INF 117
Project in Software Engineering

Lecture Notes ~Spring Quarter, 2008

Michele Rousseau

Set 7 – Going from Design to Code, Integration Test Plans

What's Next

APRIL 2008

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
20 Week 4	21	22 Earth Day	23 No Lecture	24 Proj. Iter. #3 Proj. Iter. #3 Project Plan #2	25 Proj. Iteration Proj. Approved	26
27 Week 5	28	29	30 Stud. Pres-Des Order 2,3,4,1 Team App: #2 Peer Eval #2	1 Des. Iter: #2 Test-Plan It #2 (Incl Des)	2 Team Log #2 Course Log #1	

MAY 2008

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
4 Week 6	5	6	7	8 Des Iter: #3 Code Iter #1 Proj. Plan #3	9 Cust. Milestone: Des. Approved	10

Announcements

- Client approval should be on or before the due date
  - Don't wait until the last minute in case they want changes
- Project Plan should reflect the details necessary to finish design and prepare for coding

Today's Class

- UML
  - Activity Diagrams
  - Communication Diagrams
- Going from Use Cases to Code
- Integration Testing

Activity Diagrams

- Describe
  - Procedural logic
  - Business process
  - Workflow
- A flow chart with support for parallel behavior
- Branches and Merges model the conditional behavior
- Branch: has a single incoming transition multiple, conditional, outgoing transitions
- Merge: where conditional behavior terminates

Each branch has a corresponding merge

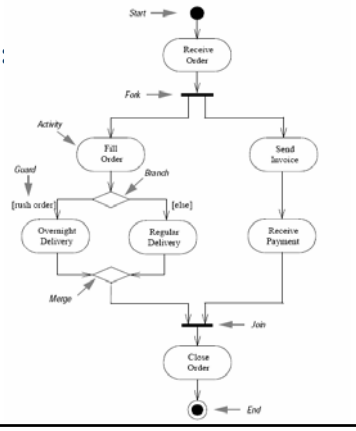
- Represented as a Diamond 

Activity Diagram (2)

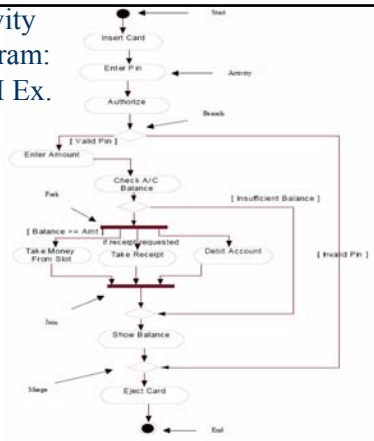
- Forks and Joins model parallel behavior
- Fork: has a single incoming transition and multiple outgoing transitions (exhibiting parallel behavior)
- Join: synchronizes the parallel behavior
  - All parallel behaviors complete at the join
- Represented as a thick line

Each Fork has generally has a corresponding Join

Activity Diagram: Order Ex.



Activity Diagram: ATM Ex.

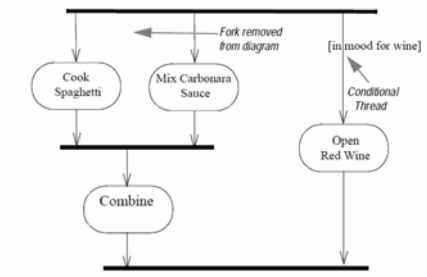


Conditional Thread

There are some exceptions to the each fork having a corresponding join:

- Conditional Thread: A condition on the thread originating from the fork to create an exception for the join rule.
  - If the condition is false then that condition is considered to be complete

Conditional Thread: Example

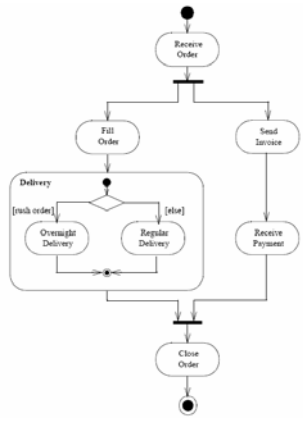


Superstates

- What if you need to decompose your activity diagram?
- Superstates
  - You can show the superstate with the internal behavior inside or
  - You can show these in a parent diagram
  - You can also use explicit initial and final states

