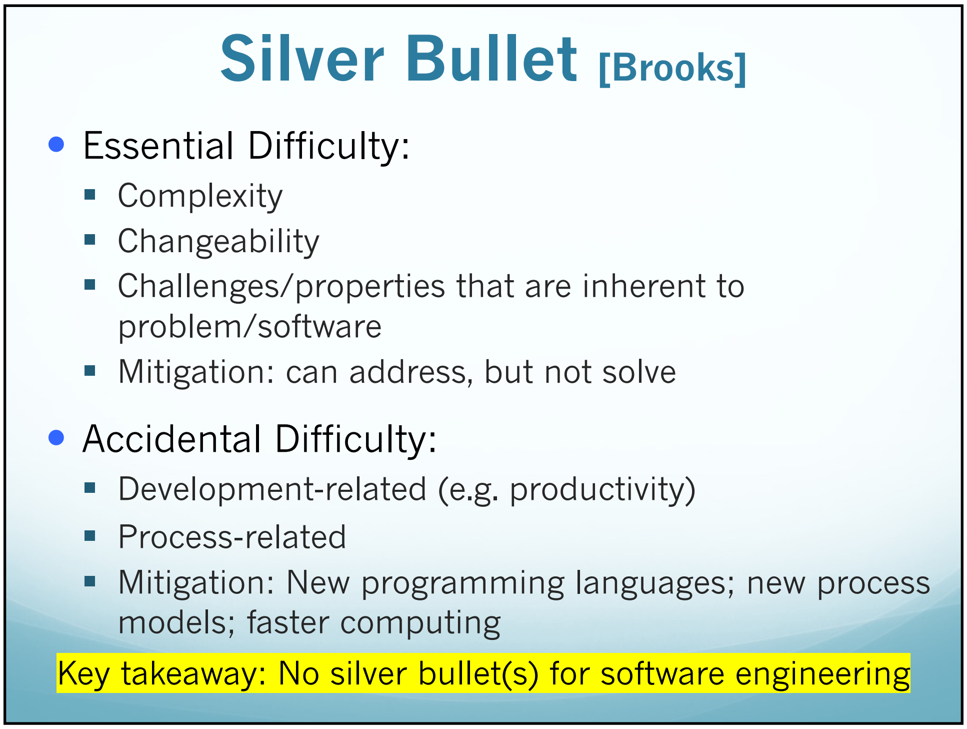
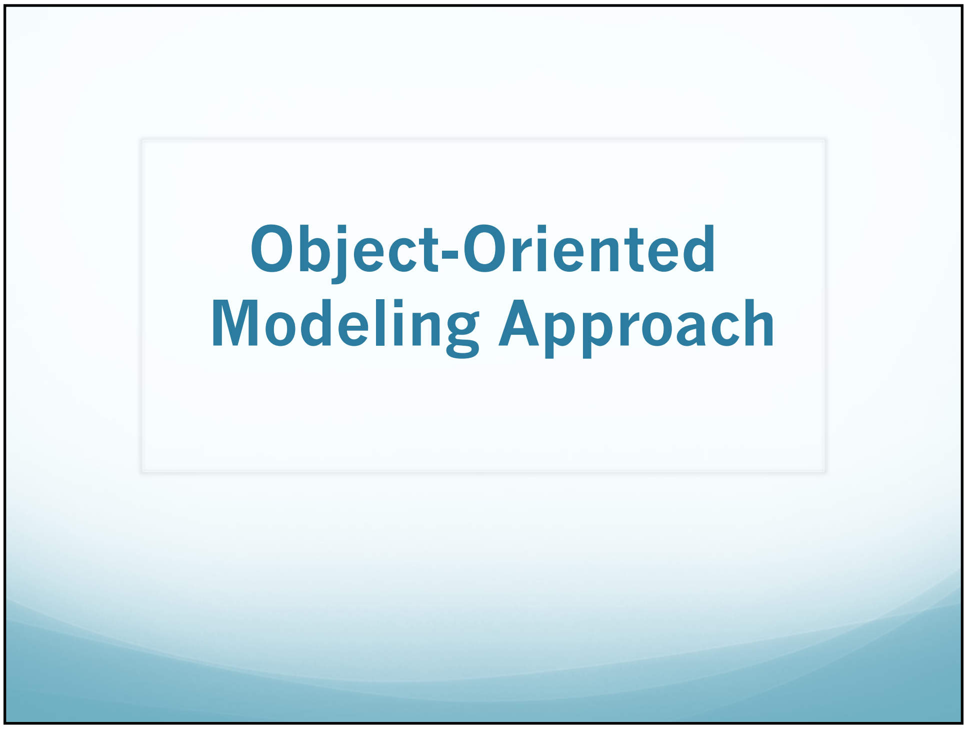


Silver Bullet [Brooks]

- Essential Difficulty:
  - Complexity
  - Changeability
  - Challenges/properties that are inherent to problem/software
  - Mitigation: can address, but not solve
- Accidental Difficulty:
  - Development-related (e.g. productivity)
  - Process-related
  - Mitigation: New programming languages; new process models; faster computing

Key takeaway: No silver bullet(s) for software engineering

1



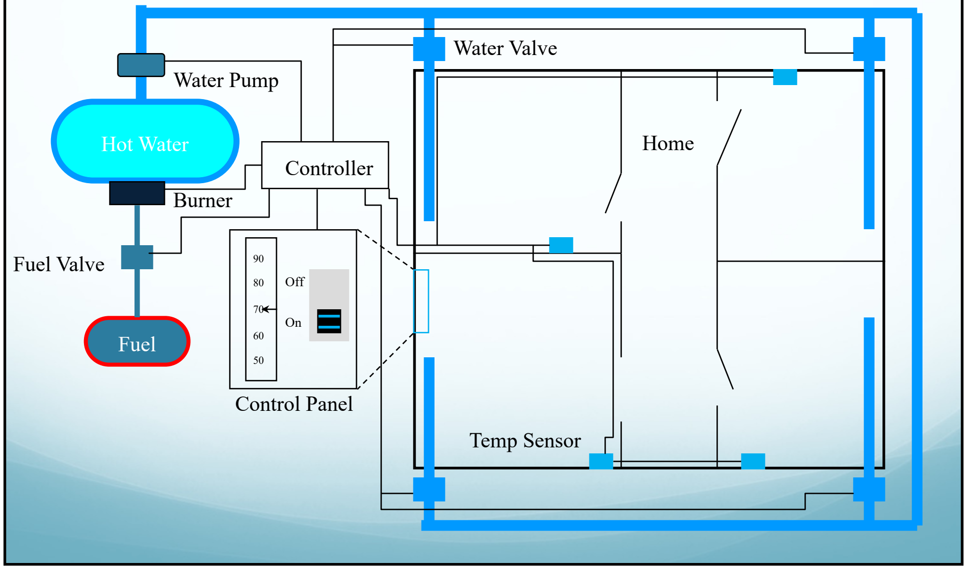
Object-Oriented Modeling Approach

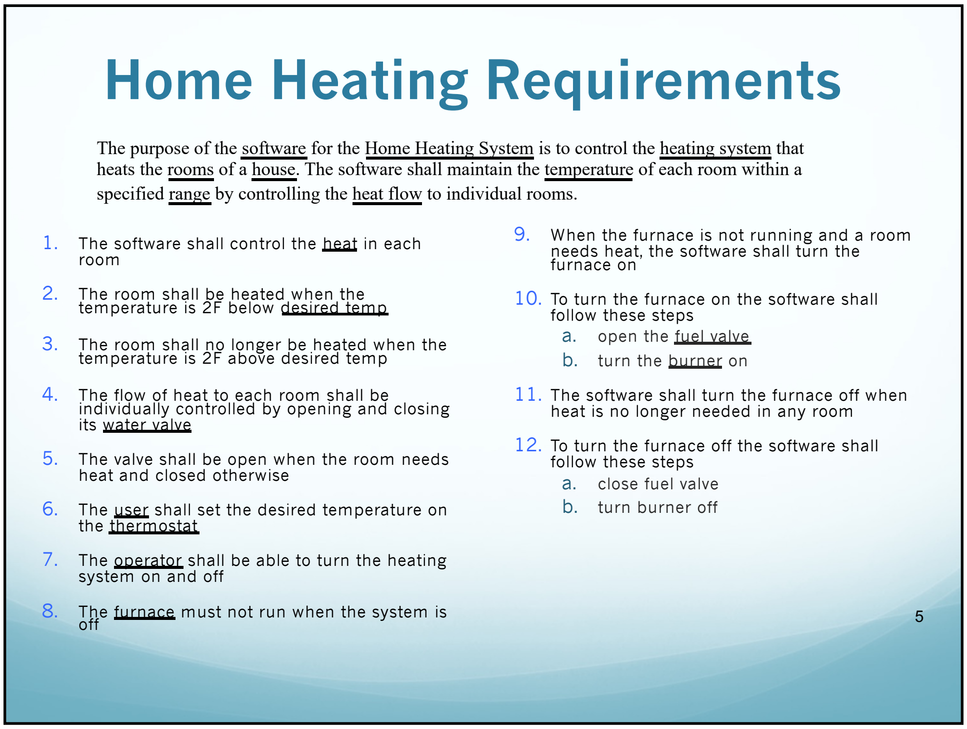
2

Object-Oriented (OO) Modeling Approach

- Start with a problem statement
  - High-level requirements
- Define domain model (high-level class diagram)
  - Identify key elements in the system
  - 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





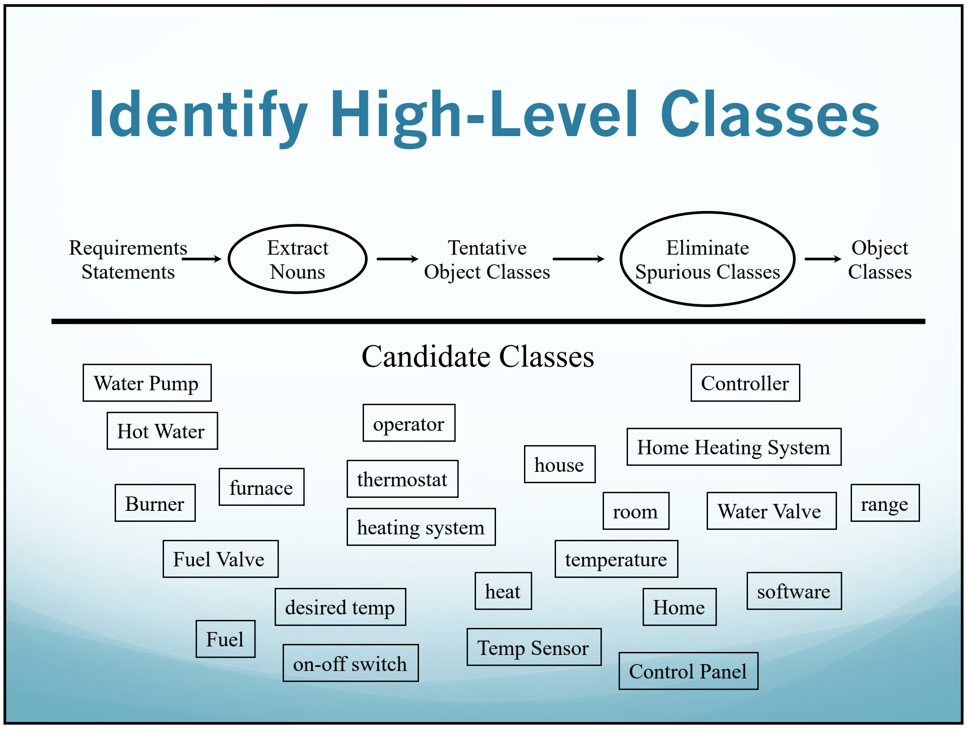
Home Heating Requirements

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.

1. The software shall control the heat in each room
2. The room shall be heated when the temperature is 2F below desired temp
3. The room shall no longer be heated when the temperature is 2F above desired temp
4. The flow of heat to each room shall be individually controlled by opening and closing its water valve
5. The valve shall be open when the room needs heat and closed otherwise
6. The user shall set the desired temperature on the thermostat
7. The operator shall be able to turn the heating system on and off
8. The furnace must not run when the system is off
9. When the furnace is not running and a room needs heat, the software shall turn the furnace on
10. To turn the furnace on the software shall follow these steps
    a. open the fuel valve
    b. turn the burner on
11. The software shall turn the furnace off when heat is no longer needed in any room
12. To turn the furnace off the software shall follow these steps
    a. close fuel valve
    b. turn burner off

5

5



Identify High-Level Classes

Requirements Statements → Extract Nouns → Tentative Object Classes → Eliminate Spurious Classes → Object Classes

