Object Modeling

CSCE 740 - Lecture 16 - 10/26/2015

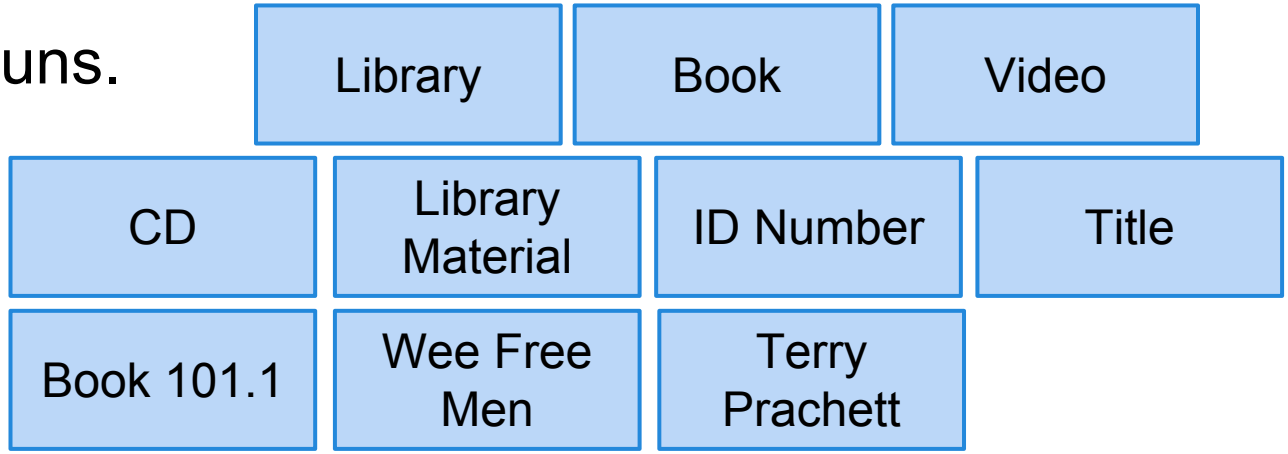
Objectives for Today

- Introduce methods for starting an object model.
  - Identifying classes, their attributes, and their operations.
  - Identifying associations between classes.
- Get some experience with OO design.

An Approach for Object Modeling

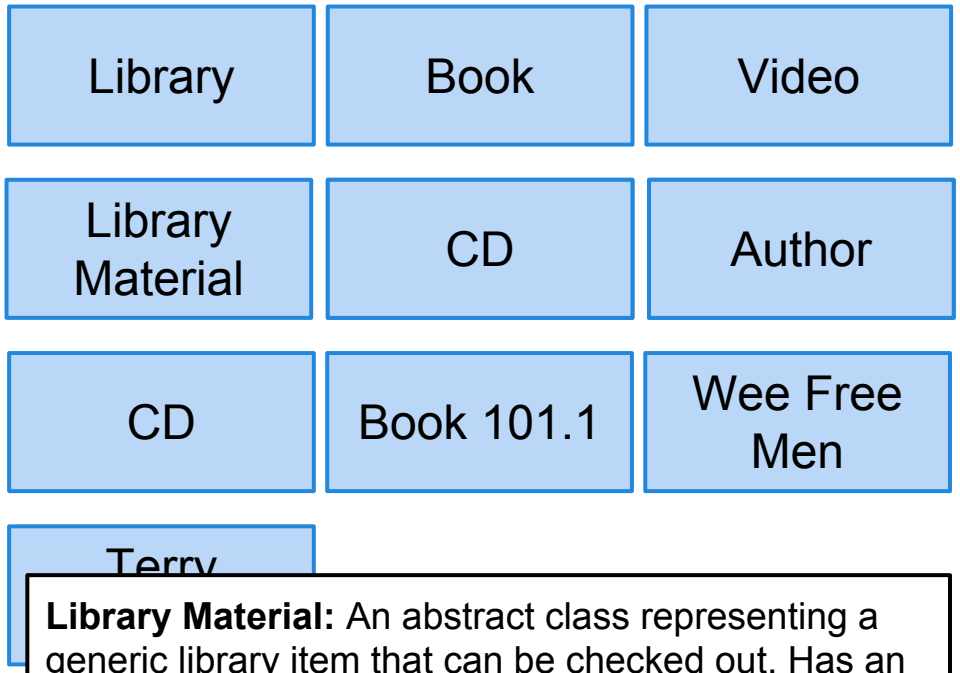
- Start with a problem statement.
  - High-level requirements
- Identify potential objects.
  - Look for nouns.

A library has books, videos, and CDs that it loans to its users. All library material has an ID number and a title. Book 101.1 is The Wee Free Men by Terry Prachett.



Object Modeling Approach

- Refine and remove bad classes
  - Redundant, vague, or irrelevant.
  - Abstract objects to classes.
- Prepare data dictionary
  - Describe each class and its purpose.

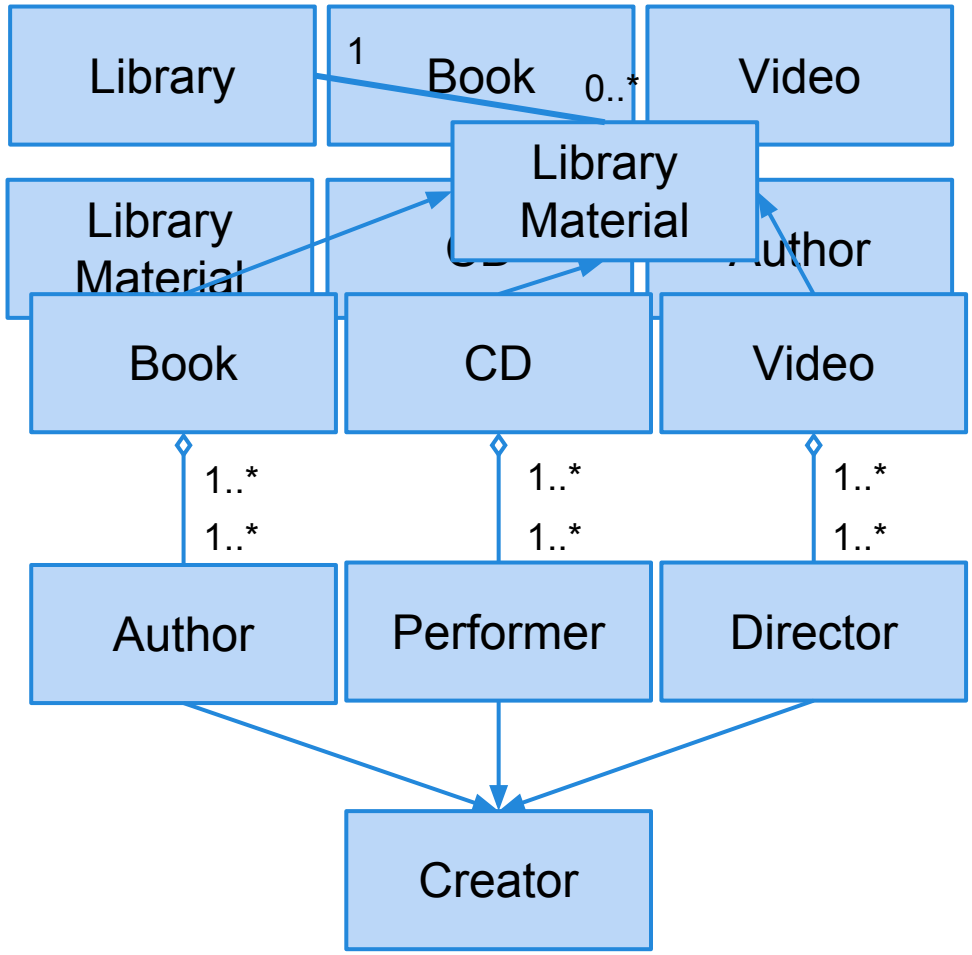


Library Material: An abstract class representing a generic library item that can be checked out. Has an ID and title.

