Object Modeling Approach

- Start with a problem statement
  - High-level requirements
- Define object model
  - Identify objects and classes
  - Prepare data dictionary
  - Identify associations and aggregations
  - Identify attributes of objects and links
  - Organize and simplify using inheritance
  - Iterate and refine the model
  - Group classes into modules
The Home Heating System

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

- The software shall control the heat in each room.
- The room shall be heated when the temperature is 2F below desired temp.
- The room shall no longer be heated when the temperature is 2F above desired temp.
- The flow of heat to each room shall be individually controlled by opening and closing its water valve.
- The valve shall be open when the room needs heat and closed otherwise.
- The user shall set the desired temperature on the thermostat.
- The operator shall be able to turn the heating system on and off.
- The furnace must not run when the system is off.
Home Heating Requirements

The purpose of the software for the Home Heating System is to control the heating system that heats the rooms of a house. The software shall maintain the temperature of each room within a specified range by controlling the heat flow to individual rooms.

- When the furnace is not running and a room needs heat, the software shall turn the furnace on
- To turn the furnace on the software shall follow these steps
  - open the fuel valve
  - turn the burner on
- The software shall turn the furnace off when heat is no longer needed in any room
- To turn the furnace off the software shall follow these steps
  - close fuel valve
  - turn burner off
Identify Object Classes

Candidate Classes

Controller

Home Heating System

range

Water Valve

creative

Temp Sensor

Control Panel

fuel

desired temp

on-off switch

fuel

controller

fuel valve

burner

furnace

water pump

hot water

Desired Object Classes

controller

fuel valve

burner

water pump

hot water

fuel

Temp Sensor

on-off switch

Eliminate Bad Classes

- Redundant classes
  - Classes that represent the same thing with different words
- Irrelevant classes
  - Classes we simply do not care about
- Vague classes
  - Classes with ill-defined boundaries
- Attributes
  - Things that describe individual objects
- Operations
  - Sequences of actions are often mistaken for classes
- Roles
  - The name of a class should reflect what it is, not the role it plays
- Implementation details
  - Save that for implementation
Eliminate Classes

Redundant
- heating system
- user

Irrelevant
- Fuel
- software
- Hot Water

Vague
- heat
- house
- heat flow
- home
- range

Attributes
- desired temp
- temperature

Operations
- None

Roles
- None

Implementation
- None

Classes After Elimination

Fuel Valve
Water Pump
Burner
Room
Home Heating System
Furnace
Temp Sensor
Water Valve
Operator
Controller
on-off switch

Prepare Data Dictionary

- Water Tank
  - The storage tank containing the water that circulates in the system.

- Pump-1
  - The pump pumping water from the Water Tank to the radiators in the rooms

Possible Associations

- Not much information from the prose requirements
- A lot of information from the system design

- A room consists of a thermometer and a radiator
- A radiator consists of a valve and a radiator element
- The home heating system consists of a furnace, rooms, a water pump, a control panel, and a controller
- The furnace consists of a fuel pump and a burner
- The control panel consists of an on-off switch and a thermostat
- The controller controls the fuel pump
- The controller controls the burner
- The controller controls the water pump
- The controller monitors the temperature in each room
- The controller opens and closes the valves in the rooms
- The operator sets the desired temperature
- The operator turns the system on and off
- The controller gets notified of the new desired temperature
Object Model

Home Heating System

Control Panel
On-Off Switch
Thermostat
Operator
Water Valve
Temp Sensor
Furnace
Burner
Fuel Valve
Water Pump
Runs

Room

Operator

Actuates

1..* Monitor

Controller

Pushes
Adjusts
Notifies
Ignites
Opens/Closes

Actuates

1..*

Heats
Object Model

Object Model - Modified
Attributes

Thermostat
desired-temp

On-Off switch
setting

Temp Sensor
temperature

Final OO Model

Home Heating
System

Control Panel

On-Off Switch
setting

Operator

Thermostat
desired-temp

Pushes
Adjusts

Room

Furnace

Water Pump

Burner

Fuel Valve

Temp Sensor
temperature

Heats

1..*

Room

1..*

1..*

Controller

Ignites

Opens/Closes

Runs
Iterate the Model

- Keep on doing this until you, your customer, and your engineers are happy with the model.