Candidate Classes

Water Pump

Controller

Hot Water

operator

Home Heating System

house

thermostat

furnace

Burner

room

Water Valve

range

heating system

Fuel Valve

temperature

heat

software

desired temp

Home

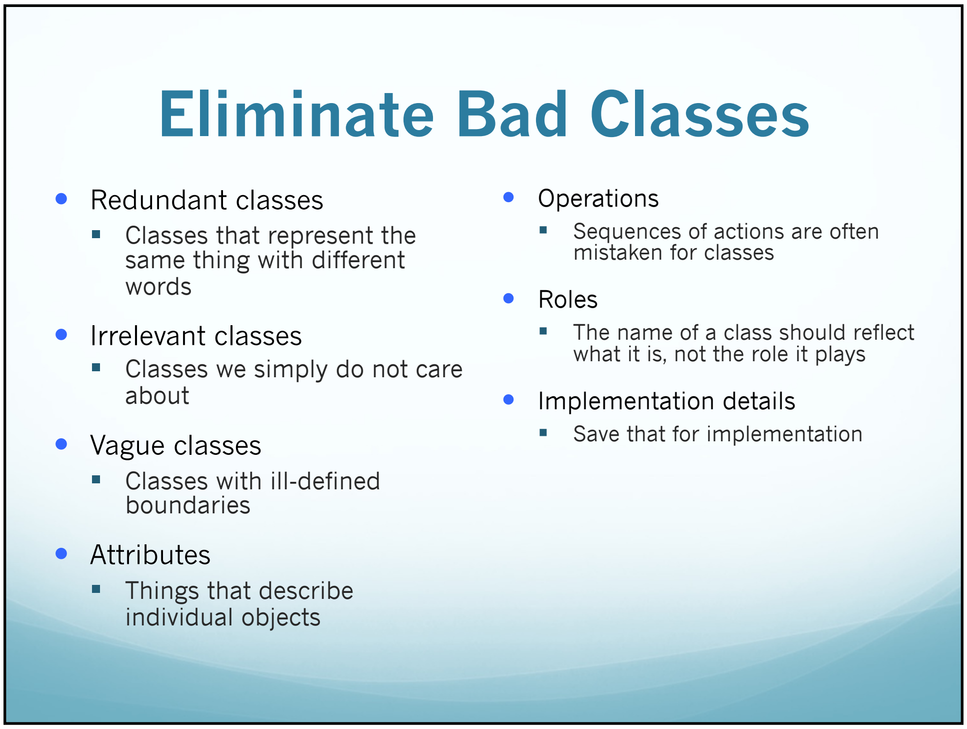
Fuel

Temp Sensor

on-off switch

Control Panel

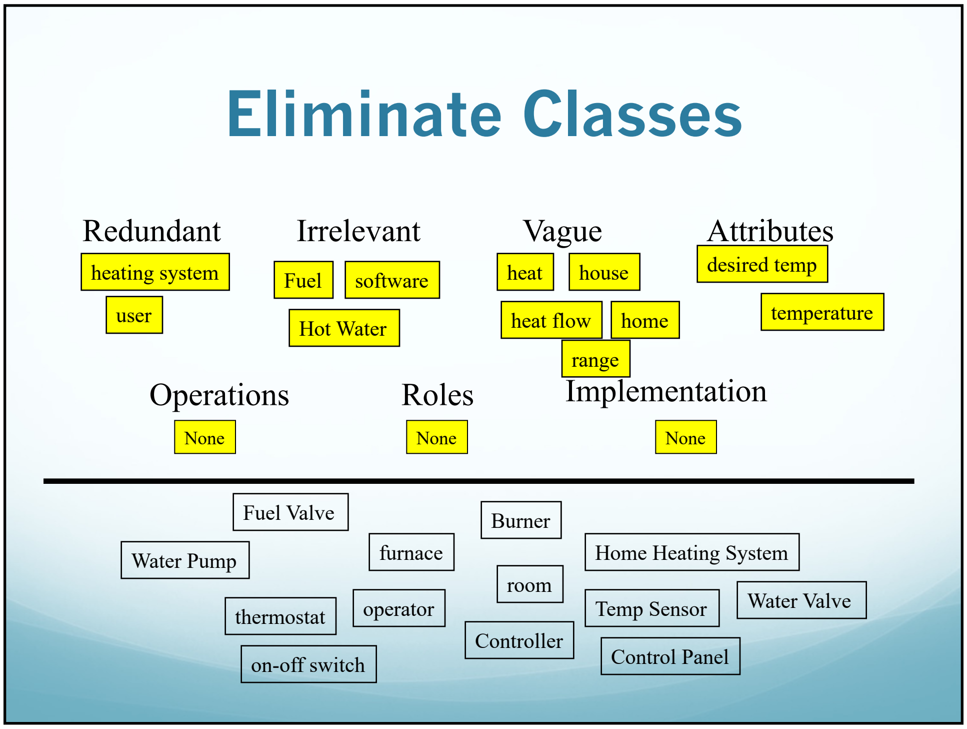
9



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

10



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

Fuel Valve

Burner

Water Pump

furnace

Home Heating System

room

thermostat

operator

Temp Sensor

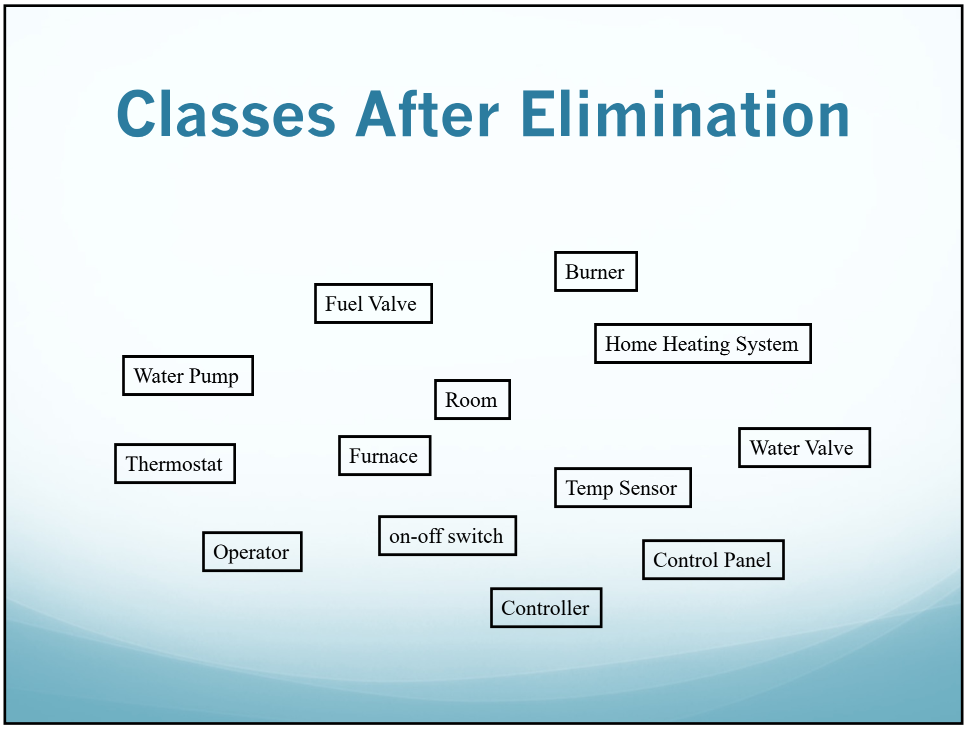
Water Valve

Controller

Control Panel

on-off switch

11



Classes After Elimination

Burner

Fuel Valve

Home Heating System

Water Pump

Room

Water Valve

Thermostat

Furnace

Temp Sensor

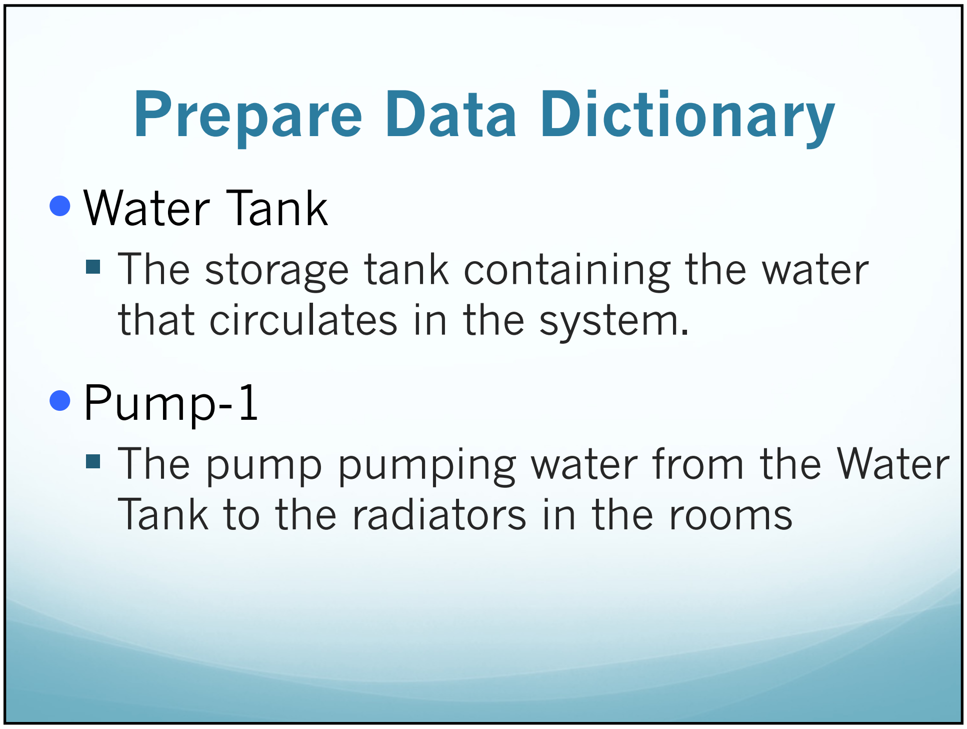
on-off switch

Operator

Control Panel

Controller

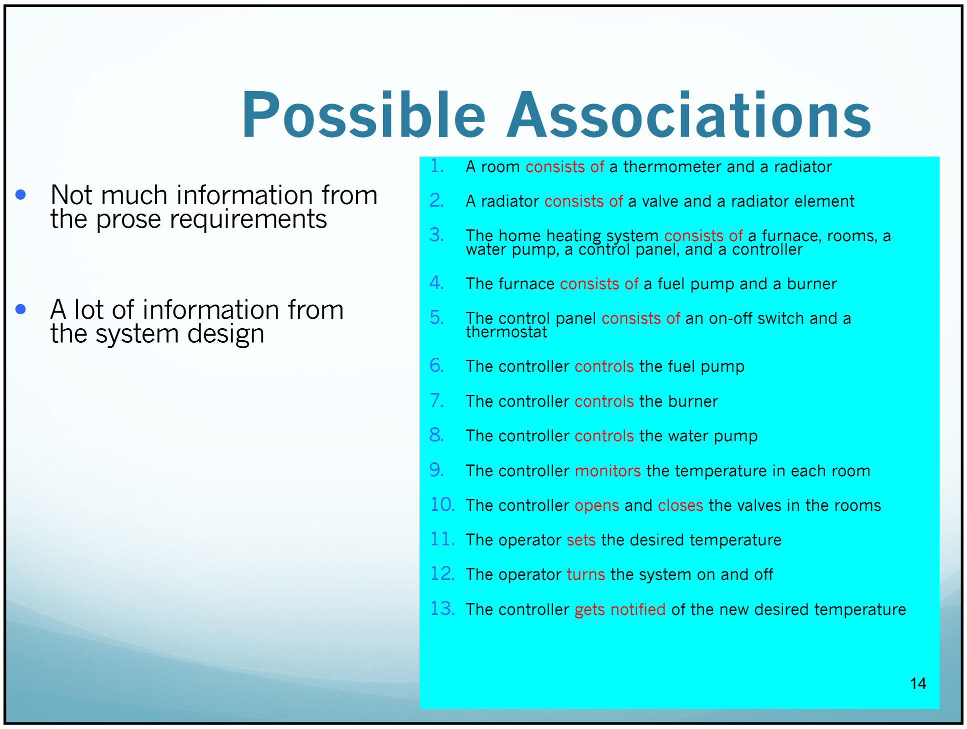
12



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

13



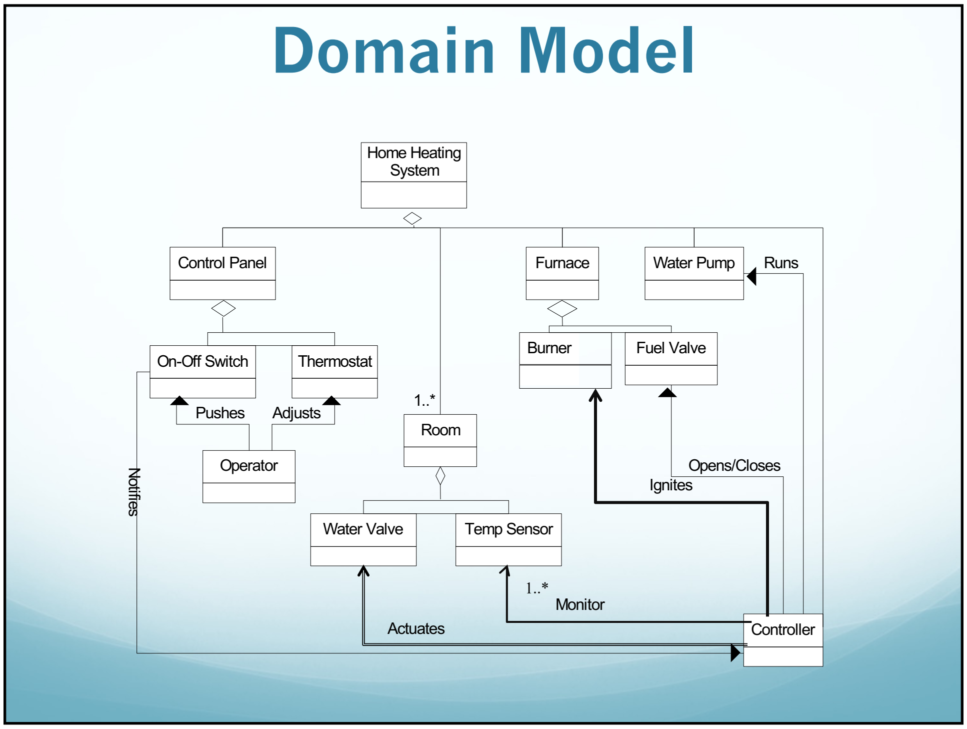
Possible Associations

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

1. A room consists of a thermometer and a radiator
2. A radiator consists of a valve and a radiator element
3. The home heating system consists of a furnace, rooms, a water pump, a control panel, and a controller
4. The furnace consists of a fuel pump and a burner
5. The control panel consists of an on-off switch and a thermostat
6. The controller controls the fuel pump
7. The controller controls the burner
8. The controller controls the water pump
9. The controller monitors the temperature in each room
10. The controller opens and closes the valves in the rooms
11. The operator sets the desired temperature
12. The operator turns the system on and off
13. The controller gets notified of the new desired temperature