Book: A class representing a book that can be checked out. Has an author in addition to inherited attributes ID and title.

Object Modeling Approach

- Identify associations and aggregations.
- Identify the attributes and operations of classes.
- Organize and simplify using inheritance.



Define Attributes and Operations

- What are the *responsibilities* of the class?
  - Use tools such as data dictionaries to define responsibilities of a class - what services must they perform or allow others to perform.
  - Classes were nouns, now look for **verbs**.
- General guidelines:
  - Responsibilities should be evenly distributed between classes.
  - Information related to a responsibility should be stored in the class responsible for that service.
    - Those are the attributes.

Identify Associations

Classes fulfill responsibilities in one of two ways:

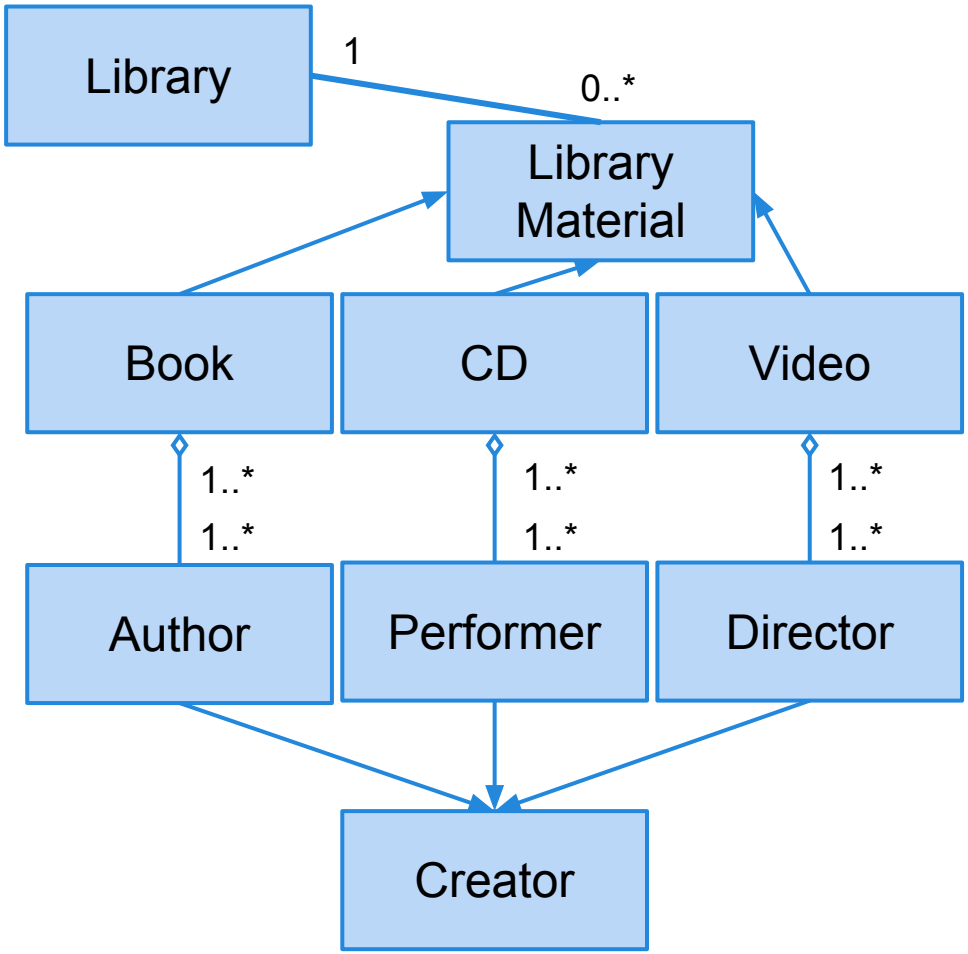
- It can use its own methods to modify its own attributes.
- It can collaborate with other classes.

If a class cannot fulfill its responsibilities alone, identify and document the associations.

- is-part-of (aggregation)
- has-knowledge-of (association)
- depends-upon (association)

Object Modeling Approach

- Iterate and refine the model.
  - You will almost always go through multiple iterations of a design.
- Group classes into subsystems.
  - Which classes can combine to form an independent grouping?



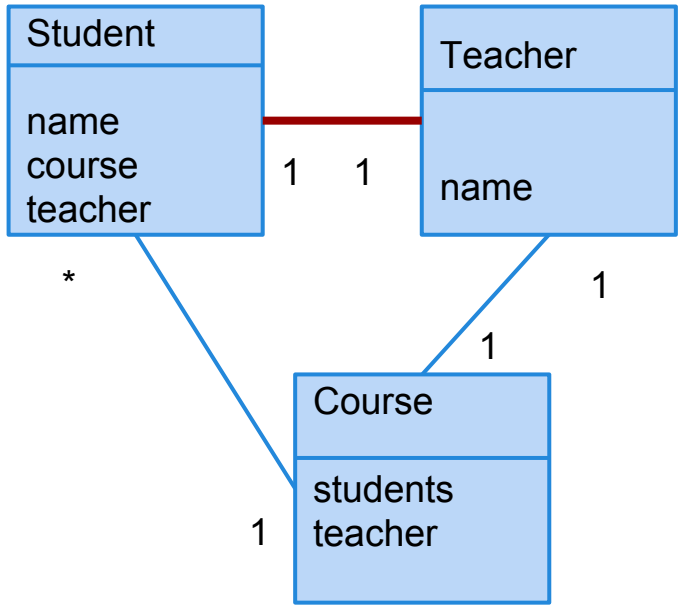
Refinement

The software design is often not optimal. Before implementation, consider how it can be improved.

Watch for:

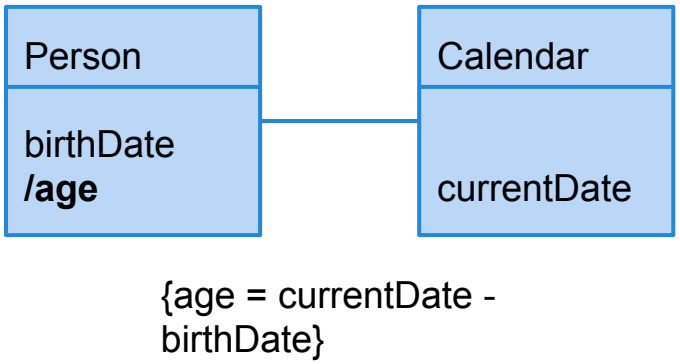
- Redundant associations.
- Attributes that can be derived at runtime.

