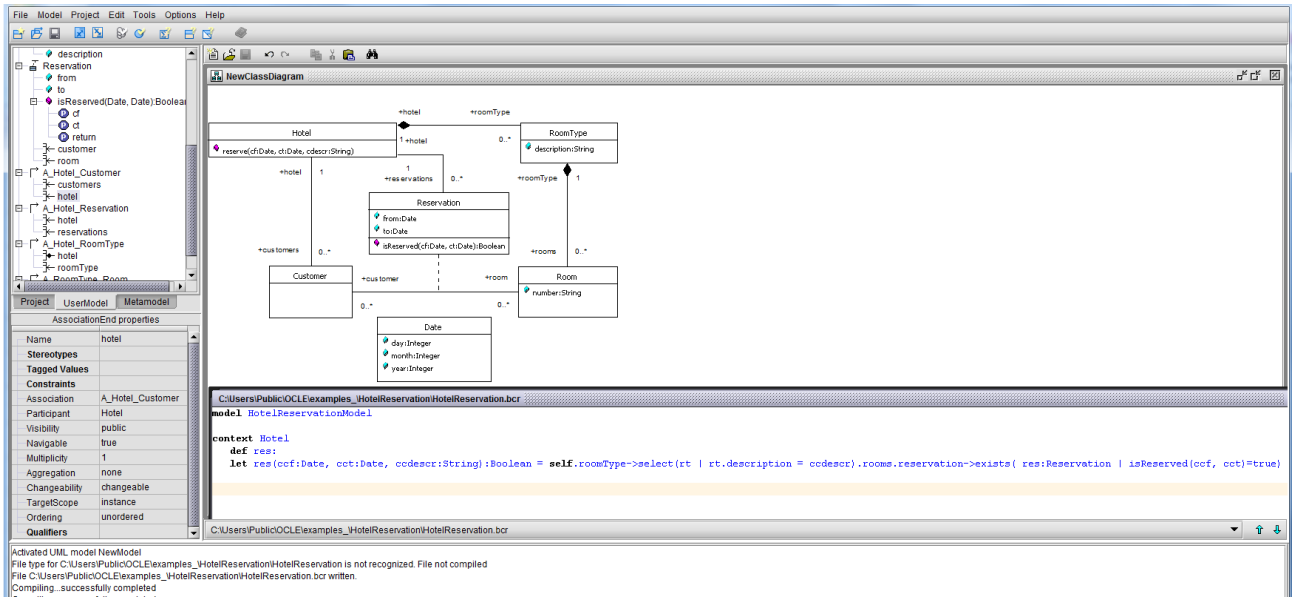
SE exam - 19th of June, 2015

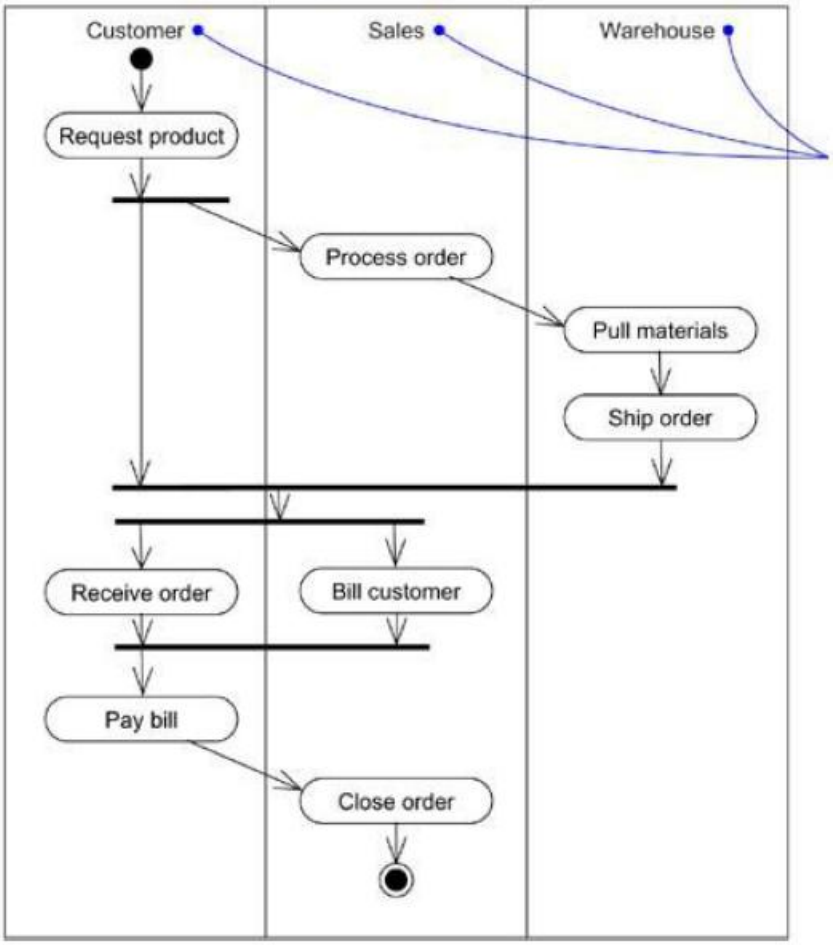
I. A Hotel has different types of rooms. When a potential client intents to reserve a room, then it specifies the interval [from:Date ... to:Date] and the type of room he wants to reserve. On the hotel site, each type of room has a textual description. A software system receives the client request and answer if the reservation is possible. In other words, if in the interval mentioned there is at least a room of the type mentioned not occupied.