14

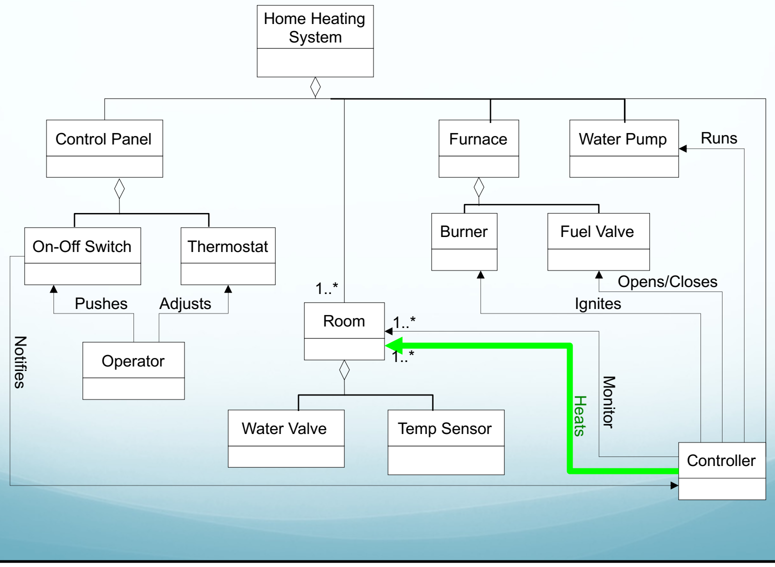
14



Domain Model

16

Object Model - Modified



17

17

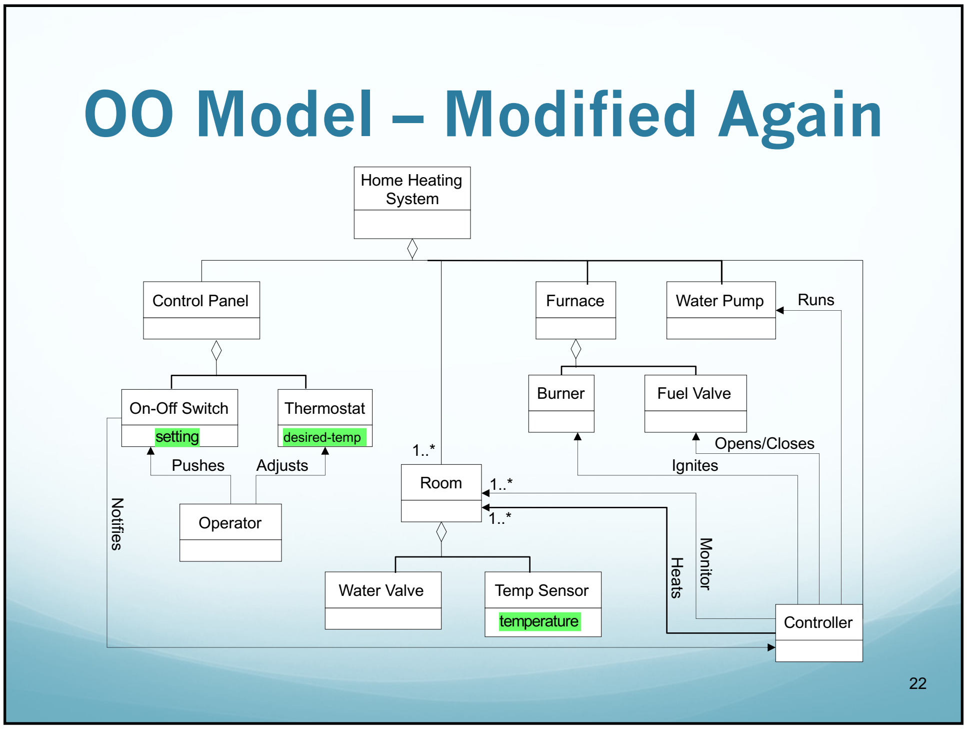
Attributes

Thermostat
desired-temp

On-Off switch
setting

Temp Sensor
temperature

21



OO Model – Modified Again

Home Heating System

Control Panel

Furnace

Water Pump

Runs

On-Off Switch

setting

Thermostat

desired-temp

Burner

Fuel Valve

Opens/Closes

1..*

Pushes

Adjusts

Ignites

Room

1..*

1..*

Notifies

Operator

Monitor

Heats

Water Valve

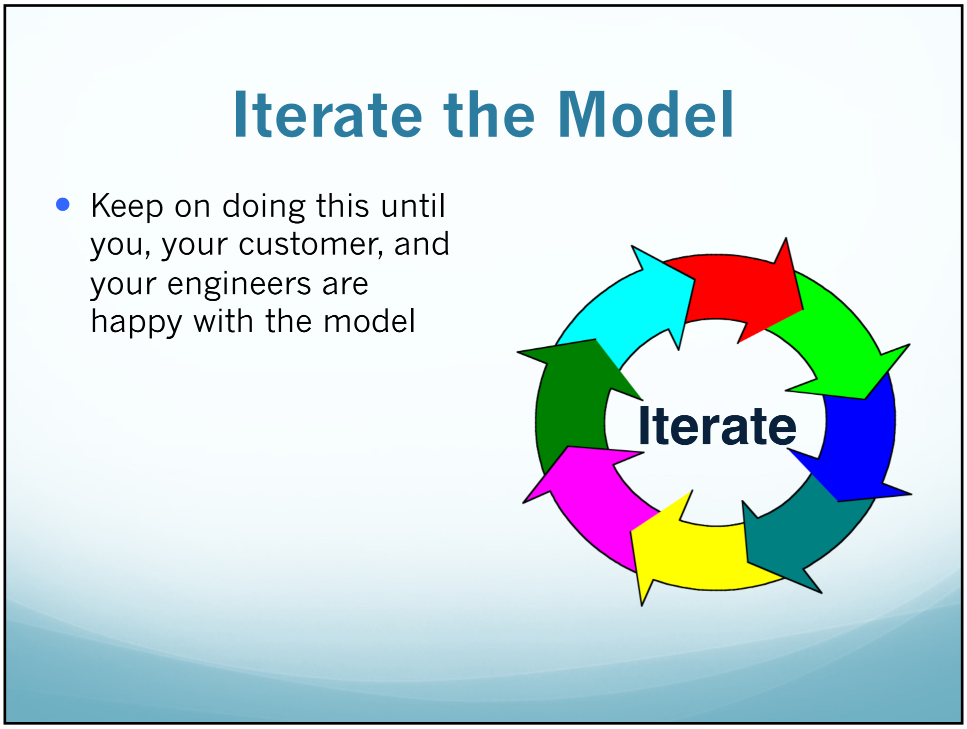
Temp Sensor

temperature

Controller

22

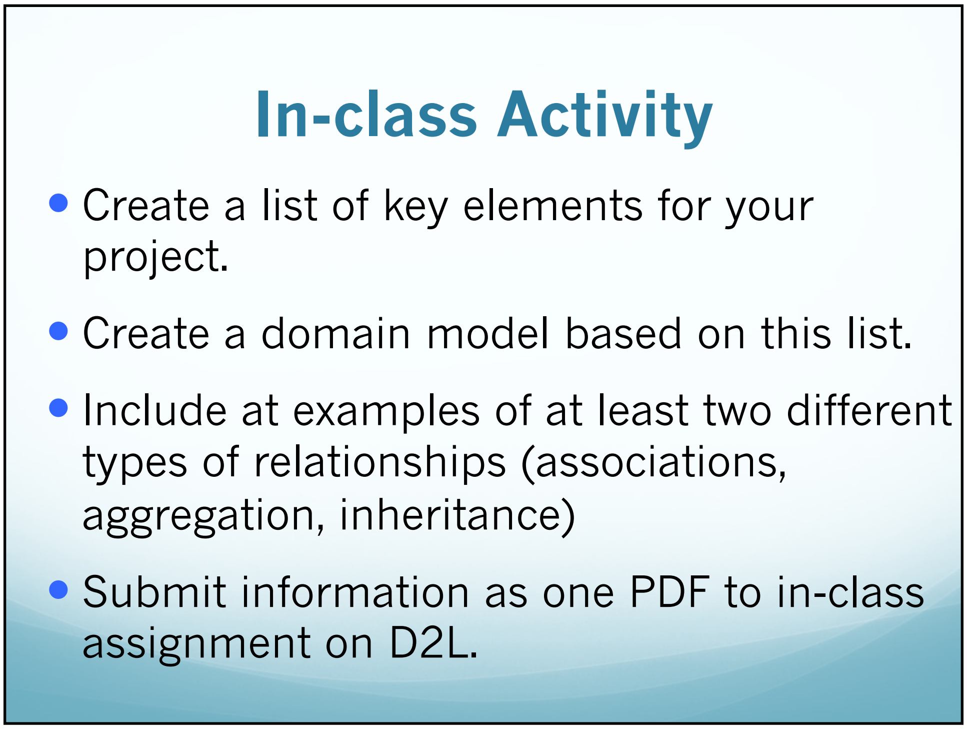
22



Iterate the Model

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

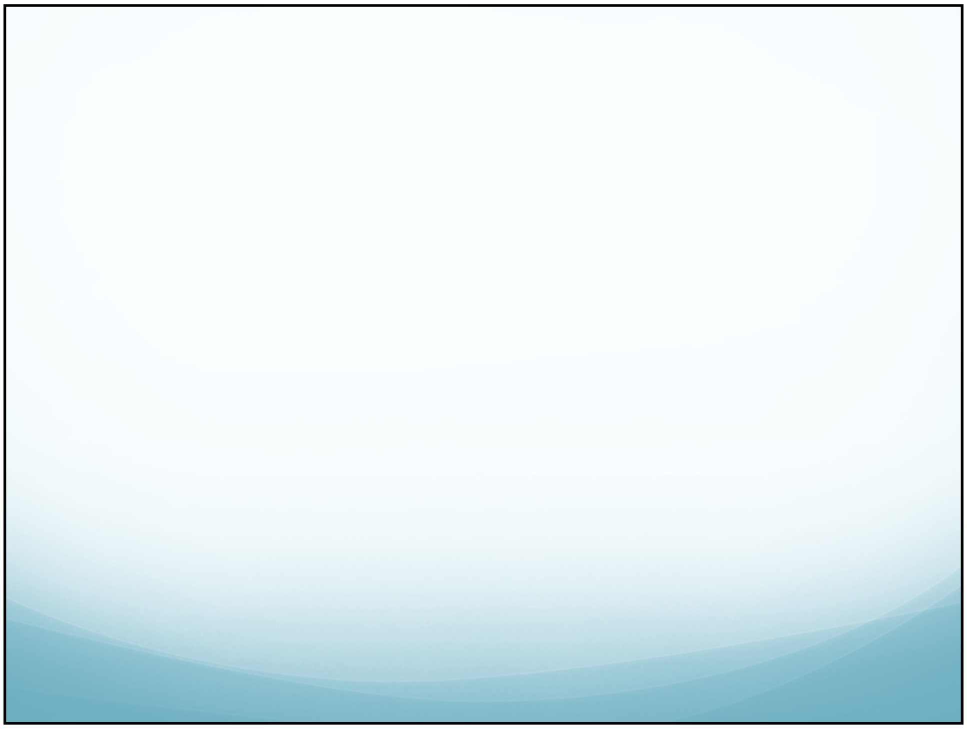
24



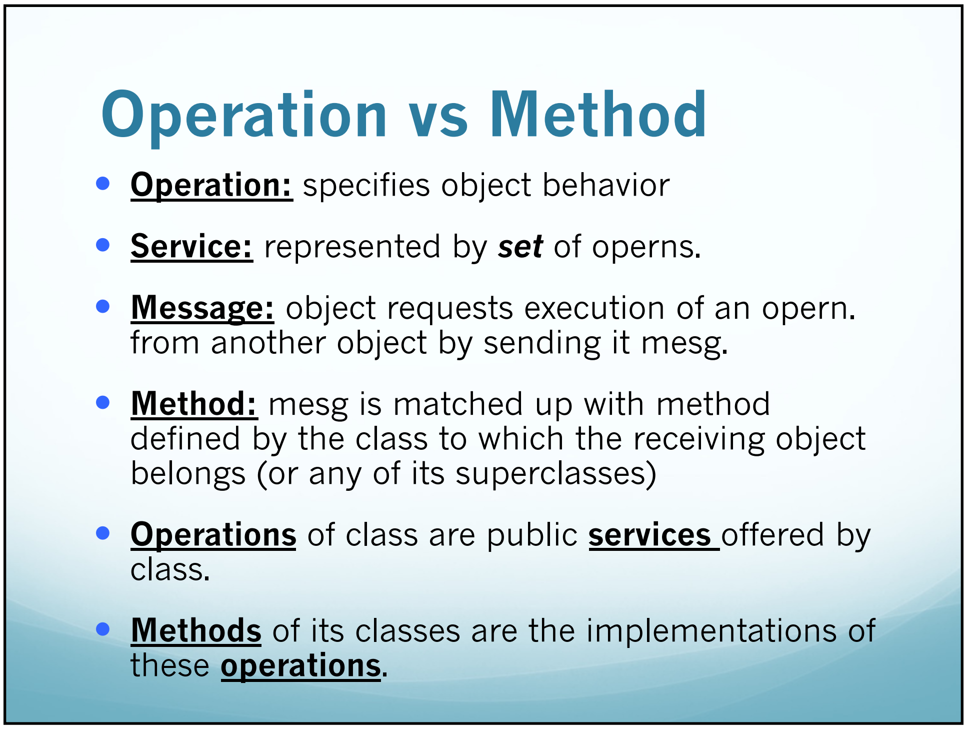
In-class Activity

- Create a list of key elements for your project.
- Create a domain model based on this list.
- Include at examples of at least two different types of relationships (associations, aggregation, inheritance)
- Submit information as one PDF to in-class assignment on D2L.