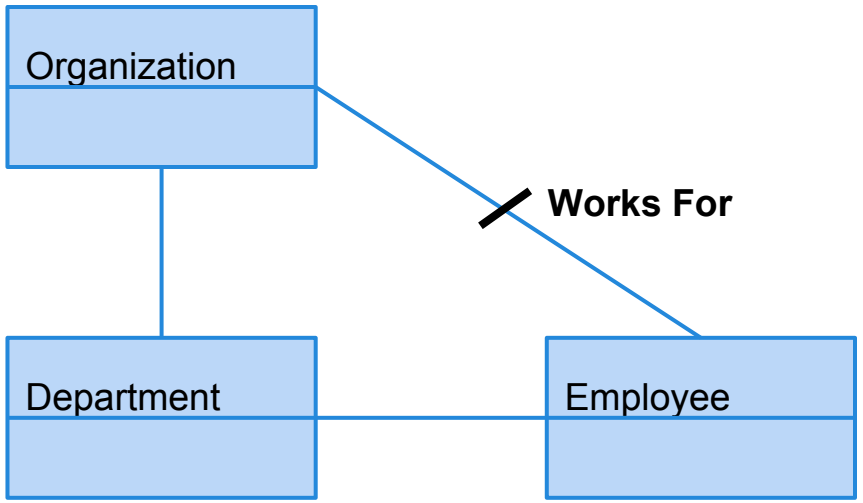
- Remove redundant associations



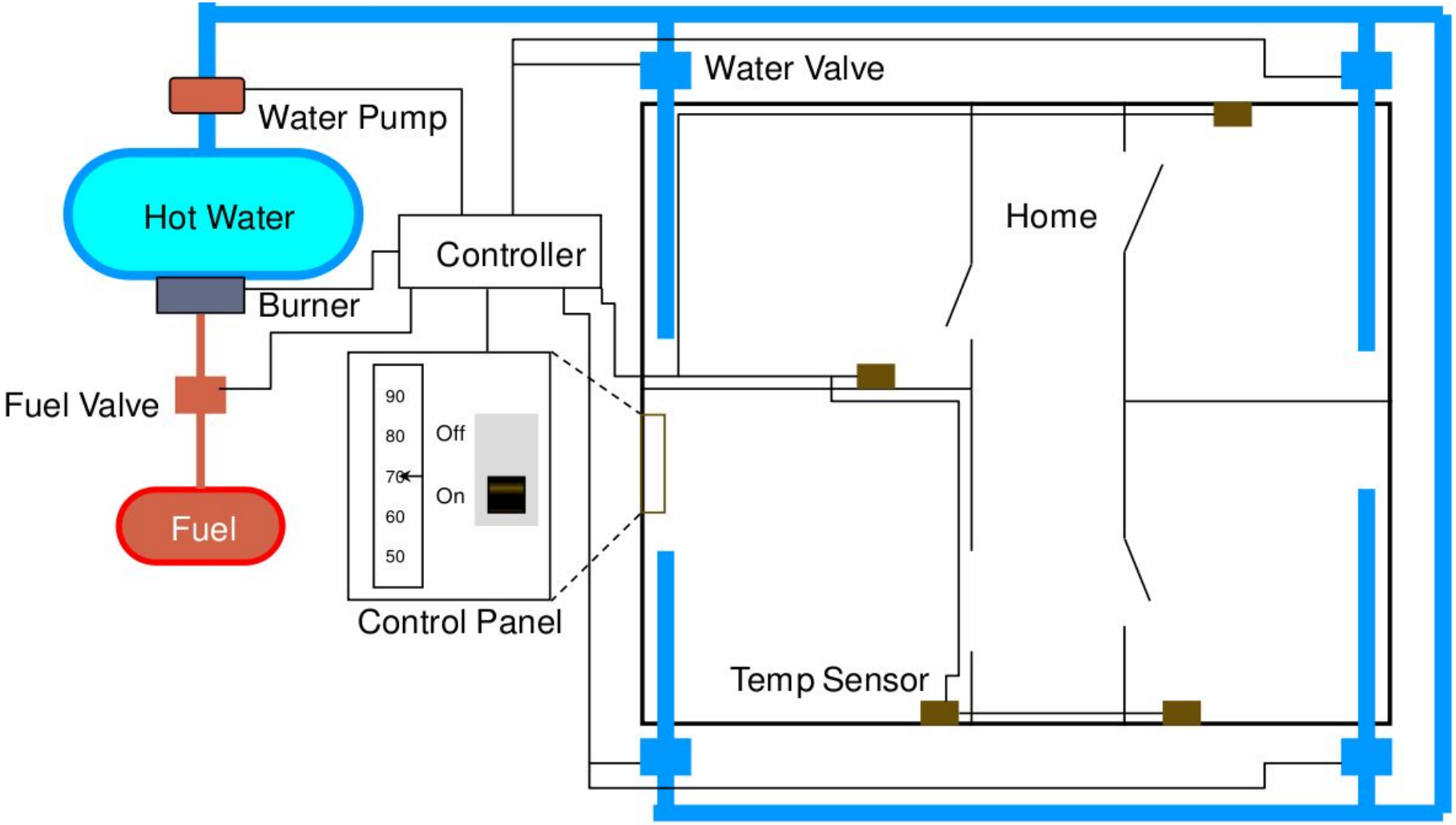
Derived Links and Attributes

Derived entities can be calculated from other entities. Indicated by a slash. They are potentially redundant.





The Home Heating System

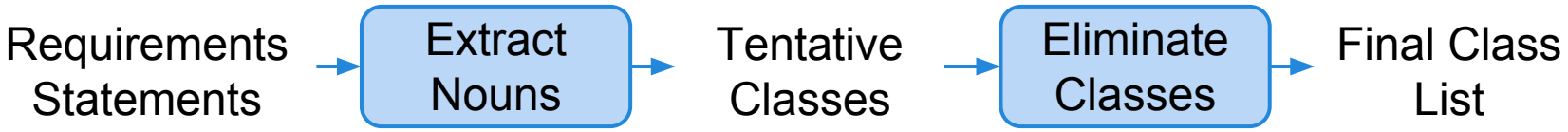


Home Heating Requirements

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.