a. By means of a UML class diagram, please specify a UML model supporting the above mentioned functionality. 2 pt

b. in the context of the Hotel class, please specify, using OCL, an observer returning a Boolean value; (true if the reservation is possible and false if it's not). In the context of the Reservation class there is implemented an observer `isReserved(cf:Date, ct:Date):Boolean` that returns true if the room is not reserved any day in the interval[cf:Date ... ctDate]. 1 pt



II. Please mention the kind of the diagram bellow, naming the UML concepts used in this diagram and explain the behavior described/specified. 1 pt



It's an activity diagram containing three swim lanes, each corresponding to the actor doing appropriate activities. Apart of activities, and of pseudo-states, there are simple transitions, fork and join transitions. After the Customer Request the product, the Sales Process the order. Once the Process order is finished, the Warehouse Pull materials and Skip order. After Skip order was executed, the Customer will Receive order and the Sales compartment will emit the Bill customer. After the two last actions will be realized, the Customer will Pay bill. Once this activity finished, the Sales compartment will Close order.

III. What describes the analysis model and what describes the design model of a problem? What kind of correspondence is there between the above mentioned models? Which of them is the most stable and why? 1 pt

The analysis model describes the concepts of the problem domain and the relationships existent among these concepts. The design model describes a possible solution of the problem. Usually there are many design models, therefore the relationship is one to many. The analysis model is the most stable because the domain concepts and their relationships change slowly compared with technologies, techniques and platforms.

IV. In the context of optimizing the design model:
   a. What do you mean by object collapsing? Please exemplify your description by means of a class diagram.

After the object model is restructured and optimized a couple of times, some of its classes may have few attributes or behaviors left. Such classes, when associated only with one other class, can be collapsed into an attribute, thus reducing the overall complexity of the model. The decision of collapsing classes is not always obvious. In the case of a social security system, the SocialSecurity class may have much more behavior, such as specialized routines for generating new numbers based on birth dates and the location of the original application. In general, developers should delay collapsing decisions until the beginning of the implementation, when responsibilities for each class are clear. Often, this occurs after substantial coding has occurred, in which case it may be necessary to refactor the code.

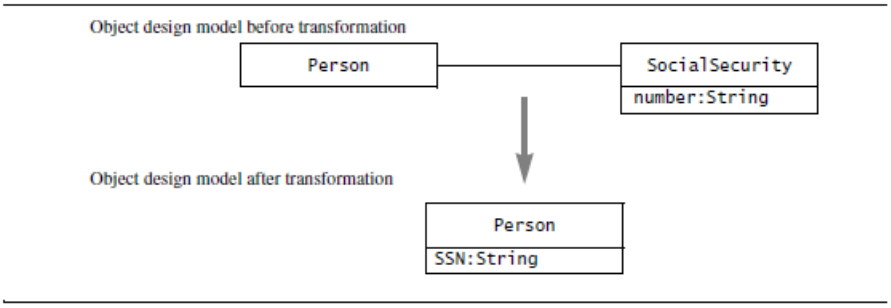


Figure 10-6 Collapsing an object without interesting behavior into an attribute (UML class diagram).