25



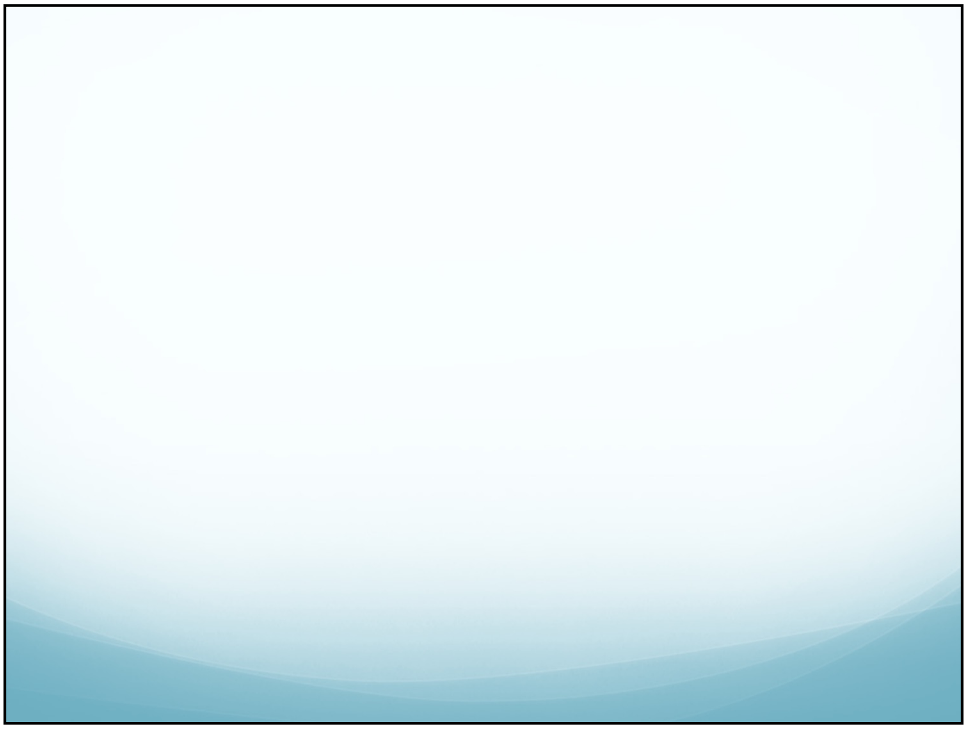
26



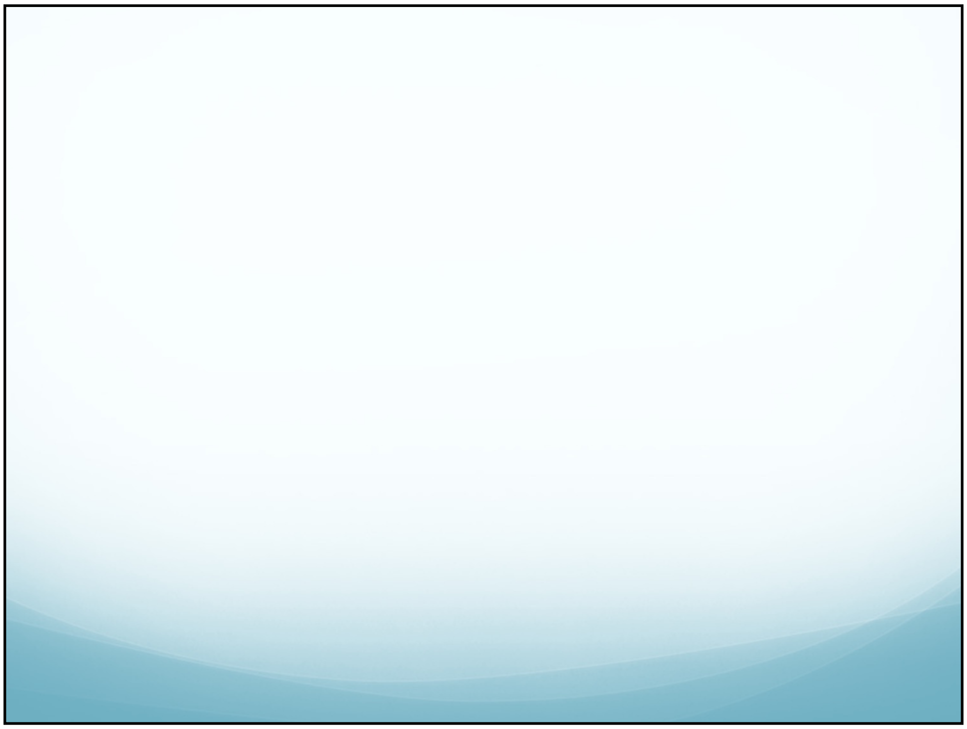
Operation vs Method

- **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**.

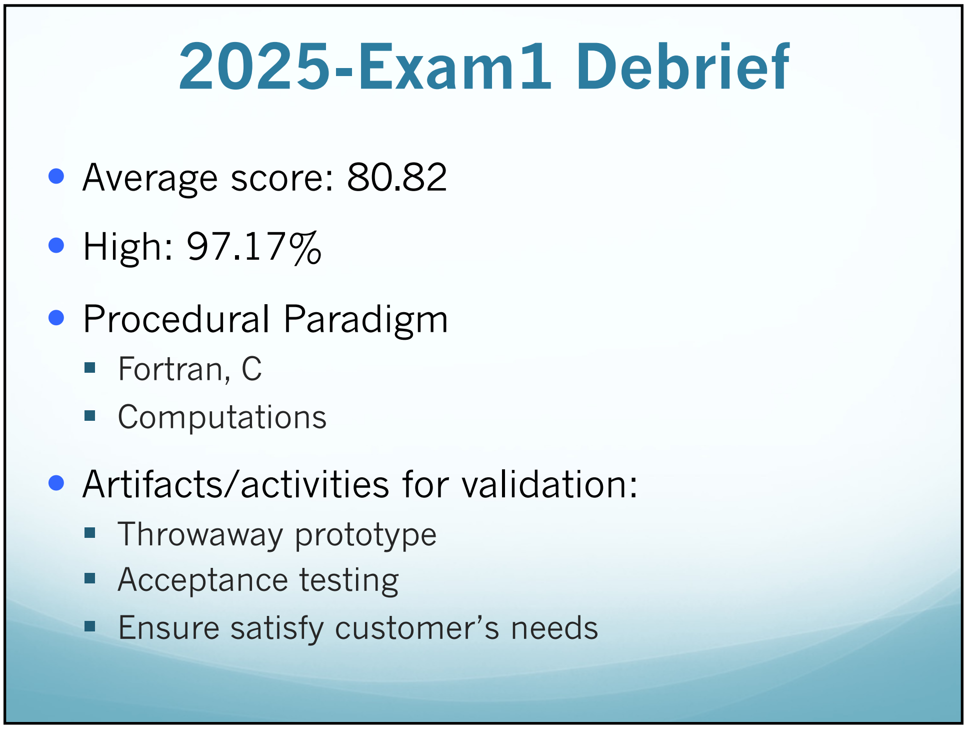
27



28



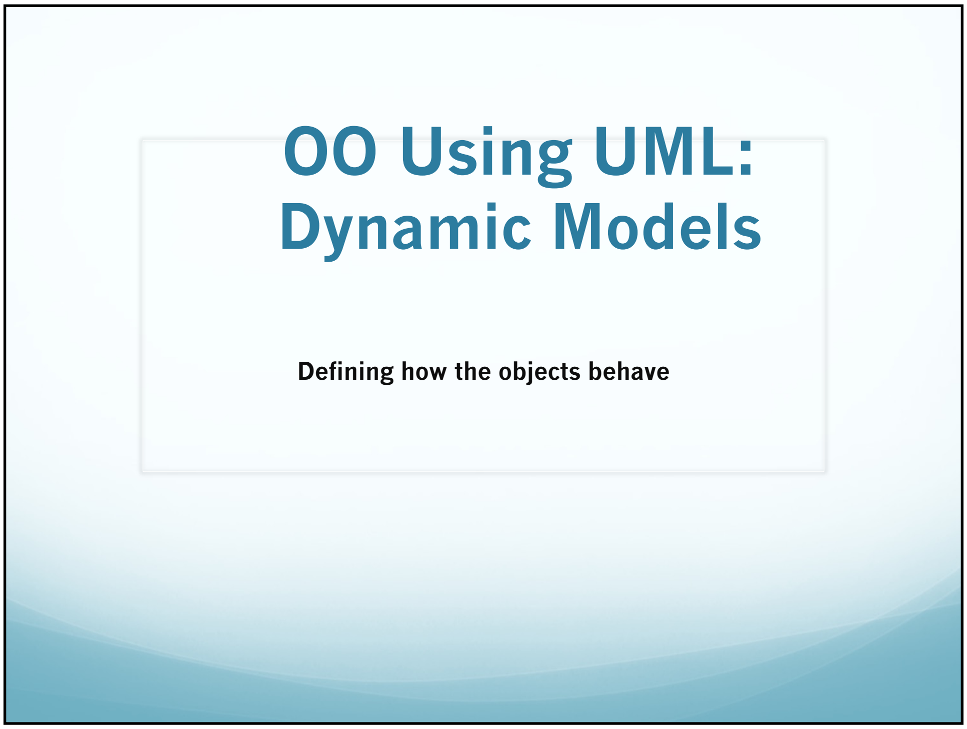
29



2025-Exam1 Debrief

- Average score: 80.82
- High: 97.17%
- Procedural Paradigm
  - Fortran, C
  - Computations
- Artifacts/activities for validation:
  - Throwaway prototype
  - Acceptance testing
  - Ensure satisfy customer’s needs

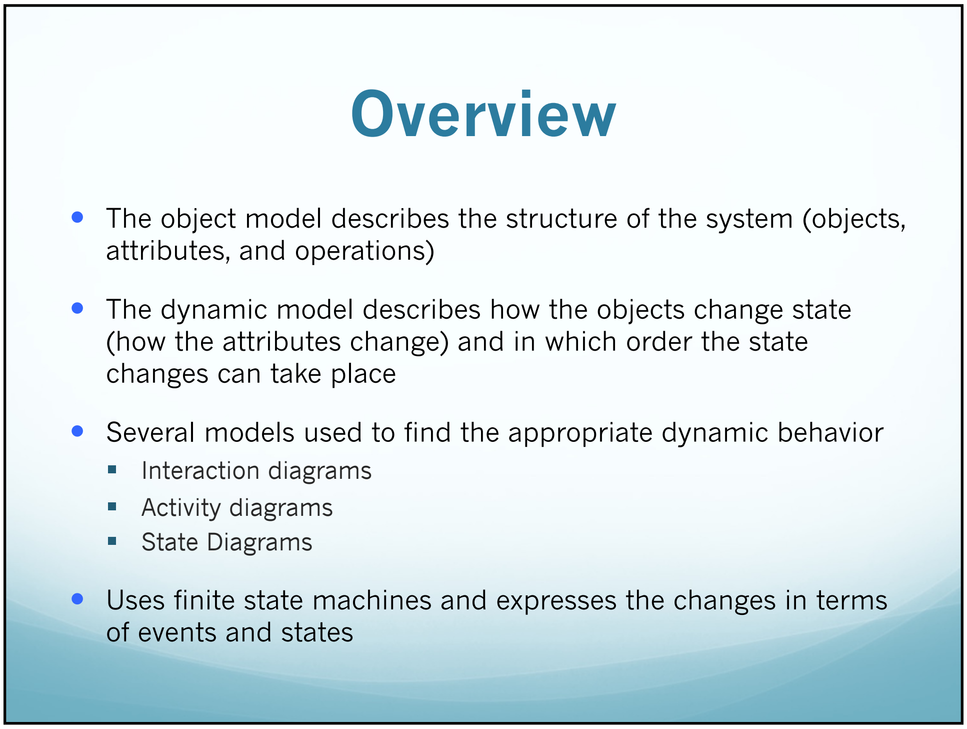
30



OO Using UML: Dynamic Models

Defining how the objects behave

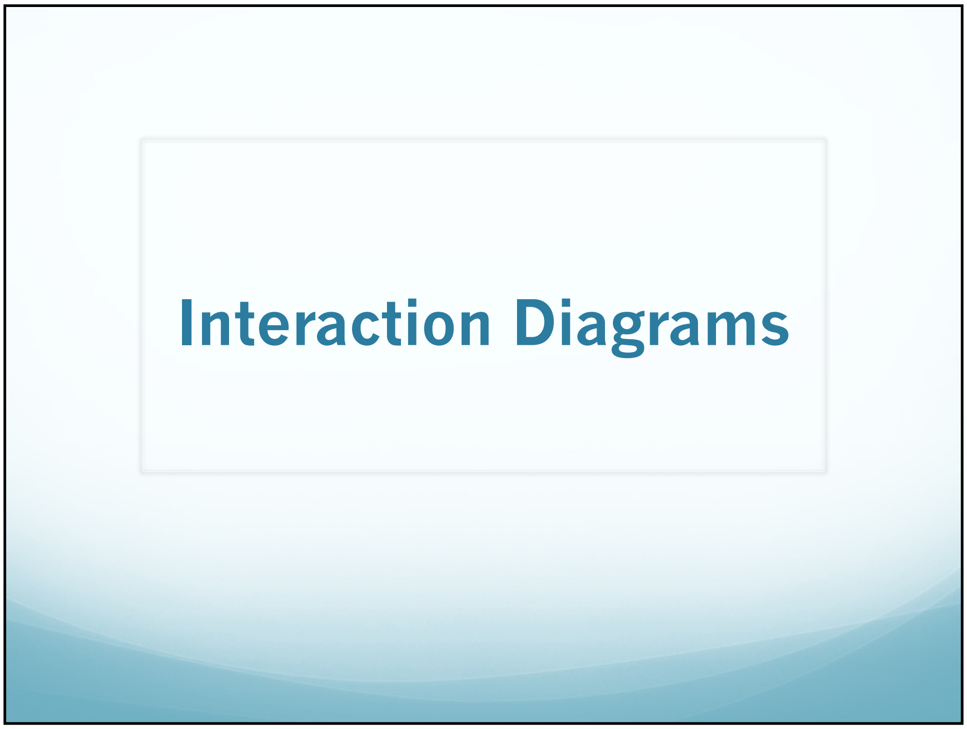
31



Overview

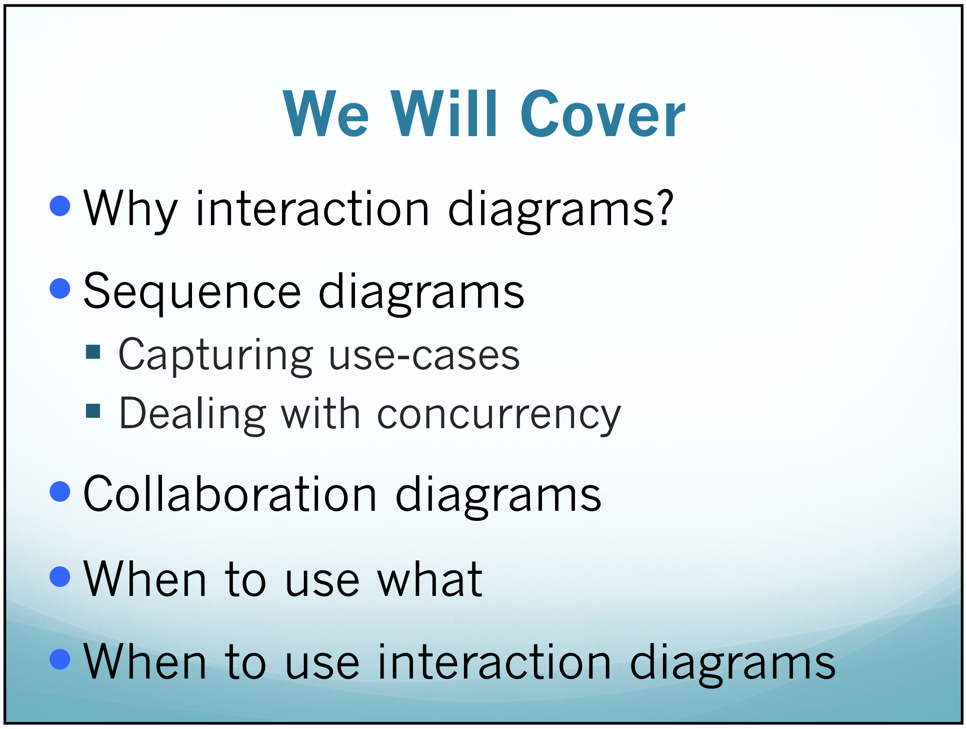
- The object model describes the structure of the system (objects, attributes, and operations)
- The dynamic model describes how the objects change state (how the attributes change) and in which order the state changes can take place
- Several models used to find the appropriate dynamic behavior
  -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