Adv: you can decouple the parent from the subsidiary and use it in other contexts

Activity Diagram: Superstate



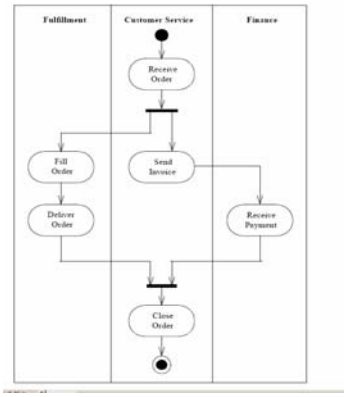
Partitioning an Activity Diagram

Activity diagrams tell you what is happening, but how do you know who does what?

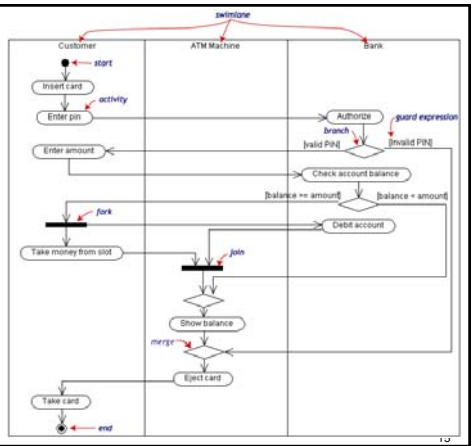
(in programming – which class is responsible for each activity)

- Swimlanes: group related activities into one column (usually organizationally)
  - You must arrange your diagram into vertical zones separated by lines.
  - Can be difficult with complex diagrams
    - In this case use non-linear zones – better than nothing

Swimlanes



Atm Ex.



When do you use Activity Diagrams?

- *Modeling parallel behavior*
- *Analyzing a use case*
  - Trying to understand what actions need to take place
  - Determine behavioral dependencies
- *Understanding workflow*
  - Documenting the logic of a business process
- *Describing a complicated sequential algorithm*
- *Dealing with multi-threaded applications*

Not so good for

- *Trying to see how objects collaborate*
  - Use an interaction diagram for that
- *Trying to see how an object behaves over its lifetime*
  - Use a state diagram for that

Communication Diagrams

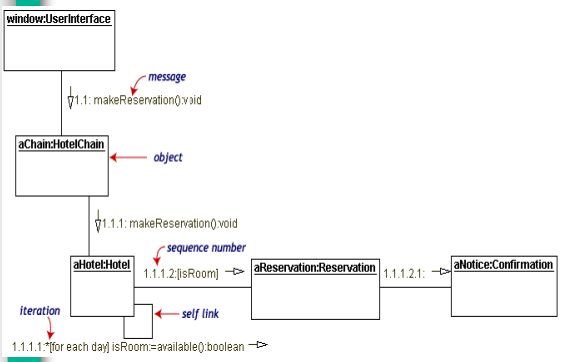
- Used to be known as Collaboration Diagrams (UML 1.x) – but modified for 2.0
- Show interactions between run-time elements
- Similar to sequence diagrams, but
  - Focus on objects roles & structure
  - Sequence diagram is better at visualizing processing over time

It is an object diagram that shows message passing relationships

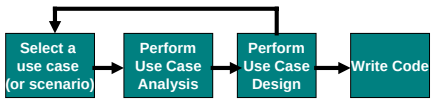
Emphasis on the flow of messages among objects, rather than timing and ordering of messages

- Sequence Numbers are on arrows rather than vertical order

Communication Diagrams: Ex



From Use Cases to Code



Use Case Analysis

For each use case in an iteration…

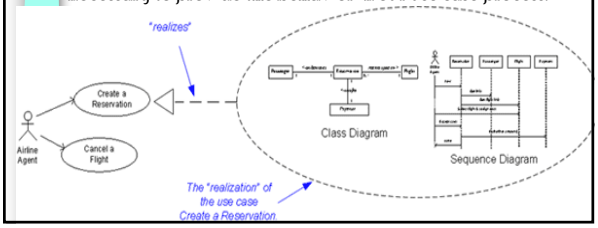
1. Create a use case realization
2. Supplement the Use-Case descriptions
   - k if necessary
3. Find Analysis Classes from Use-Case Behavior
4. Distribute Behavior to Analysis Classes

1. Use-Case Realization

A ***use-case realization*** is a collection of UML diagrams which together validate that we have

- the classes → Class Diagrams (static relationships)
- responsibilities
- object interactions → Interaction Diagrams (dynamic relationships) – could be Sequence or Collaboration

necessary to provide the behavior in our use case process.