Operation vs Method

- **Operation**: specifies object behavior
- **Service**: represented by *set* of operns.
- **Message**: object requests execution of an opern. from another object by sending it mesg.
- **Method**: mesg is matched up with method defined by the class to which the receiving object belongs (or any of its superclasses)
- **Operations** of class are public *services* offered by class.
- **Methods** of its classes are the implementations of these *operations*. 
OO Using UML: Dynamic Models

Defining how the objects behave

Overview

- The object model describes the structure of the system (objects, attributes, and operations)
- The dynamic model describes how the objects change state (how the attributes change) and in which order the state changes can take place
- Several models used to find the appropriate dynamic behavior
  - Interaction diagrams
  - Activity diagrams
  - State Diagrams
- Uses finite state machines and expresses the changes in terms of events and states
Interaction Diagrams

We Will Cover

- Why interaction diagrams?
- Sequence diagrams
  - Capturing use-cases
  - Dealing with concurrency
- Collaboration diagrams
- When to use what
- When to use interaction diagrams
Different Types of Interaction Diagrams

- An Interaction Diagram typically captures a use-case
  - A sequence of user interactions

- **Sequence diagrams**
  - Highlight the sequencing of the interactions between objects

- **Collaboration diagrams**
  - Highlight the structure of the components (objects) involved in the interaction

---

Home Heating Use-Case

**Use case:** Power Up  
**Actors:** Home Owner (initiator)  
**Type:** Primary and essential  
**Description:** The Home Owner turns the power on. Each room is temperature checked. If a room is below the desired temperature the valve for the room is opened, the water pump started, the fuel valve opened, and the burner ignited. If the temperature in all rooms is above the desired temperature, no actions are taken.  
**Cross Ref.:** Requirements XX, YY, and ZZ  
**Use-Cases:** None
Sequence Diagram

Use the left column to offer comments about the messages

Synchronous message

Response to synchronous message

Guard for message

Example from Fowler

An Entry Window
An Order: Order
An Order: Order
A Stock Item: StockItem

A Reorder Item: ReorderItem

A Delivery Item: DeliveryItem
Concurrency

Another Example
Comment the Diagram

When the owner
turns the system on,
the on switch
notifies
the controller.
The controller
creates a room object
for each room in
the building.
The room
samples
the temperature
in each room every 5 s.
When a low temp
is detected
the room
notifies
the controller.

Example from Fowler

An Entry
Window
An Order
Window
An Order
Order
A Stock
Item
StockItem
A Delivery
Item
DeliveryItem

prepare

[for all order lines]
[prepare]

if [stock] = [check]

if [stock] = [check]

if [needsReorder] = [needsToReorder]

if [needsReorder] = [needsToReorder]

if [needsReorder] = [needsToReorder]
Collaboration Diagrams

Conditional Behavior

- Something you will encounter trying to capture complex use-cases
  - The user does something. If this something is X do this... If this something is Y do something else... If this something is Z...

- Split the diagram into several
  - Split the use-case also

- Use the conditional message
  - Could become messy

- **Remember, clarity is the goal!**
Comparison

- Both diagrams capture the same information
  - People just have different preferences
- We prefer sequence diagrams
  - They clearly highlight the order of things
  - Invaluable when reasoning about multi-tasking
- Others like collaboration diagrams
  - Shows the static structure
    - Very useful when organizing classes into packages
- We get the structure from the Class Diagrams

When to Use Interaction Diagrams

- When you want to clarify and explore single use-cases involving several objects
  - Quickly becomes unruly if you do not watch it
- If you are interested in one object over many use-cases -- state transition diagrams
- If you are interested in many objects over many use cases -- activity diagrams
State Diagrams

We Will Cover

- State Machines
  - An alternate way of capturing scenarios
    - Large classes of scenarios
- Syntax and Semantics
- When to use state machines
Events, Conditions, and States

- **Event**: something that happens at a point in time
  - Operator presses self-test button
  - The alarm goes off

- **Condition**: something that has a duration
  - The fuel level is high
  - The alarm is on

- **State**: an abstraction of the attributes and links of an object (or entire system)
  - The controller is in the state self-test after the self-test button has been pressed and the reset-button has not yet been pressed
  - The tank is in the state too-low when the fuel level has been below level-low for alarm-threshold seconds