32



Interaction Diagrams

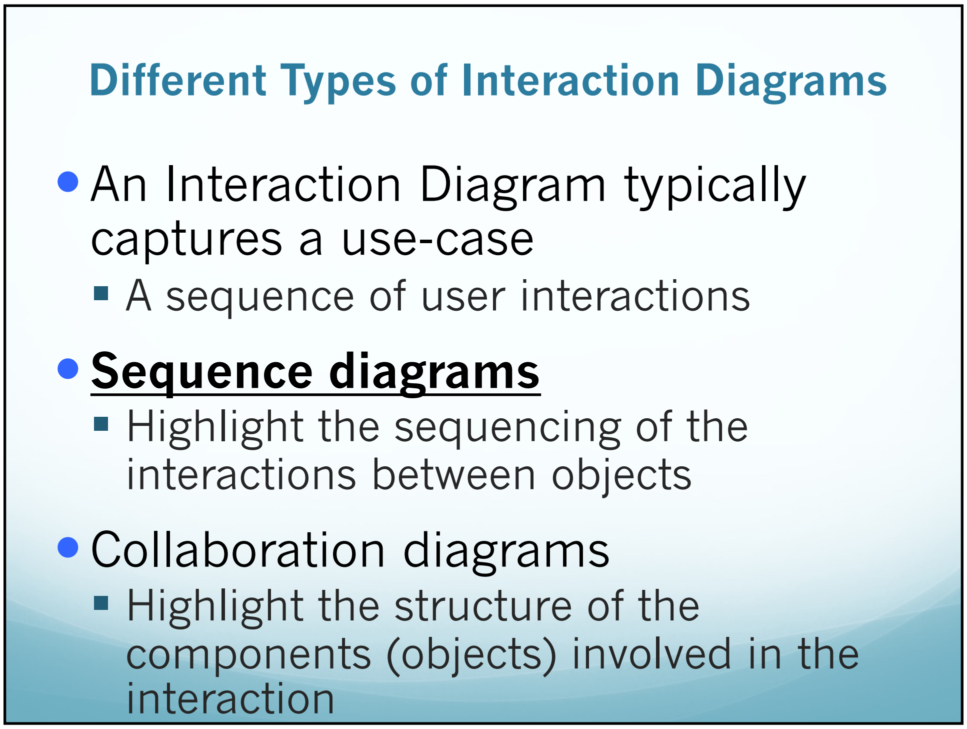
33



We Will Cover

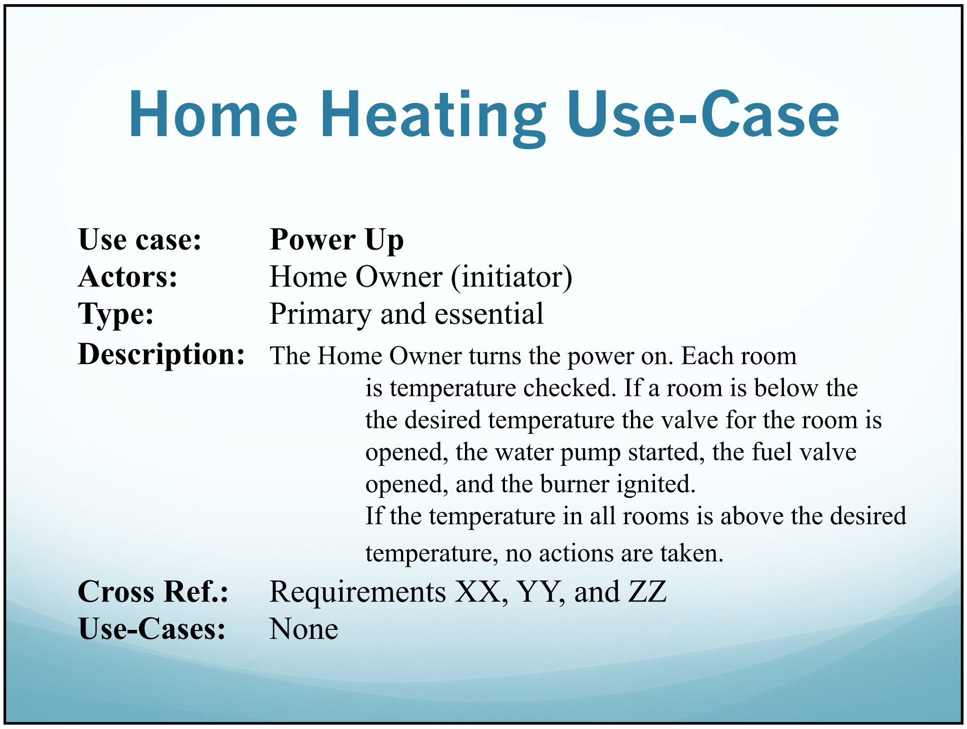
- Why interaction diagrams?
- Sequence diagrams
  - Capturing use-cases
  - Dealing with concurrency
- Collaboration diagrams
- When to use what
- When to use interaction diagrams

34



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**

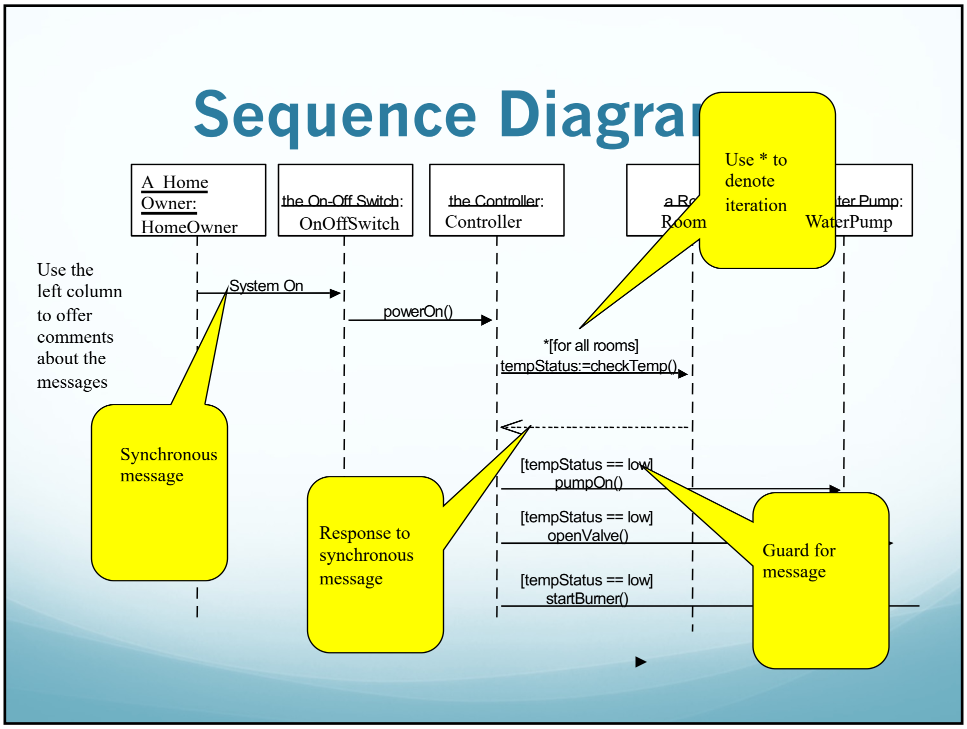
Actors: Home Owner (initiator)

Type: Primary and essential

Description: The Home Owner turns the power on. Each room is temperature checked. If a room is below the 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

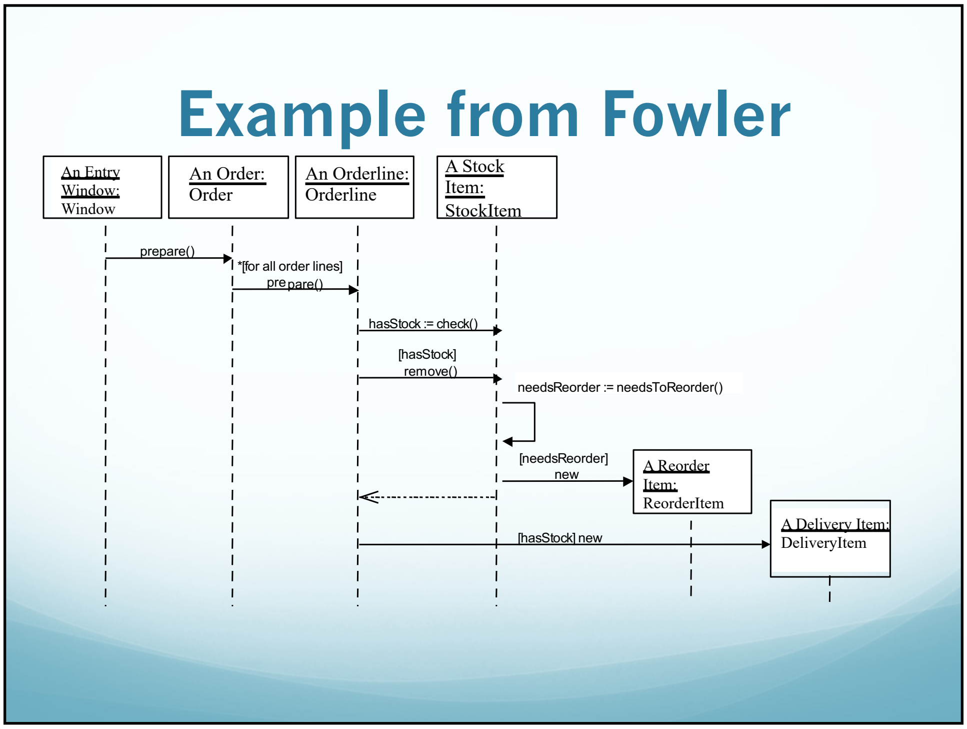
Use-Cases: None



Sequence Diagra

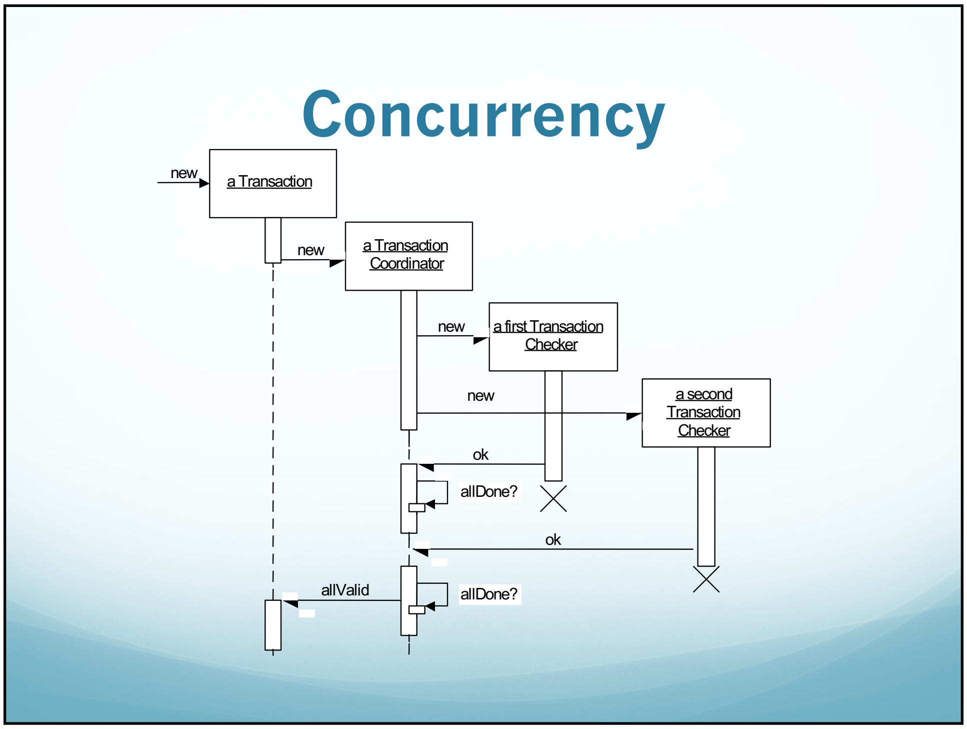
Use the left column to offer comments about the messages

37



Example from Fowler

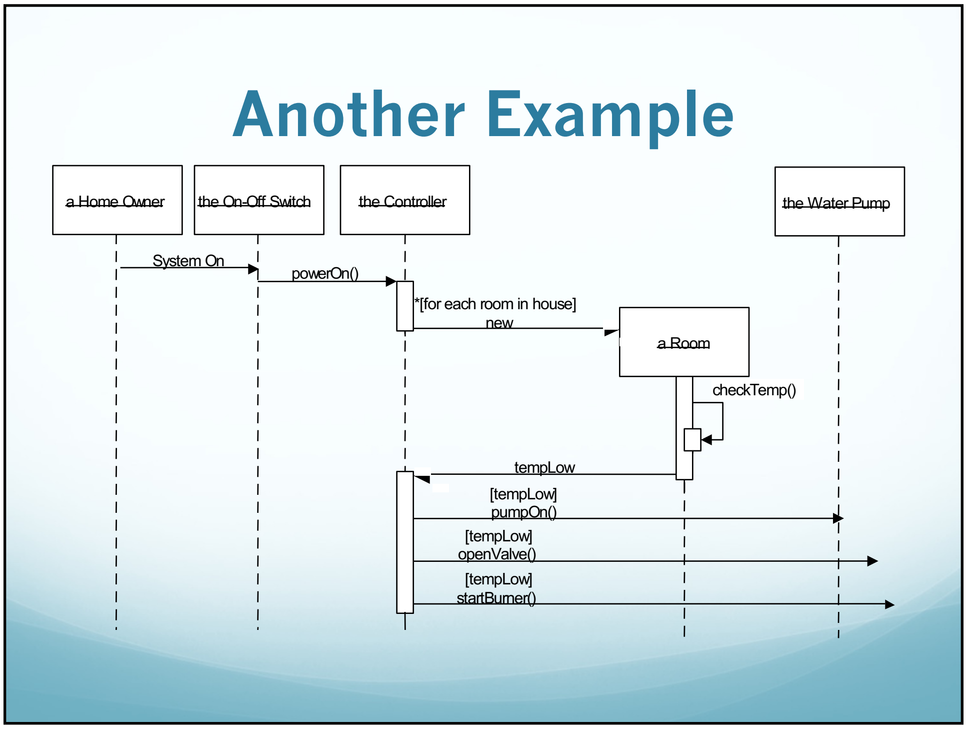
38



Concurrency

new
a Transaction
new
a Transaction Coordinator
new
a first Transaction Checker
new
a second Transaction Checker
ok
allDone?
ok
allValid
allDone?