2. Supplement the Use-Case descriptions (if necessary)

- k Beef up your use-case descriptions
  - Can include internal or non-visible behavior of the system
  - Do you need to do this for all of them?

    No! → Include just enough detail to understand the classes you will need

3. Find Analysis Classes from Use-Case Behavior &

- k identify a candidate set of analysis classes
- k Analysis Class
  - 3 Categories
    - Entity → Business level
      - Banking system → Customer, account, transaction (e-commerce or old school)
    - Controller → process & sequence aware
      - Control & direct the flow of control on an execution sequence
    - Boundary → I/O required by the s/w system

Describe the Class's Responsibilities

- Use nouns to determine classes

Class Name	Description	Responsibilities
Customer	Represents the human individual (no company accounts) who may request to reserve a vehicle	Manages the information associated with a specific customer (e.g. email address, physical address, phone #, etc.)
Customer Profile	Represents a set of properties describing the rental preferences for the associated Customer	Manages its attributes and values as a cohesive set of properties associated with a given Customer. Knows the Customer for which it manages these properties.
Vehicle	Represents a physical vehicle that has been requested by a customer	Knows its status (rented, damaged, dirty, etc..). Knows the vehicle inventory it is a part of, or the reservation it is assigned to. Knows its schedule for availability

Car Rental Example

4. Distribute Behavior to Analysis Class

- Sequence Diagrams
- Activity / State Diagrams

Next

For each resulting analysis class
Describe the Class's Responsibilities

- Describe the Class's Attributes and Associations
  - Define Class Attributes
  - Establish Associations between Analysis Classes
  - Describe Event Dependencies between Analysis Classes
- Establish Traceability
- Evaluate the Results of Use-Case Analysis

Other Notes

- Simplify your diagrams using subsystems
  - Packages can be used anywhere
- Use some underlying concepts
  - Abstraction
  - Encapsulation → Information hiding
    - Hide design decisions most likely to change
  - Polymorphism
    - Use Operations/functions in different ways

UML Reminders

- Don't drive your design by the diagram.
  - Drive it by the functionality of the system
- Always describe the entities of your diagrams with text
- Explicitly define your interfaces
- One more time..
- EXPLICITY DEFINE YOUR INTERFACES
  - What does this mean?

The more detailed/accurate the design → The easier it is to code

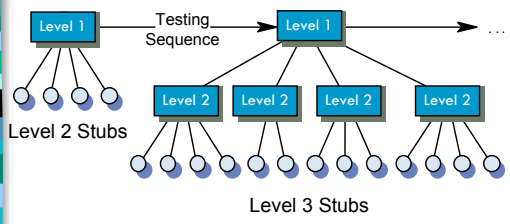
Integration Testing

- Purpose: to exercise the interfaces between classes/modules
  - Driven by design
  - What should it take in?
  - What should it supply?
  - What happens if they send the wrong stuff?
- Basic approaches
  - Top-Down
  - Bottom-up

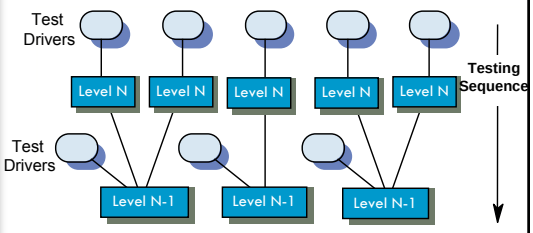
Integration Testing Approaches

- Top Down & Bottom Up
  - Top-down integration testing →
    - better at discovering errors in the system architecture
    - allows a limited demonstration at an early stage in the development
  - Bottom up →
    - Often easier to implement
- Problems with both approaches. Extra code may be required to observe tests

Top-Down Integration Testing



Bottom-Up Testing



Which Approach to use?

- Top-Down or Bottom Up?
- In practice, most integration involves a combination of these strategies