- 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
- 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



Water Pump
Hot Water
Burner
Furnace
Fuel Valve
Fuel
Desired Temperature
On-Off Switch
Heating System
House
Room
Temperature
Home
Thermostat
Range
Control Panel
Heat Flow
Home Heating System
Water Valve
Controller
Software
User
Heat
Operator

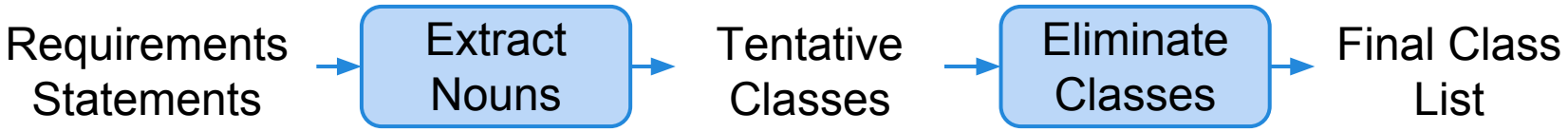
Eliminate Bad Classes

- 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 or make up other classes.

Eliminate Bad Classes (Continued)

- 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 those for the implementation.

Identify Object Classes

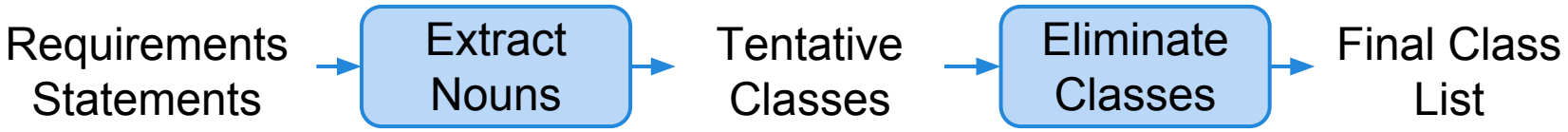


Water Pump
~~Hot~~ Water
Burner
Furnace
Fuel Valve
~~Fuel~~
~~Desired~~ Temperature
On-Off Switch
~~Heating~~ System

~~House~~
Room
~~Temperature~~
~~Home~~
Thermostat
~~Range~~
Control Panel
~~Heat~~ Flow
Home Heating System

Water Valve
Controller
~~Software~~
~~User~~
~~Heat~~
Operator

Classes After Elimination



Water Pump
Room
Burner
Furnace
Fuel Valve
Operator
Control Panel
On-Off Switch
Home Heating System
Water Valve
Controller
Thermostat

Prepare Data Dictionary

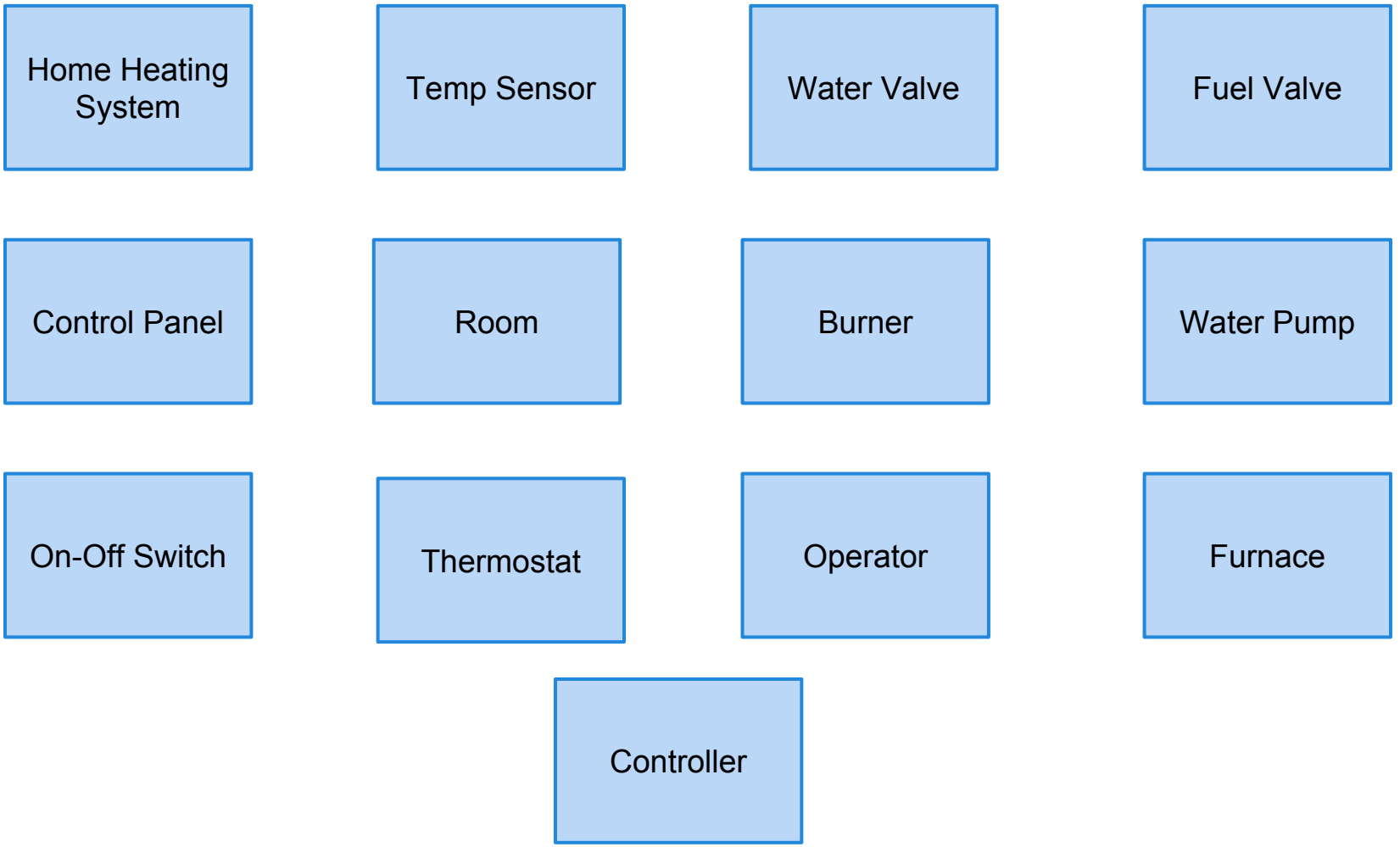
- Describe each class and its purpose.
- What are the classes' responsibilities? What information does it need to perform those services?