39



Another Example

40

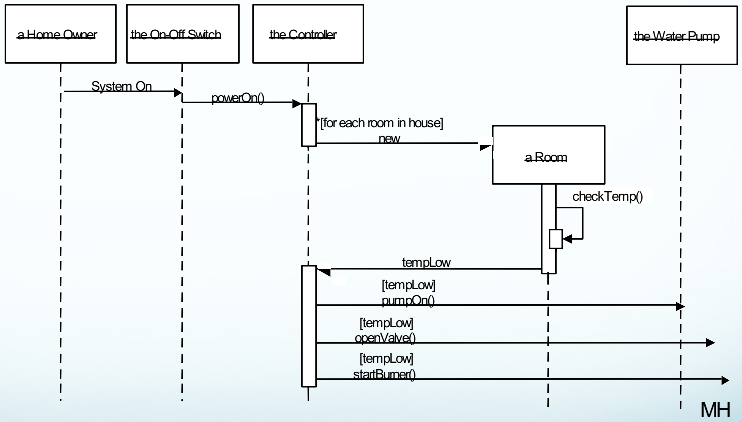
Comment the Diagram

When the owner turns the system on

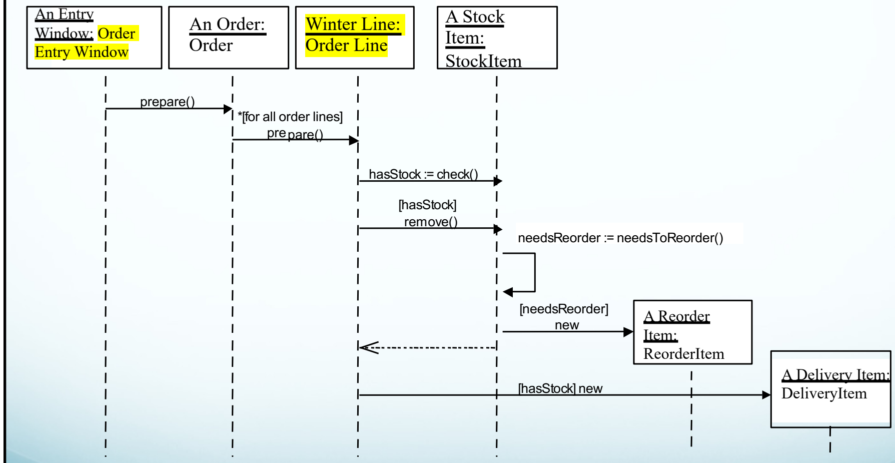
the on switch notifies the controller

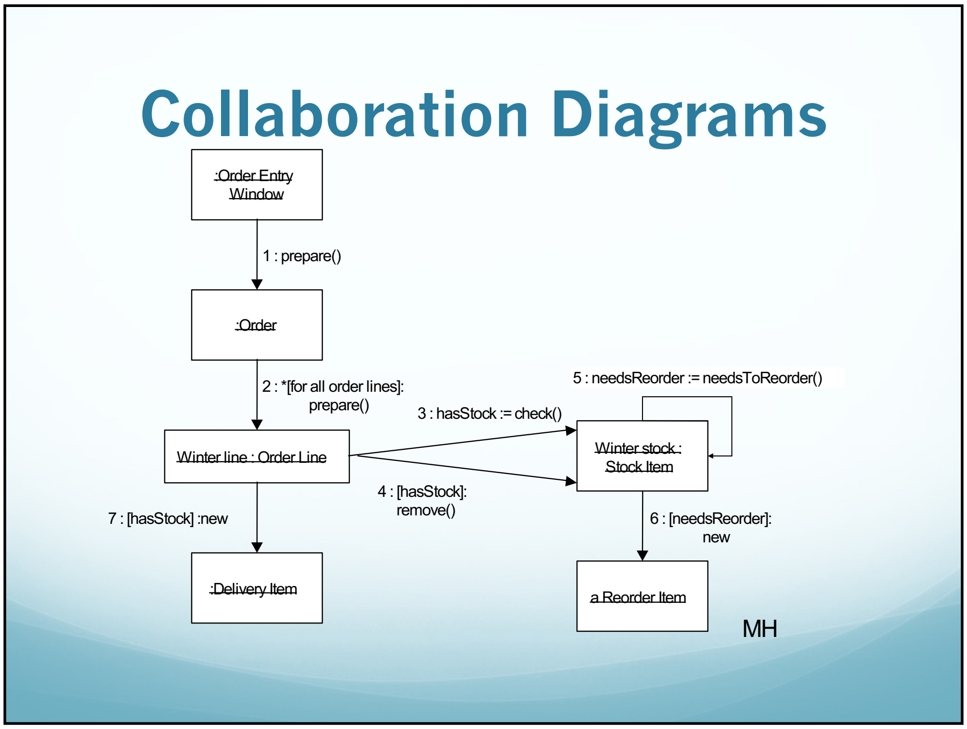
The controller creates a room object for each room in the building

The rooms sample the temperature in the toom every 5 s. When a low temp is detected the room notifies the controller.



Example from Fowler

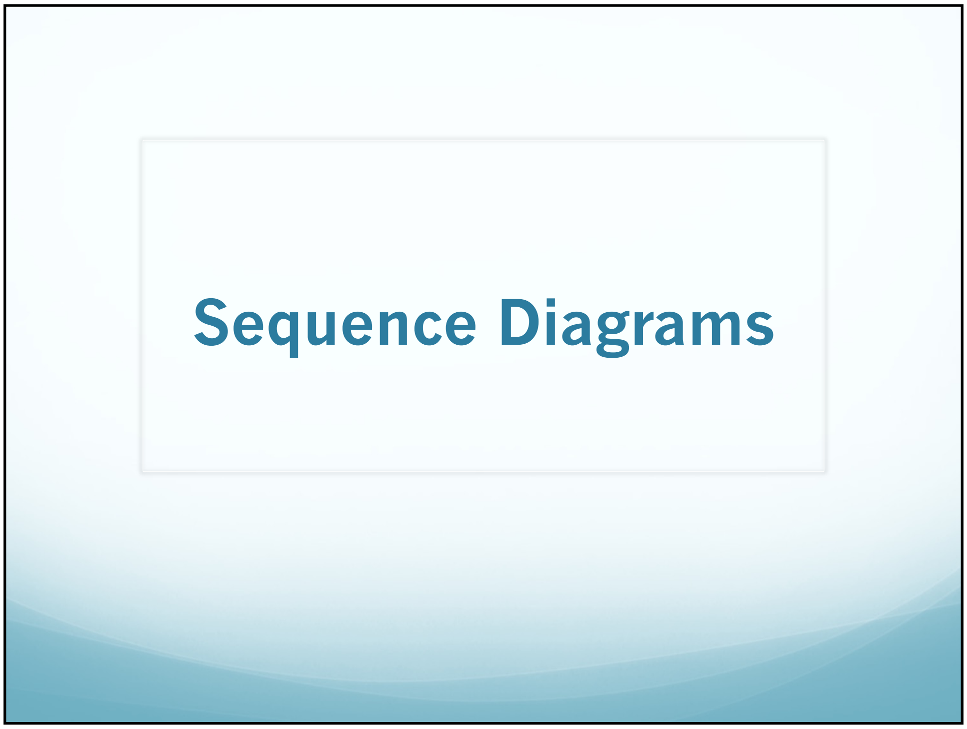




Collaboration Diagrams

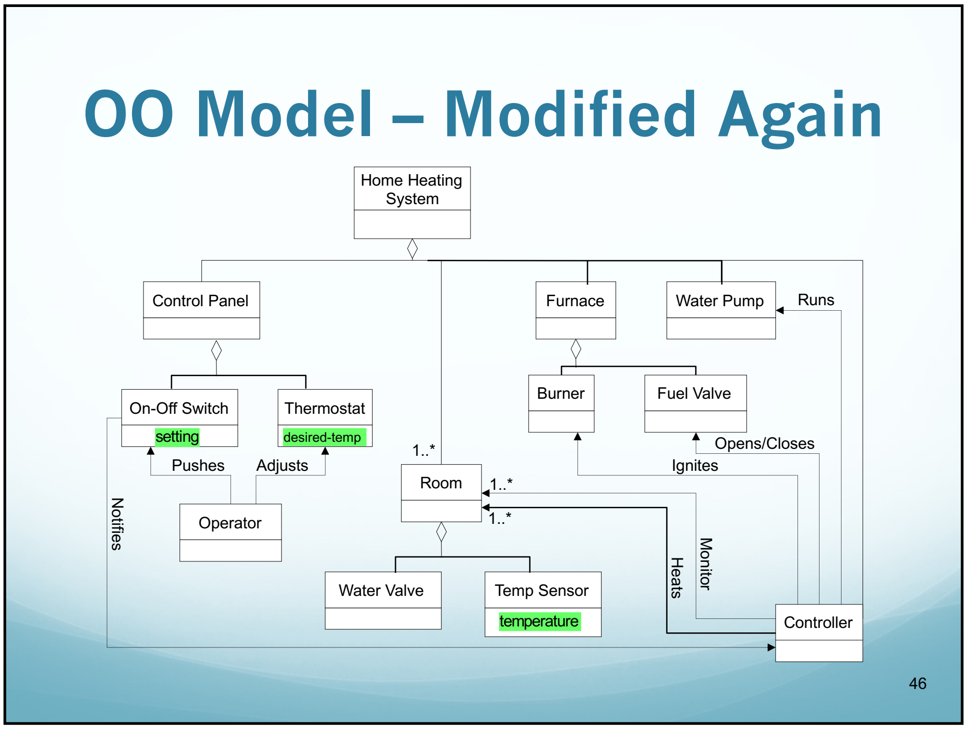
MH

44



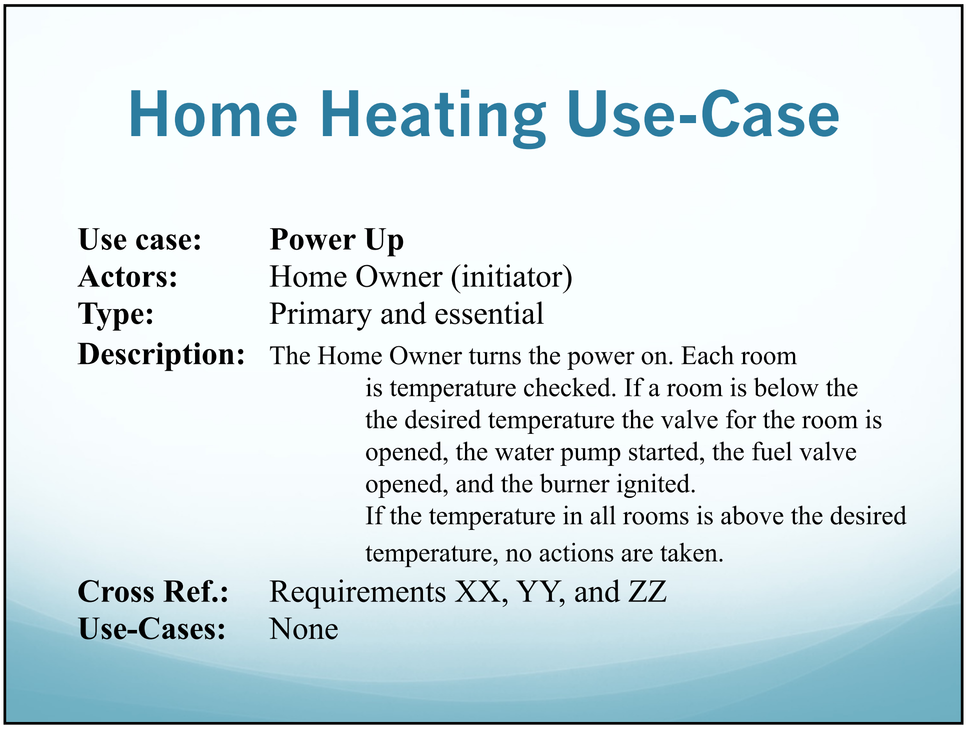
Sequence Diagrams

45



OO Model – Modified Again

46



Home Heating Use-Case

Use case: **Power Up**

Actors: Home Owner (initiator)