1 pt

b. What do you mean by delaying expensive computations? Please exemplify your description by means of a class diagram and mention the design pattern used. 1.5 pt

Often, specific objects are expensive to create. However, their creation can often be delayed until their actual content is needed. For example, consider an object representing an image stored as a file (e.g., an ARENA AdvertisementBanner). Loading all the pixels that constitute the image from the file is expensive. However, the image data need not be loaded until the image is displayed. We can realize such an optimization using a **Proxy design pattern** [Gamma et al., 1994]. An ImageProxy object takes the place of the Image and provides the same interface as the Image object (Figure 10-7). Simple operations such as width() and height() are handled by ImageProxy. When Image needs to be drawn, however, ImageProxy loads the data from disk and creates a RealImage object. If the client does not invokes the paint() operation, the RealImage object is not created, thus saving substantial computation time. The calling classes only access the ImageProxy and the RealImage through the Image interface.

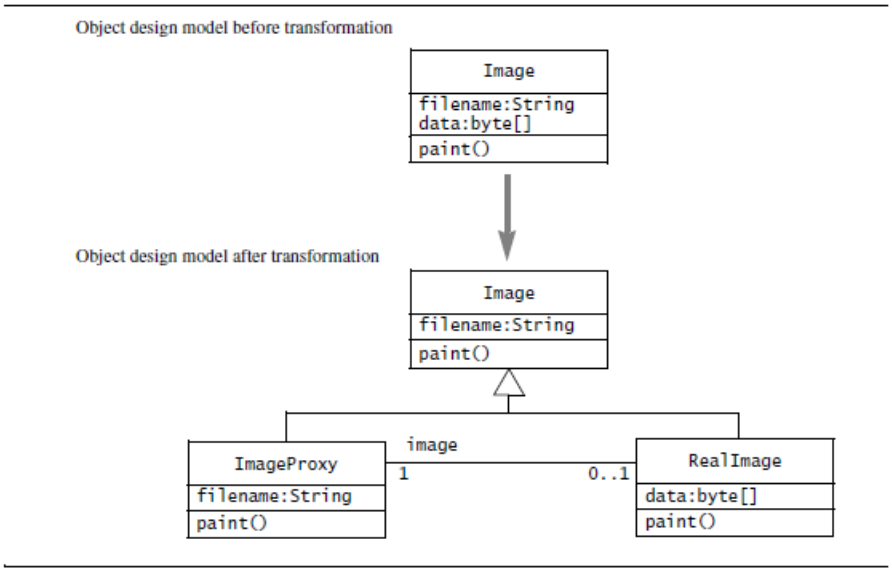


Figure 10-7 Delaying expensive computations to transform the object design model using a Proxy design pattern (UML class diagram).

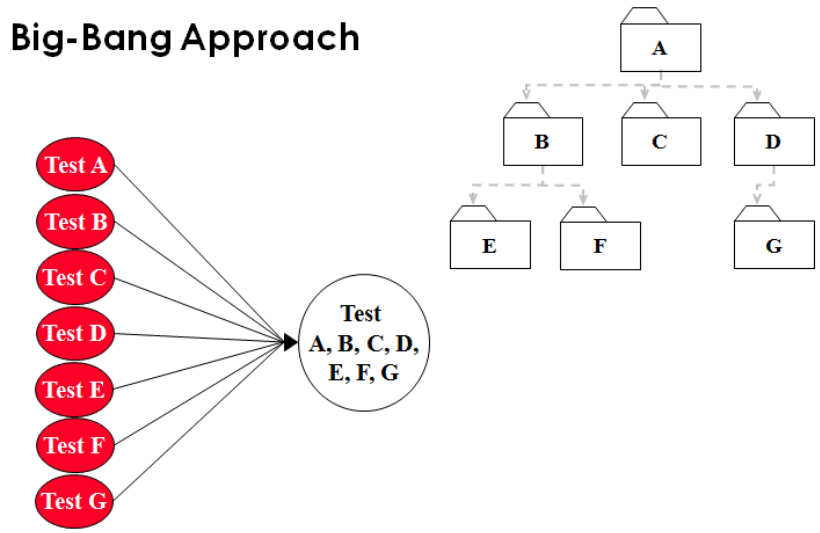
V. What do you mean .by integration testing? Please mention the main goal of integration testing. Please mention the testing strategies you know and describe any two of them. 1.5 pt

A system is viewed as a collection of subsystems determined during the system and object design. The Integration testing strategy determines the order in which the subsystems are selected for testing and integration.