Derive Possible Associations

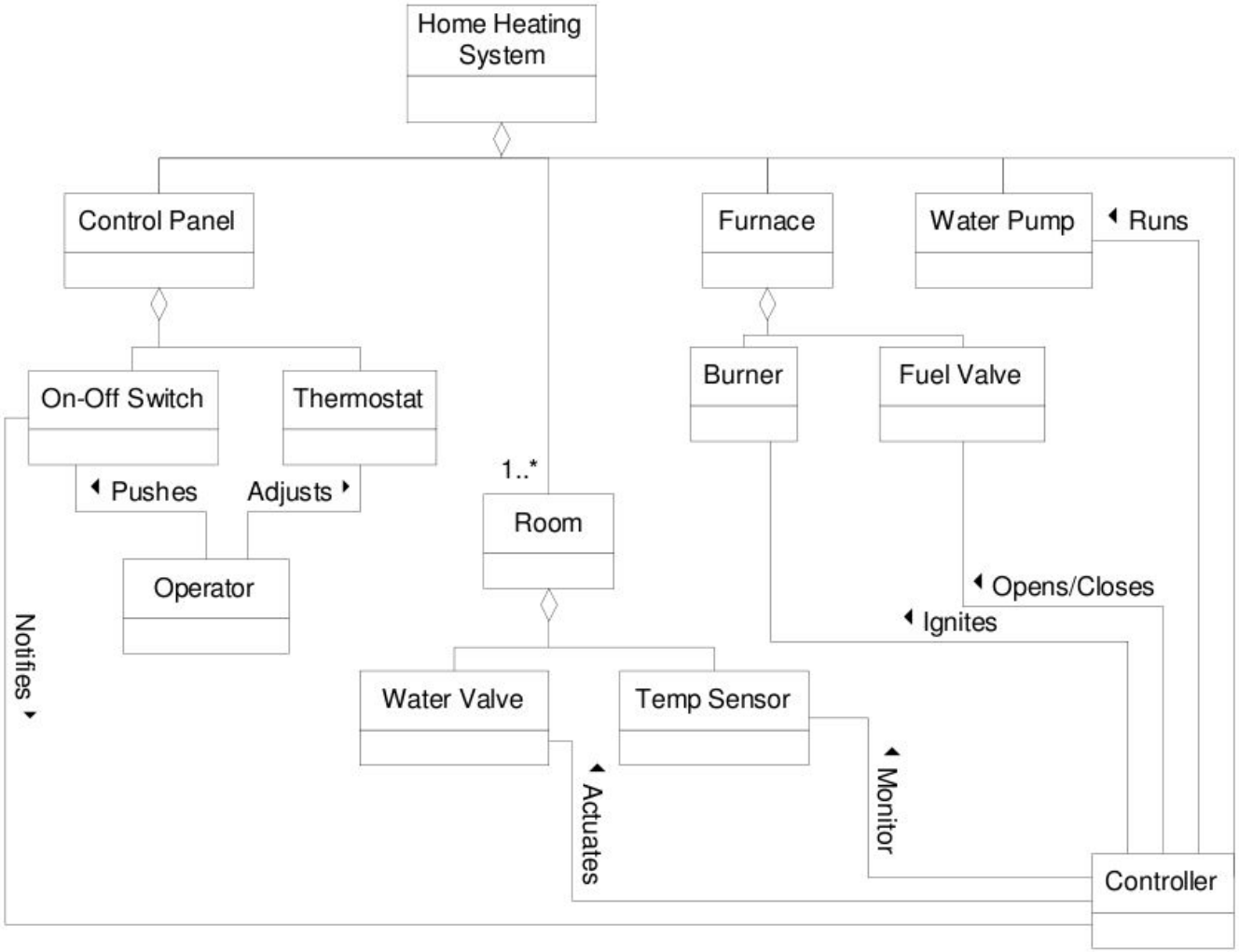
- Not much information from the prose requirements.
- … but, a lot of information from the data dictionary and physical design.

- 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 burner, and the water pump. It monitors the temperature in each room, and opens and closes the valves in the rooms
- The operator sets the desired temperature, and turns the system on and off
- The controller gets notified of the new desired temperature

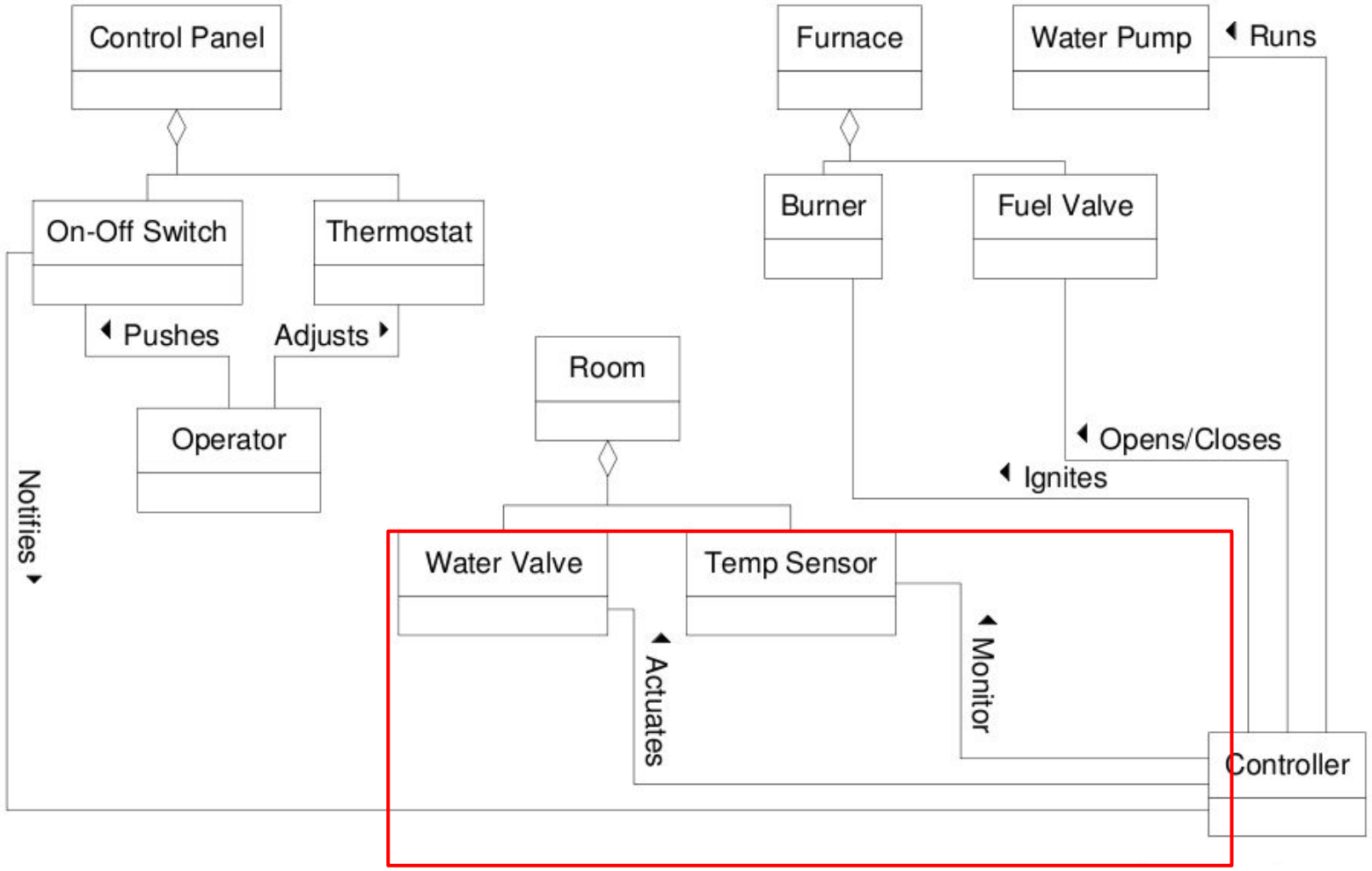
Add Associations to Complete the Class Diagram