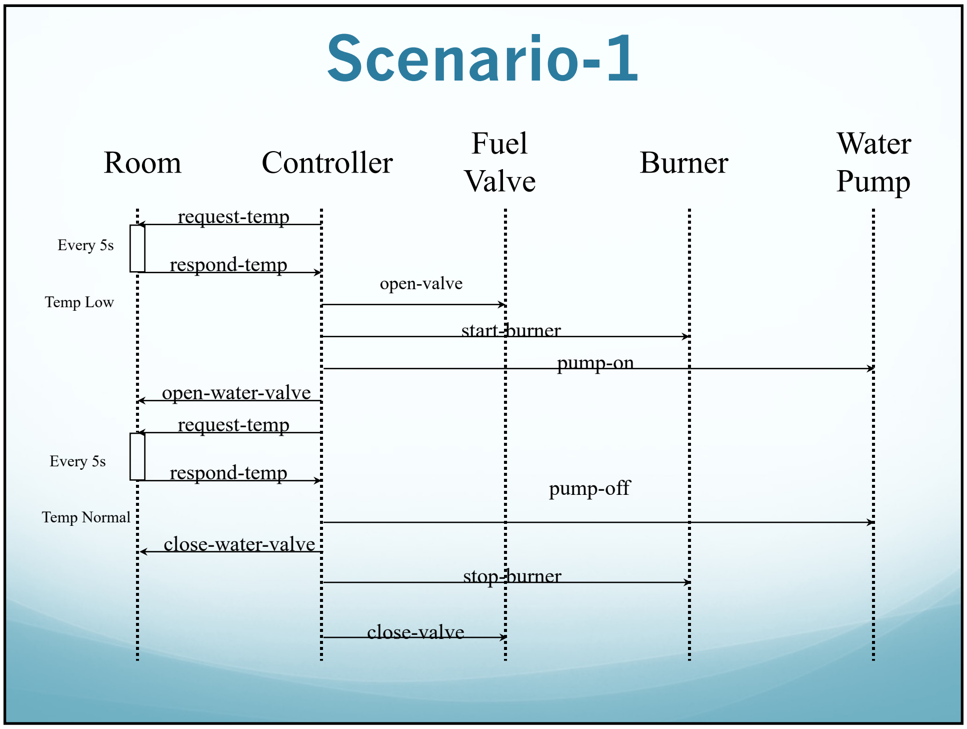
Type: Primary and essential

Description: The Home Owner turns the power on. Each room is temperature checked. If a room is below the 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