Making a Phone Call Scenario

Context for example: (shorter version)

To make a call, the caller lifts receiver. The caller gets a dial tone and the caller dials digit (x). The dial tone ends. The caller completes dialing the number. The callee phone begins ringing at the same time a ringing begins in caller phone. When the callee answers the called phone stops ringing and ringing ends in caller phone. The phones are now connected. The caller hangs up and the phones are disconnected. The callee hangs up.
Partial Class Diagram

Connected by

1

Line

Caller

Connected by

1

Phone

Callee

Event Trace

Phone:Caller

Line:theLine

Phone:Callee

caller lifts receiver
dial tone begins
dials digit (3)
dial tone ends

dials digit (2)
dials digit (3)
dials digit (4)
dials digit (5)
ringing tone

phone rings
tone stops
phones connected
phones disconnected
callee answers
ringing stops
phones connected
phones disconnected
caller hangs up

callee hangs up
State Diagram for Scenario

Scenario 2

Phone:Caller
- caller lifts receiver
- dial tone begins
- dial digit (3)
- dial tone ends
- dial digit (2)
- dial digit (3)
- dial digit (4)
- dial digit (5)
  - busy tone
- caller hangs up

Phone:Caller

Line:theLine

Phone:Caller

Disconnected
Modified State Machine

- Sometimes the state transitions are conditional

**Conditions**
Operations (AKA Actions)

- Actions are performed when a transition is taken or performed while in a state.
- Actions are terminated when leaving the state.

Hierarchical State Machines

- Group states with similar characteristics.
- Enables information hiding.
- Simplifies the diagrams.
Information Hiding

Event Generalization

- Related events can inherit properties from each other
- If an event at a lower level occurs, the event at a higher level also occurred
- Event attributes
  - mouse-up.location
  - mouse-down.device
  - mouse-button.time

---

Software Engineering (Cheng)
Concurrency

- Some states represent several concurrent concepts
- Concurrency is supported by the state machines
- Concurrent state machines are separated by dashed lines

State Machines - Summary

- Events
  - instances in time
- Conditions
  - conditions over time
- States
  - abstraction of the attributes and associations
- Transitions
  - Takes the state machine from one state to the next
    - Triggered by events
    - Guarded by conditions
    - Cause actions to happen
- Internal actions
  - something performed in a state
- Hierarchies
  - allows abstraction and information hiding
- Parallelism
  - models concurrent concepts
When to use State Machines

- When you want to describe the behavior of one object for all (or at least many) scenarios that affect that object
- Not good at showing the interaction between objects
  - Use interaction diagrams or activity diagrams
- Do not use them for all classes
  - Some methods prescribe this
  - Very time consuming and questionable benefit

HERE
Coming up with the State Diagrams

Modeling Approach

- Prepare scenarios
  - Work with the customer
  - Start with normal scenarios
  - Add abnormal scenarios

- Identify events (often messages)
  - Group into event classes

- Draw some sequence diagrams
  - Find objects with complex functionality you want to understand better

- Build a state diagram for the complex classes
## Scenario-2

<table>
<thead>
<tr>
<th>Control Panel</th>
<th>Room</th>
<th>Controller</th>
<th>Fuel Valve</th>
<th>Burner</th>
<th>Water Pump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every 5s</td>
<td></td>
<td>desired-temp-change</td>
<td>request-temp</td>
<td>respond-temp</td>
<td></td>
</tr>
<tr>
<td>Desired temp change</td>
<td>Every 5s</td>
<td></td>
<td>open-valve</td>
<td>start-burner</td>
<td>pump-on</td>
</tr>
<tr>
<td>Temp Low</td>
<td>Every 5s</td>
<td></td>
<td>open-water-valve</td>
<td>request-temp</td>
<td>respond-temp</td>
</tr>
<tr>
<td>Temp Normal</td>
<td>Every 5s</td>
<td></td>
<td>close-water-valve</td>
<td>request-temp</td>
<td>respond-temp</td>
</tr>
</tbody>
</table>