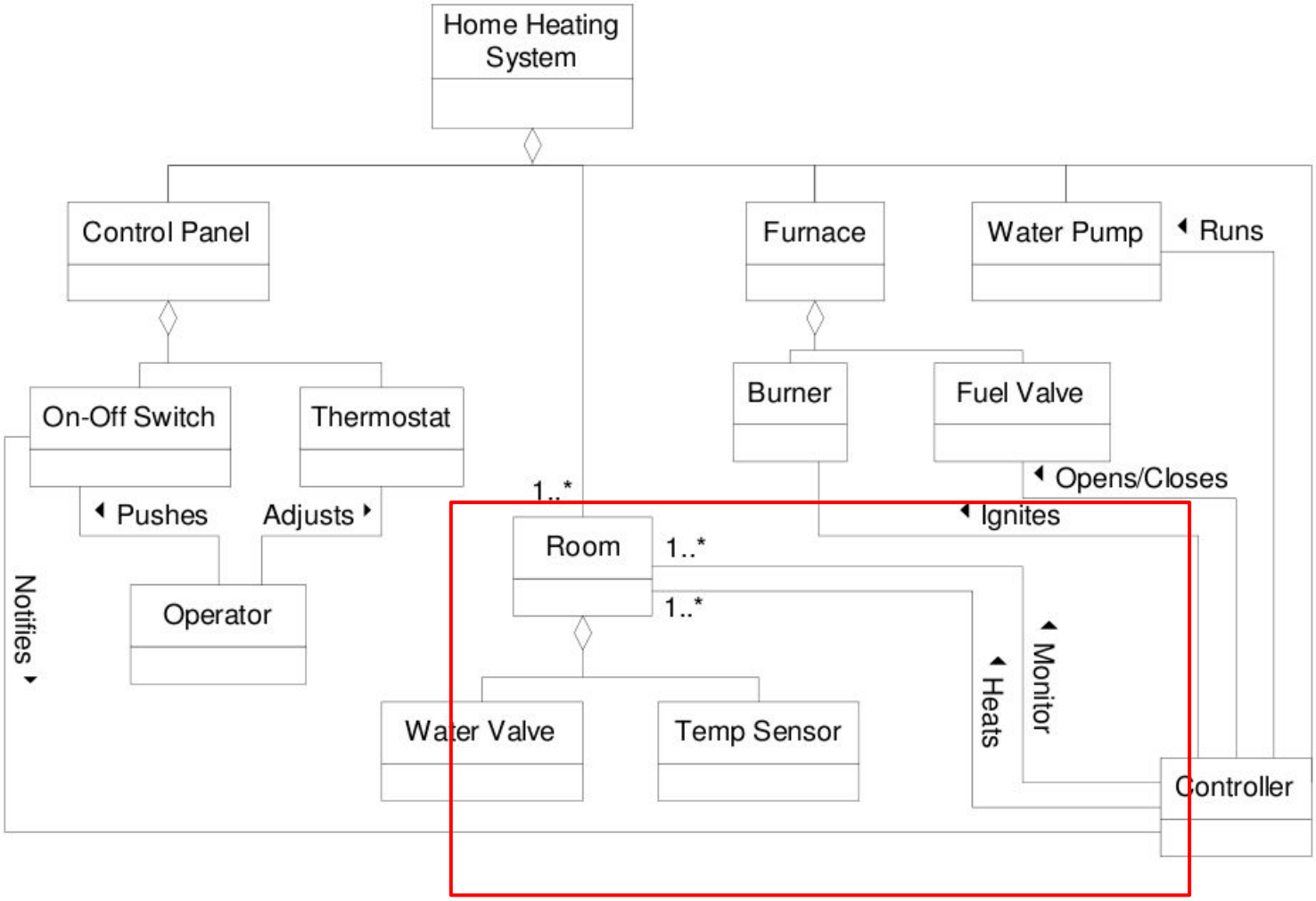
Class Diagram



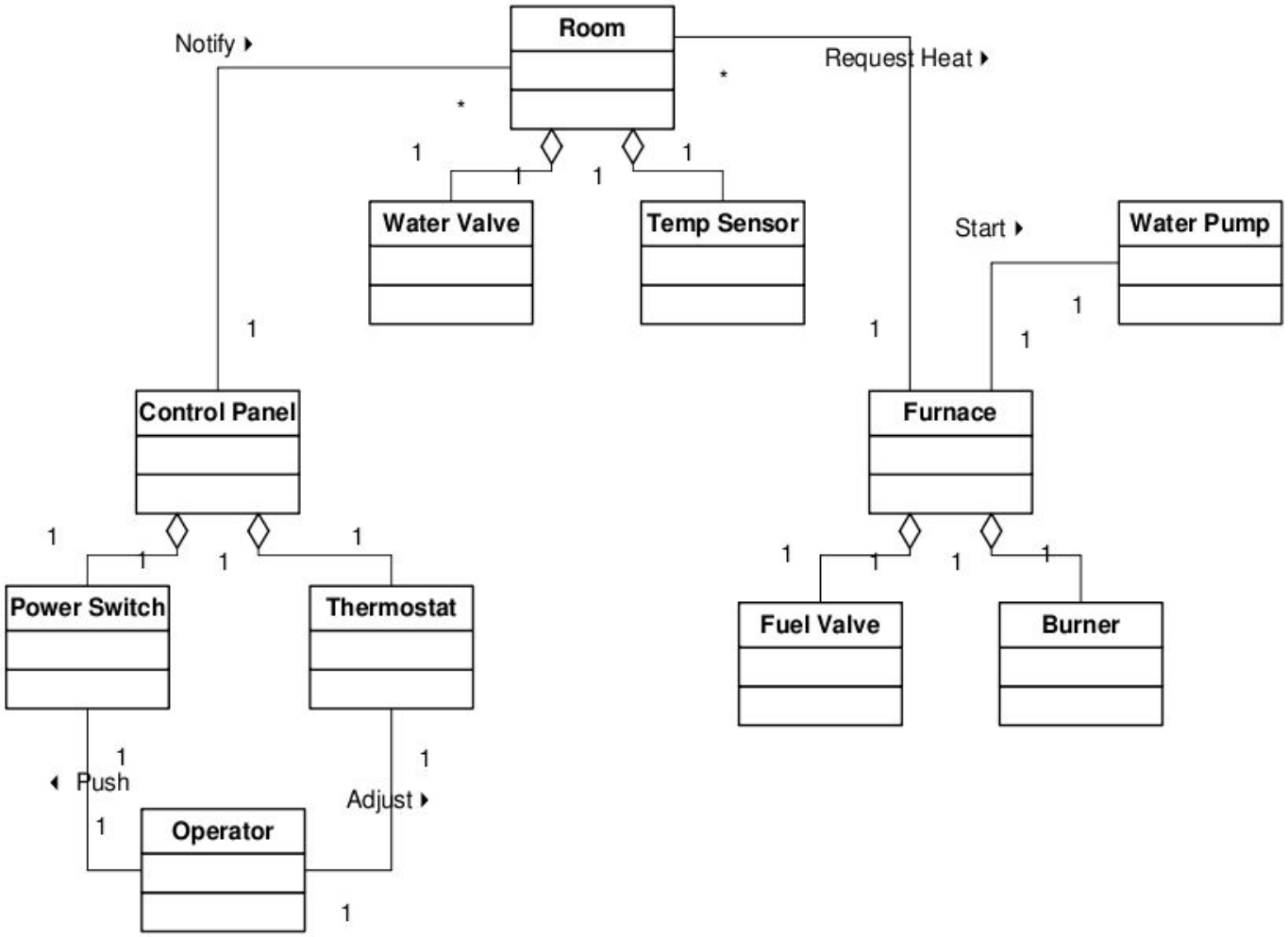
Refinement 1 - Better?