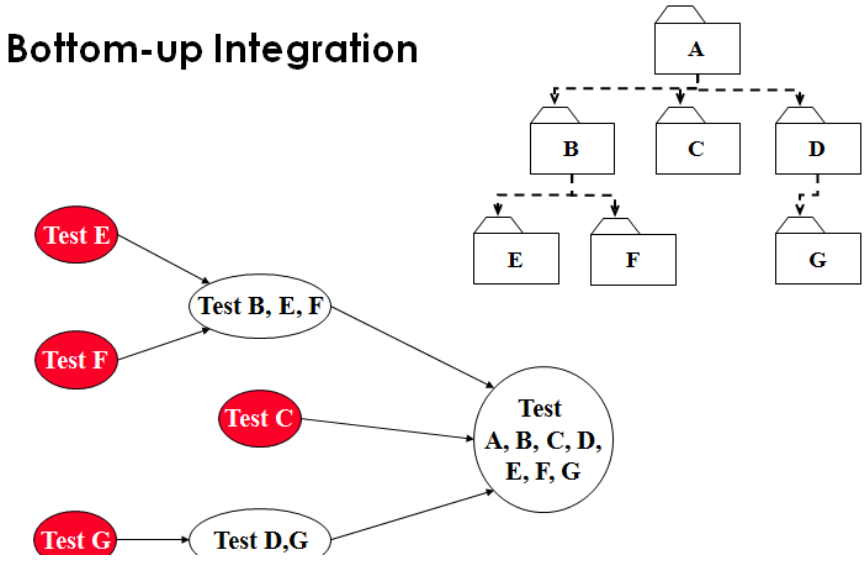
The main goal is to test all interfaces between subsystems and the interaction of subsystems.

Reasons for doing integration testing:

- Unit tests only test the unit in isolation
- Many failures result from faults in the interaction of subsystems
- Often many Off-the-shelf components are used that cannot be unit tested
- Without integration testing the system test will be very time consuming
- Failures that are not discovered in integration testing will be discovered after the system is deployed and can be very expensive.
-



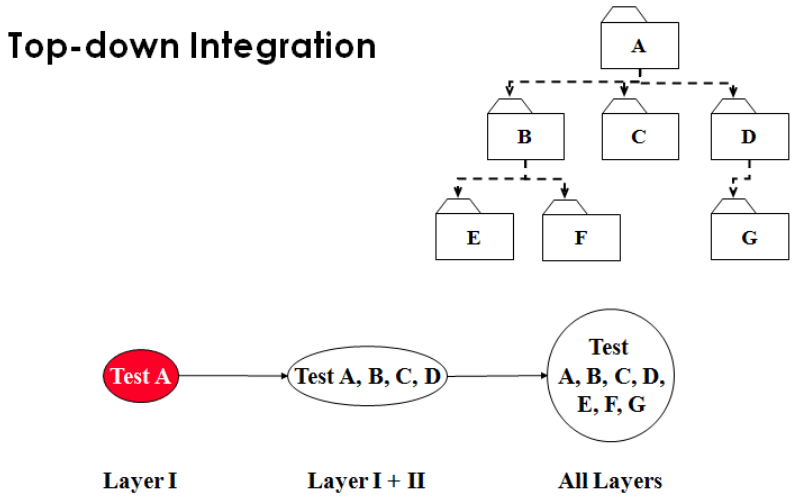
This unit tests each of the subsystems, and then does one gigantic integration test, in which all the subsystems are immediately tested together. Don't try this!! Why: The interfaces of each of the subsystems have not been tested yet.



- The subsystems in the lowest layer of the call hierarchy are tested individually
- Then the next subsystems are tested that call the previously tested subsystems
- This is repeated until all subsystems are included
- Drivers are needed.