### Scenario-2, v2

<table>
<thead>
<tr>
<th>Control Panel</th>
<th>Room</th>
<th>Controller</th>
<th>Fuel Valve</th>
<th>Burner</th>
<th>Water Pump</th>
</tr>
</thead>
<tbody>
<tr>
<td>Every 5s</td>
<td></td>
<td>desired-temp-change</td>
<td>request-temp</td>
<td>respond-temp</td>
<td></td>
</tr>
<tr>
<td>Desired temp change</td>
<td>Every 5s</td>
<td></td>
<td>open-water-valve</td>
<td>request-temp</td>
<td>respond-temp</td>
</tr>
<tr>
<td>Temp Low</td>
<td>Every 5s</td>
<td></td>
<td>open-valve</td>
<td>start-burner</td>
<td>pump-on</td>
</tr>
<tr>
<td>Temp Normal</td>
<td>Every 5s</td>
<td></td>
<td>request-temp</td>
<td>respond-temp</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Every 5s</td>
<td></td>
<td>close-water-valve</td>
<td>request-temp</td>
<td>respond-temp</td>
</tr>
</tbody>
</table>

Software Engineering (Cheng) 30
**Dynamic Model**

- **Water Pump**
  - On
  - Off

- **Fuel Valve**
  - Open
  - Closed

- **Burner**
  - On
  - Off

**More Dynamic Model**

- **Room**
  - Idle
    - Temp-Sensor
      - Idle
      - Processing Request

- **Valve**
  - open-water-valve/wv-open
  - close-water-valve/wv-close

- **Temp-Sensor**
  - request-temp
  - temp-report(x)/respond-temp(x)
**Even More Dynamic Model**

**Controller**

TemperatureControl
- respond-temp(x)\{x>desired-temp+2\} / stop-heating
- timeout(5s) / request-temp

HomeHeatingSystemControl
- timeout(1s) / start-burner
- timeout(1s) / pump-on, open-water-valve

**Even More Dynamic Model, v2**

**Controller**

TemperatureControl
- respond-temp(x)\{x>desired-temp+2\} / stop-heating
- timeout(5s) / request-temp

HomeHeatingSystemControl
- timeout(1s) / start-burner
- timeout(1s) / pump-on, open-water-valve
Identify Key Operations

- Operations from the object model
  - Accessing and setting attributes and associations (often not shown)
- Operations from events
  - All events represent some operation
- Operations from actions and activities
  - Actions and activities represent some processing activity within some object
- Operations from functions
  - Each function typically represent one or more operations
- Shopping list operations
  - Inherent operations (what should be there)

Complete OO Model
Iterate the Model

- Keep on doing this until you, your customer, and your engineers are happy with the model

Activity Diagrams
We Will Cover

- History of activity diagrams in UML
  - A highly personal perspective
- Activity diagrams
- Swimlanes
- When to use activity diagrams
  - When not to

Activity Diagrams

- Shows how activities are connected together
  - Shows the order of processing
  - Captures parallelism
- Mechanisms to express
  - Processing
  - Synchronization
  - Conditional selection of processing
Why Activity Diagrams

- Very good question
  - Not part of any previous (UML related) method
  - To make UML more inclusive of business modeling needs

- Suitable for modeling of business activities
  - UML and OO is becoming more prevalent in business applications
  - Object frameworks are making an inroad
  - Stay within one development approach and notation
  - Notation similar to process modeling languages

Coffee Example
HACS Use-Cases

Use case: Distribute Assignments  
Actors: Instructor (initiator), Student  
Type: Primary and essential  
Description: The Instructor completes an assignment and submits it to the system. The instructor will also submit the delivery date, due date, and the class the assignment is assigned for. The system will at the due date mail the assignment to the student.

Cross Ref.: Requirements XX, YY, and ZZ  
Use-Cases: Configure HACS must be done before any user (Instructor or Student) can use HACS

Activity Diagrams for Use Cases
Swimlanes (Who Does What?)