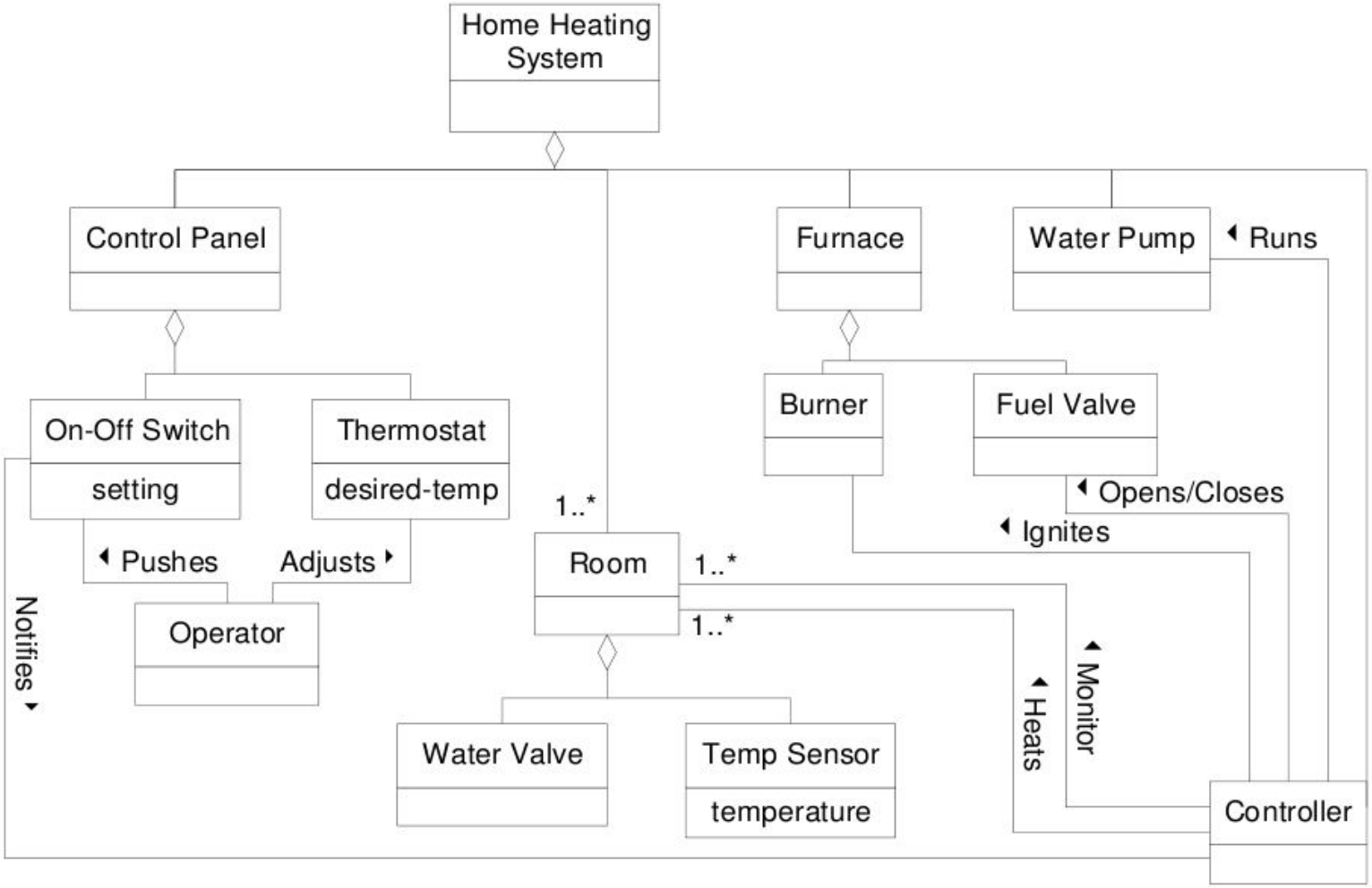
Class Diagram - Round 2



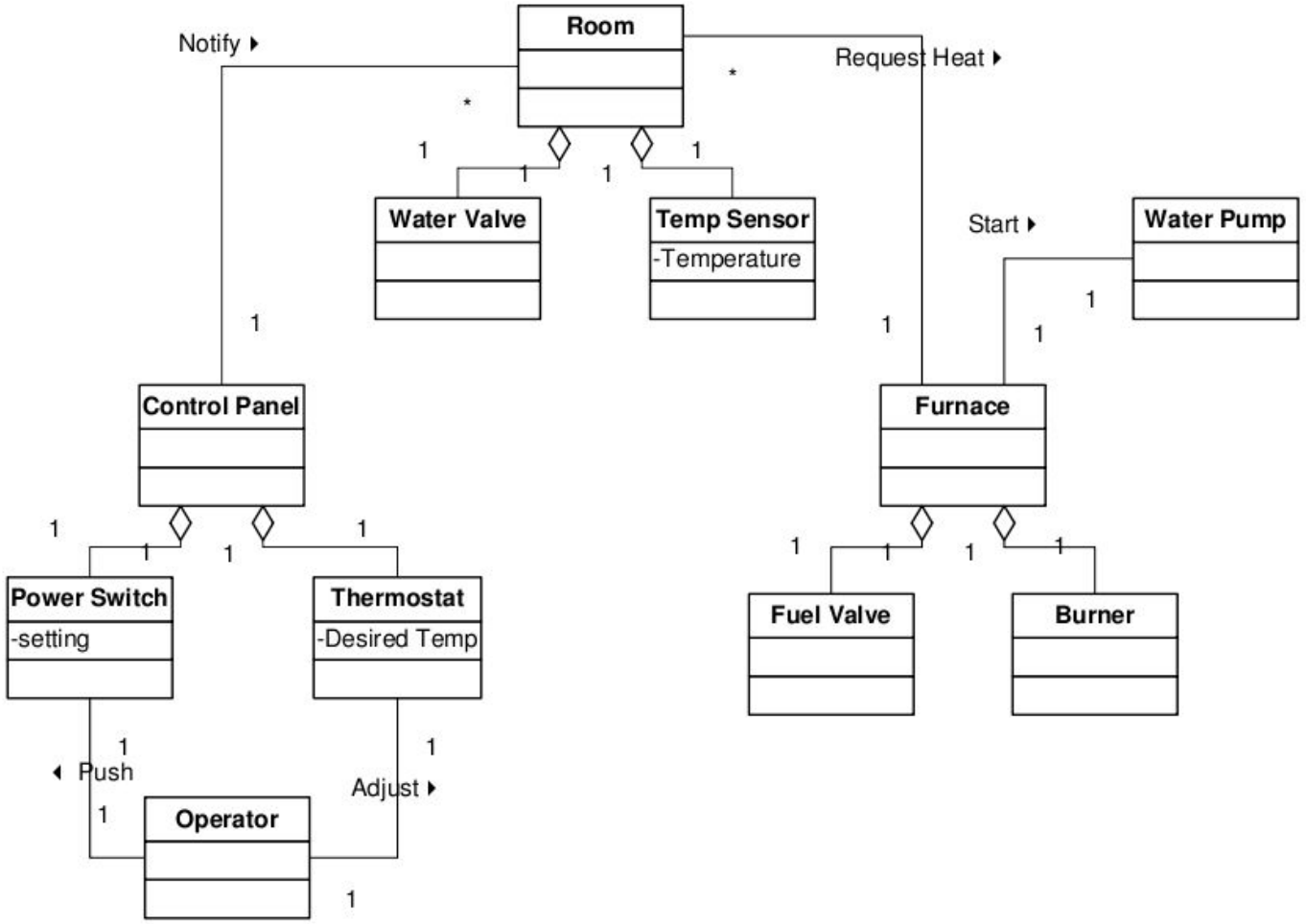
Class Diagram - Alternate



What about Attributes?



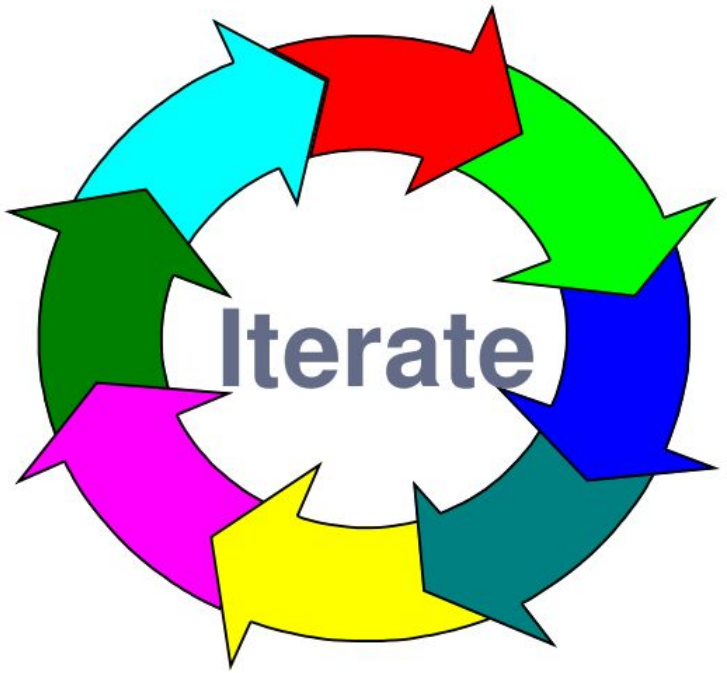
Attributes - Alternate



Iterate the Model

Keep on iterating until you, your customers, and your engineers are happy with the design.

Any questions on class diagrams?



We Have Learned

- How to approach an OO modeling effort
  - Identify objects (nouns)
  - Identify operations and associations (verbs)
  - Identify attributes.
  - Refine, refine, refine!
- The model will need a lot of iteration.
  - And often requires a dynamic view of the system as well (we'll get to that soon).

Next Time

- Design Patterns
  - Design advice for certain types of systems.
- Reading
  - Sommerville, chapter 7
- Start working on class diagrams for GRADS.
- Questions?

Preparing for Implementation

Choosing Data Structures

Design documents detail *what is being stored*, but not *how to store it.*

Choice of data structure matters:

- Storage and operation costs
- Suitability to problem (and what data is being stored)
- Many guidelines out there - key is to think through the problem and your priorities (ease-of-use vs efficiency)

Choosing Algorithms

Design gives you *what a method should do*, implementation concerns *how to code it to do that.*

Many ways to solve a problem, think carefully about choice.

- Good design may suggest certain realization.
- Be prepared to trade efficiency for maintainability or understandability.

Error-Prone Constructs

Should NOT always be avoided, but must be used with great care.

- Floating-point numbers
  - Inherently imprecise. The imprecision may lead to invalid comparisons.
- Pointers
  - Pointers referring to the wrong memory areas can corrupt data.
  - Aliasing can make programs difficult to understand and change.

Error-Prone Constructs

- Dynamic memory allocation
  - Run-time allocation can cause memory overflow and garbage collection issues.
- Parallelism
  - Can result in subtle timing errors because of unforeseen interaction between parallel processes.
- Recursion
  - Errors in recursion can cause memory overflow.
- Interrupts
  - Can cause a critical operation to be terminated and make a program difficult to understand.

Code Reuse

Most modern software is constructed, in part, by reusing existing components or systems.

- When developing software, consider how to make use of existing code.
- Possible at many levels of development.
- Be careful - many problems and costs associated with reuse.

Code Reuse Levels

1. Abstraction Level