- Con:
  - Tests the most important subsystem (user interface) last
  - Drivers needed
- Pro
  - No stubs needed
  - Useful for integration testing of the following systems
    - Object-oriented systems
    - Real-time systems

- Systems with strict performance requirements.



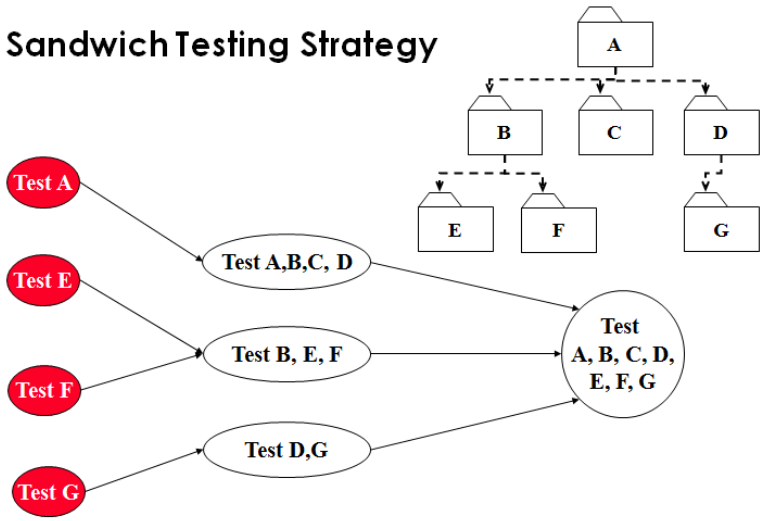
- Test the top layer or the controlling subsystem first
- Then combine all the subsystems that are called by the tested subsystems and test the resulting collection of subsystems
- Do this until all subsystems are incorporated into the test
- Stubs are needed to do the testing.

Pro

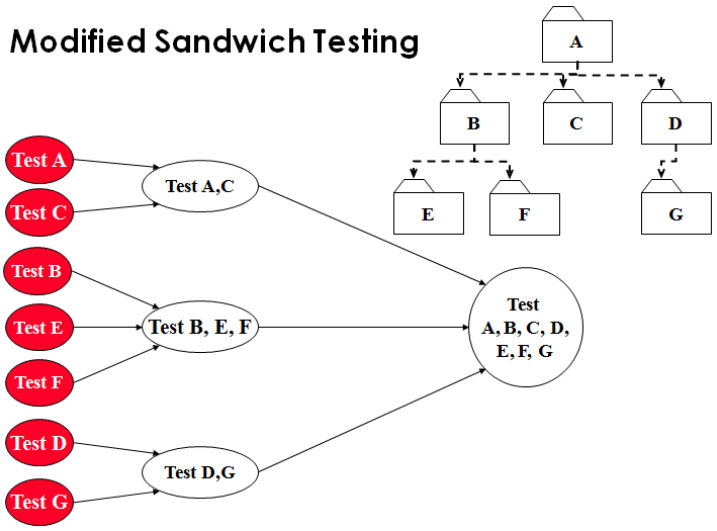
- Test cases can be defined in terms of the functionality of the system (functional requirements)
- No drivers needed

Cons

- Writing stubs is difficult: Stubs must allow all possible conditions to be tested.
- Large number of stubs may be required, especially if the lowest level of the system contains many methods.
- Some interfaces are not tested separately.



- Combines top-down strategy with bottom-up strategy
- The system is viewed as having three layers
  - A target layer in the middle
  - A layer above the target
  - A layer below the target
- Testing converges at the target layer.
- Top and Bottom Layer Tests can be done in parallel
- Problem: Does not test the individual subsystems and their interfaces thoroughly before integration
- Solution: Modified sandwich testing strategy



- Test in parallel:
  - Middle layer with drivers and stubs
  - Top layer with stubs
  - Bottom layer with drivers
- Test in parallel:
  - Top layer accessing middle layer (top layer replaces drivers)
  - Bottom accessed by middle layer (bottom layer replaces stubs).

Continuous testing

- Continuous build:
  - Build from day one
  - Test from day one
  - Integrate from day one
  - System is always runnable
- Requires integrated tool support:
  - Continuous build server
  - Automated tests with high coverage
  - Tool supported refactoring
  - Software configuration management
  - Issue tracking.

Total 9 pt
1 pt by default