Problems with Activity Diagrams

- They are similar to flowcharts
  - Very easy to make a traditional data-flow oriented design
- Switching to the OO paradigm is hard enough as it is
  - Extensive use of activity charts can make this shift even harder
- However...
  - Very powerful when you know how to use them correctly
When to Use Activity Diagrams

- Useful when
  - Analyzing a use case (or collection of use cases)
  - Understanding workflow in an organization
  - Working with multi-threaded applications
    - For instance, process control applications
  - Do not use activity diagrams
    - To figure out how objects collaborate
    - See how objects behave over time

Approaching a Problem

**Where do we start?**

**How do we proceed?**
Where Do We Start?

- Start with the requirements
  - Capture your goals and possible constraints
  - Environmental assumptions
- Use-case analysis to better understand your requirements
  - Find actors and a first round of use-cases
- Start conceptual modeling
  - Conceptual class diagram
  - Interaction diagrams to clarify use-cases
  - Activity diagrams to understand major processing

How Do We Continue?

- Refine use-cases
  - Possibly some “real” use-cases
    - Using interface mockups
- Refine (or restructure) your class diagram
  - Based on your hardware architecture
    - For instance, client server
- Refine and expand your dynamic model
  - Until you are comfortable that you understand the required behavior
- Identify most operations and attributes
How Do We Wrap Up?

- Refine the class diagram based on platform and language properties
  - Navigability, public, private, etc
  - Class libraries
- Identify all operations
  - Not the trivial get, set, etc.
- Write a contract for each operation
- Define a collection of invariants for each class
- Implement

Why is requirements analysis difficult?

- Communication: misunderstandings between the customer and the analyst
  - Analyst doesn’t understand the domain
  - Customer doesn’t understand alternatives and trade-offs
- Problem complexity
  - Inconsistencies in problem statement
  - Omissions/incompleteness in problem statement
  - Inappropriate detail in problem statement
Why is requirements analysis difficult?

- Need to accommodate change
  - Hard to predict change
  - Hard to plan for change
  - Hard to foresee the impact of change

First Law of Software Engineering

“No matter where you are in the system lifecycle, the system will change, and the desire to change it will persist throughout the lifecycle.”
Reasons for changing requirements

- Poor communication
- Inaccurate requirements analysis
- Failure to consider alternatives
- New users
- New customer goals
- New customer environment
- New technology
- Competition
- Software is seen as malleable

Changes made after the requirements are approved increase cost and schedule

Requirements Products

- Specification document
  - Agreement between customer and developer
  - Validation criteria for software
- Preliminary users manual
- Prototype
  - If user interaction is important
  - If resources are available
- Review by customer and developer
  - Iteration is almost always required
Analysis: Steps to follow

- Obtain a problem statement
- Develop use cases (depict scenarios of use)
- Build an object model and data dictionary
- Develop a dynamic model
  - state and sequence diagrams
- Verify, iterate, and refine the models
- Produce analysis document (e.g., SRS)

Use Cases

- High-level overview of system use
- Identify scenarios of usage
- Identify actors of the system:
  - External entities (e.g., users, systems, etc.)
- Identify system activities
- Draw connections between actors and activities
- Identify dependencies between activities (i.e., extends, includes)
Analysis: Object Model

- Organization of system into classes connected by associations
  - Shows the static structure
  - Organizes and decomposes system into more manageable subsystems
  - Describes real world classes and relationships

Analysis: Object Model

- Object model precedes the dynamic model because
  - static structure is usually better defined
  - less dependent on details
  - more stable as the system evolves
Analysis: Object Model

- Information comes from
  - The problem statement and use cases
  - Expert knowledge of the application domain
    - Interviews with customer
    - Consultation with experts
    - Outside research performed by analyst
  - General knowledge of the real world

Object Model: Steps to follow

- Identify classes and associations
  - Nouns and verbs in a problem description
- Create data dictionary entry for each
- Add attributes
- Combine and organize classes using inheritance
**Analysis: Dynamic model**

- Shows the time dependent behavior of the system and the objects in it
- Expressed in terms of
  - states of objects and activities in states
  - events and actions
- State diagram summarizes permissible event sequences for objects with important dynamic behavior

**Dynamic Model: Steps to follow**

- Use cases provide scenarios of typical interaction sequences
- Identify events between objects (Sequence Diagram)
- Prepare an event trace for each scenario
- Build state diagrams
- Match events between objects to verify consistency
Analysis: Iteration

• Analysis model will require multiple passes to complete
• Look for inconsistencies and revise
• Look for omissions/vagueness and revise
• Validate the final model with the customer