   Use knowledge from similar projects in your system design (design/architectural patterns)
2. Object Level
   Import individual objects and functions from libraries and use them in your project.
3. Component Level
   Incorporate collections of objects and adapt them to your needs.
4. System Level
   Reuse complete applications, wired together with scripting code.

Costs of Code Reuse

- The time spent looking for software to reuse and addressing whether it fits your needs can be high.
- Buying and licensing software for reuse can be expensive.
- Cost of adapting and configuring the reusable components to fit your requirements can be more expensive than coding yourself.
- Integrating reused systems with each other and with your new code can result in new defects.

Host-Target Development

Most software is developed on one type of computer (the host) and deployed on different types of computers (targets).

- For embedded systems, the target is **very** different from the host.
- For desktop applications, still need to consider a wide variety of target environments.

Target Support Issues

- The hardware and software requirements of a component.
  - If a component is designed for a specific hardware architecture, requires certain CPU/RAM/GPU, or requires special software, then make sure those assumptions are clearly stated.
- The availability requirements of the system.
  - Components may be deployed on multiple platforms. Make sure an alternative implementation of the component is available if one fails.
- Component Communications
  - If distributed components must communicate, try to install them on a single system or ensure geographically close servers exist.

Managing Change

Change happens all the time, so managing change is essential.

- When teams work together, their work must not conflict.
  - Changes must be coordinated. Otherwise, one programmer may overwrite the other's work.
  - Everybody must have access to the most up-to-date versions of all project components.
- If something is broken, we should be able to go back to the working version.

Configuration Management

The process of managing a changing system.
Three fundamental activities:

1. Version Management
   Different versions of system components are tracked. Coordinates development by several programmers. Prevents overwriting of code.
2. System Integration
   Support is provided to help developers define what versions of a component are used to create a system build. Supports automated builds by linking components.
3. Problem Tracking
   Allow users to report bugs and other problems, and allow developers to see who is working on these problems.