47



Scenario-1

Room

Controller

Fuel Valve

Burner

Water Pump

Every 5s

request-temp

respond-temp

Temp Low

open-valve

start-burner

pump-on

open-water-valve

request-temp

Every 5s

respond-temp

pump-off

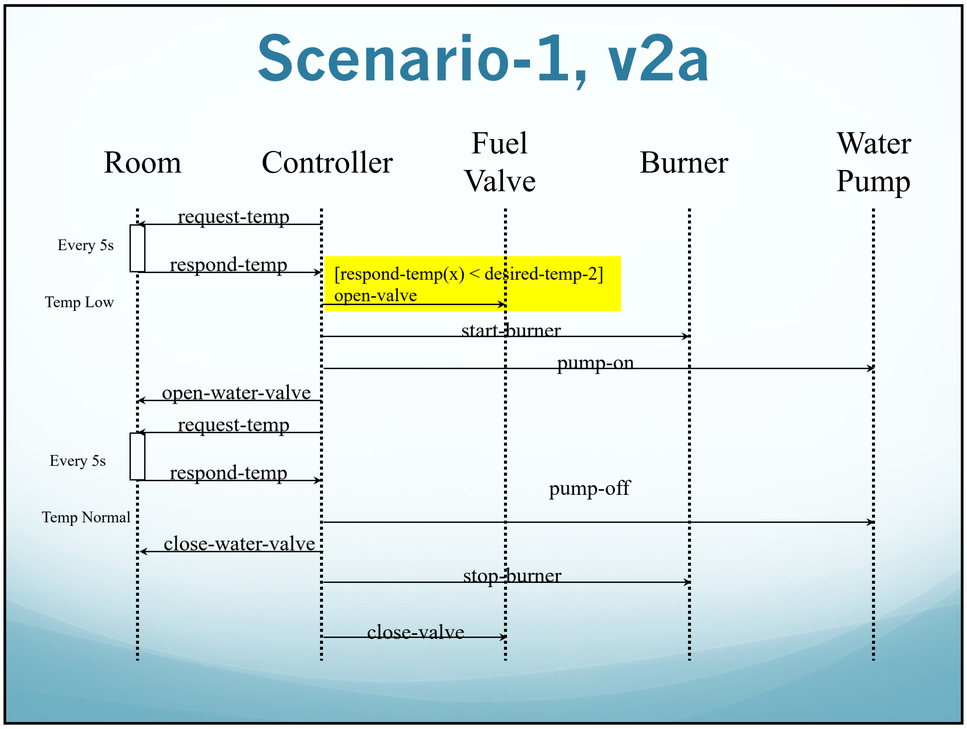
Temp Normal

close-water-valve

stop-burner

close-valve

48



Scenario-1, v2a

Room

Controller

Fuel Valve

Burner

Water Pump

Every 5s

request-temp

respond-temp

Temp Low

[respond-temp(x) < desired-temp-2]
open-valve

start-burner

pump-on

open-water-valve

request-temp

Every 5s

respond-temp

pump-off

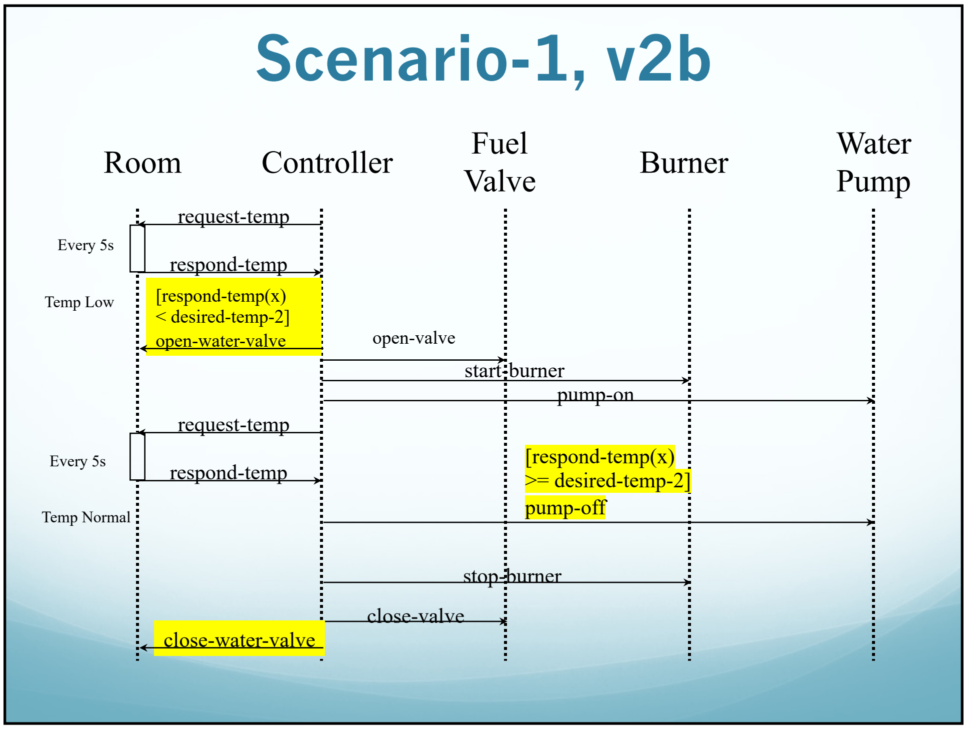
Temp Normal

close-water-valve

stop-burner

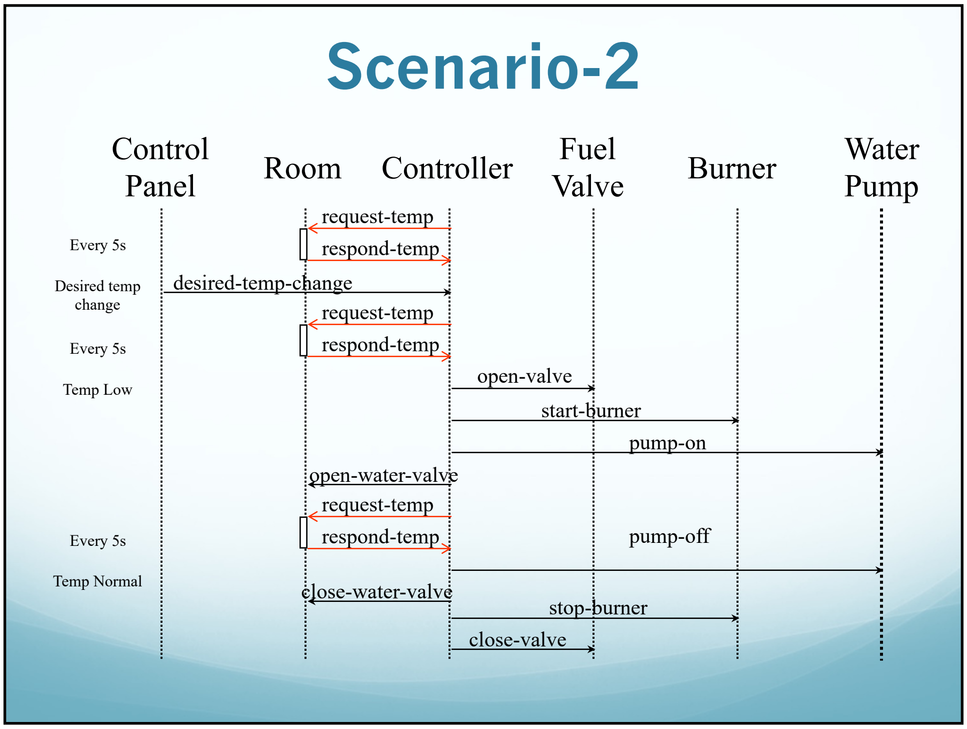
close-valve

49



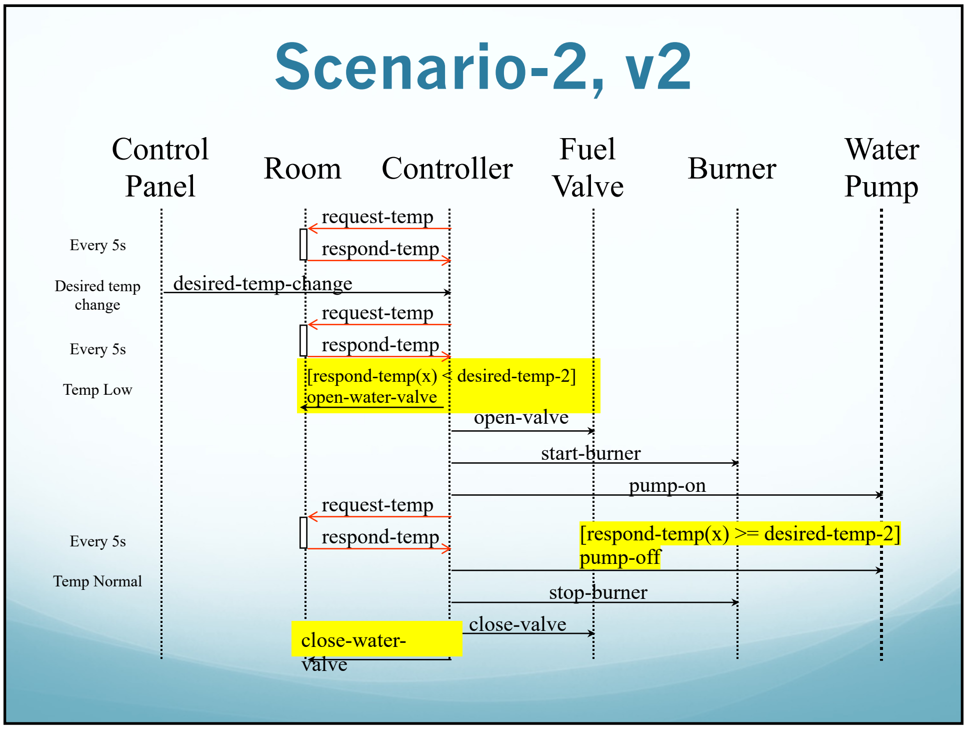
Scenario-1, v2b

50



Scenario-2

51



Scenario-2, v2

Control Panel

Room

Controller

Fuel Valve

Burner

Water Pump

Every 5s

request-temp

respond-temp

Desired temp change

desired-temp-change

request-temp

Every 5s

respond-temp

Temp Low

[respond-temp(x) < desired-temp-2]
open-water-valve

open-valve

start-burner

pump-on

request-temp

Every 5s

respond-temp

[respond-temp(x) >= desired-temp-2]
pump-off

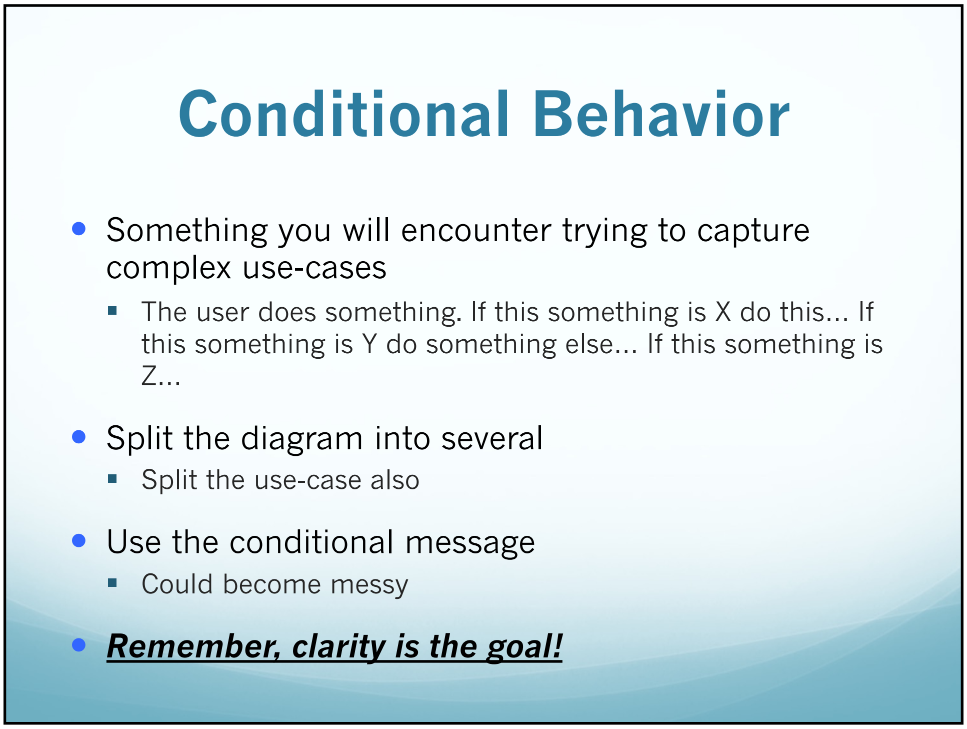
Temp Normal

stop-burner

close-valve

close-water-valve

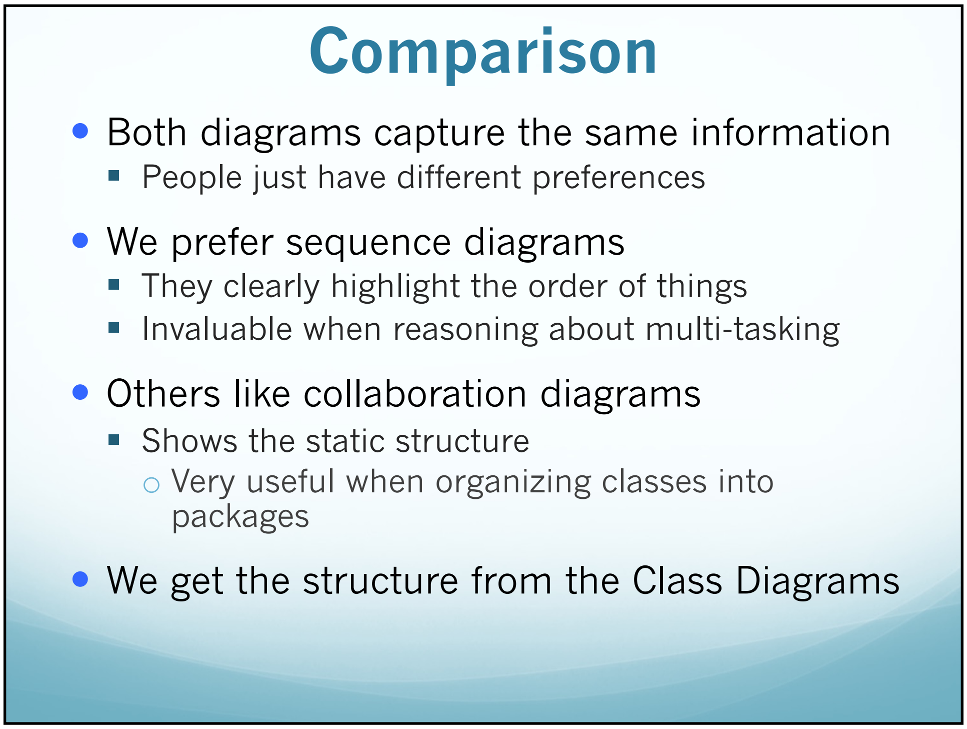
52



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!***

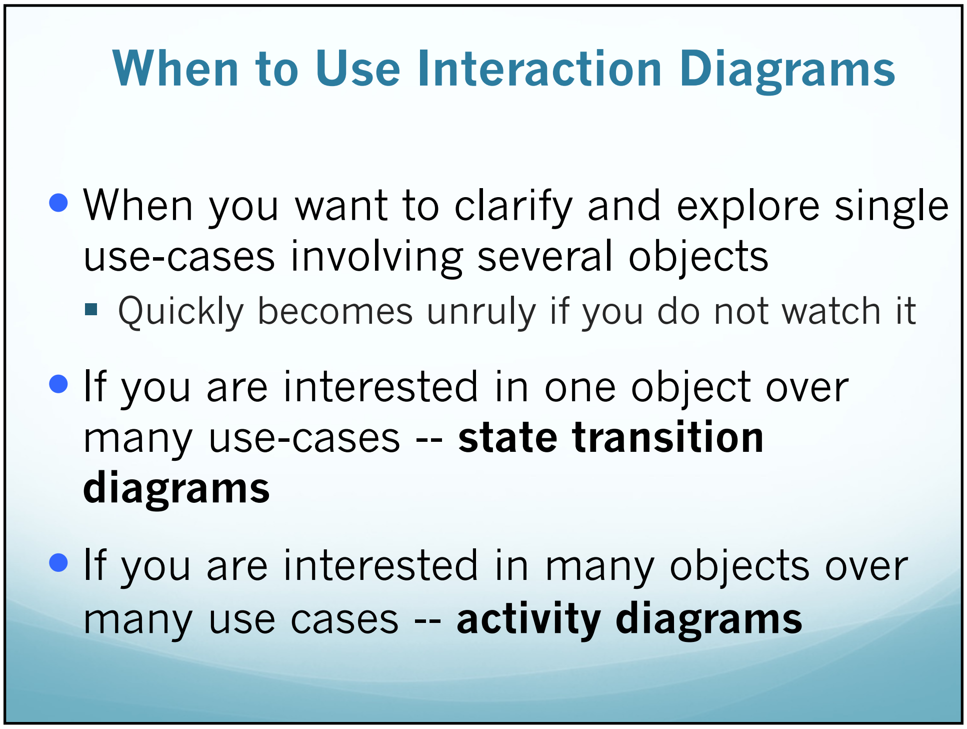
53



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

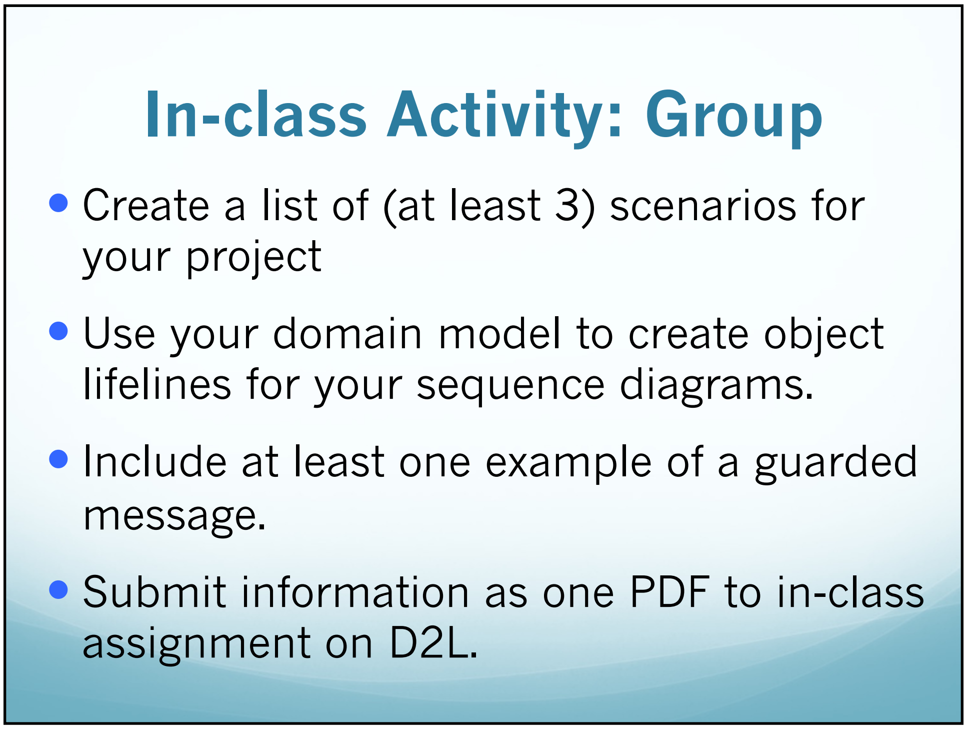
54



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**

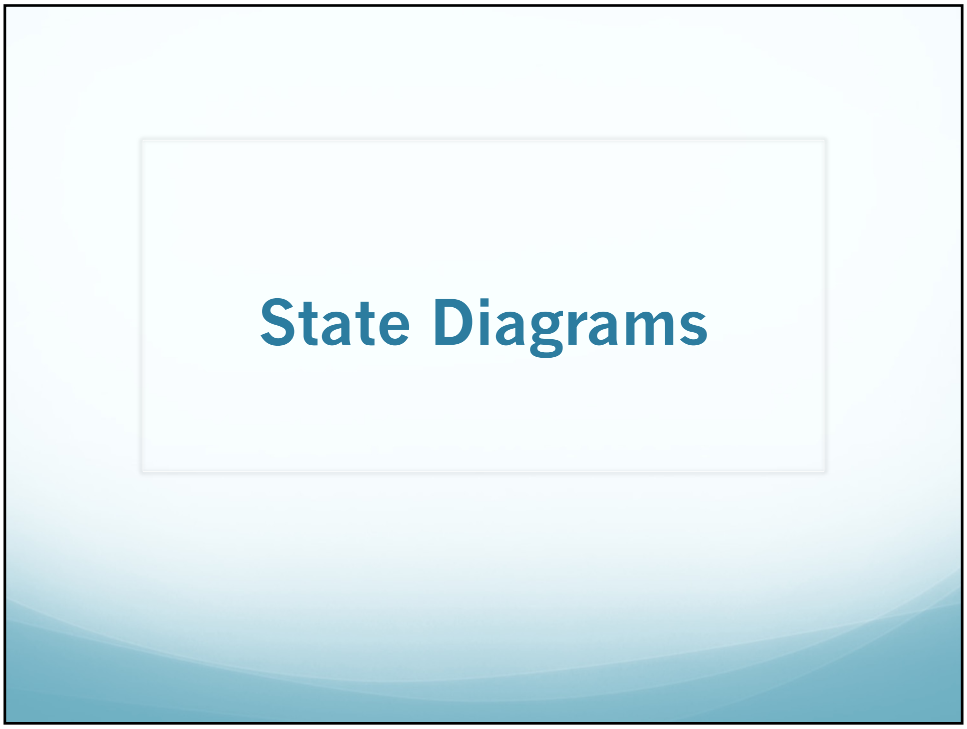
55



In-class Activity: Group

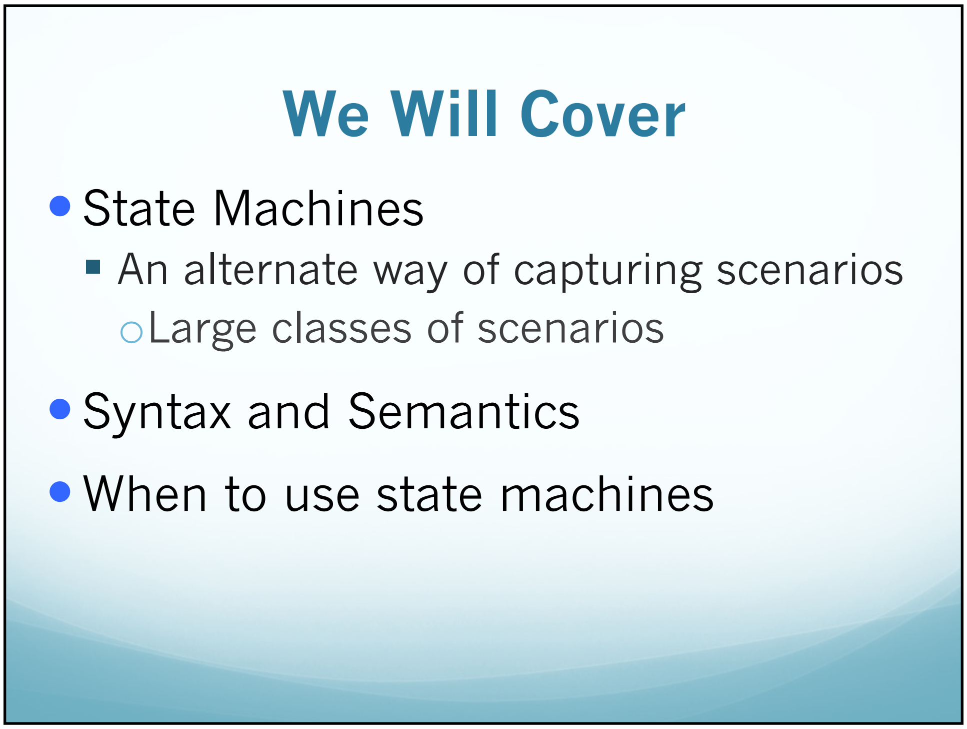
- Create a list of (at least 3) scenarios for your project
- Use your domain model to create object lifelines for your sequence diagrams.
- Include at least one example of a guarded message.
- Submit information as one PDF to in-class assignment on D2L.

57



State Diagrams

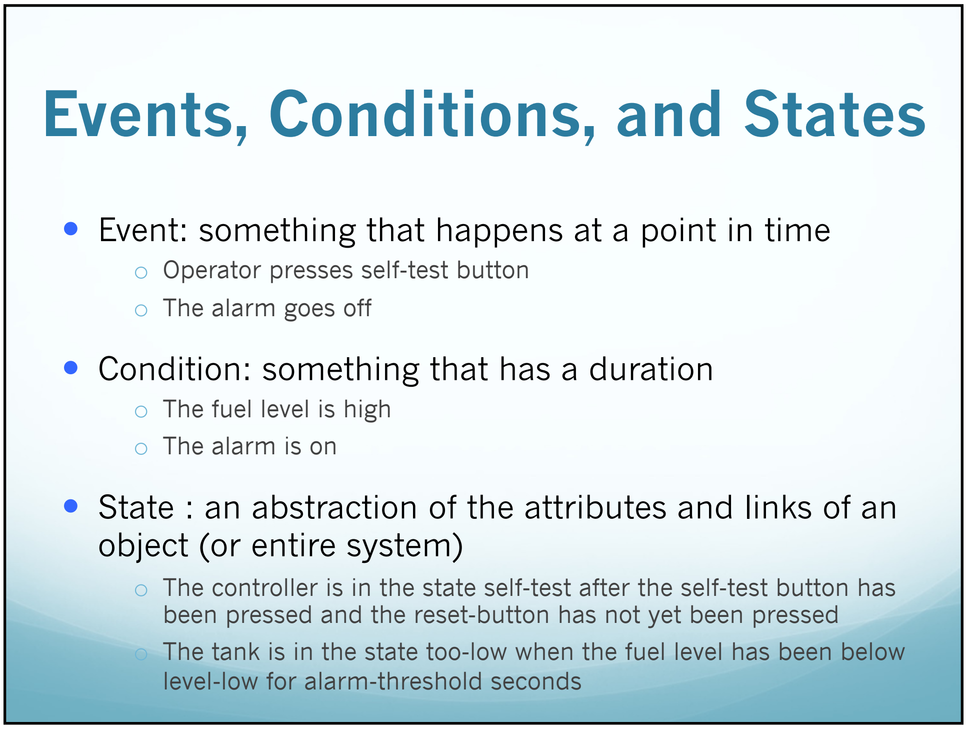
58



We Will Cover

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

59



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

60

Making a Phone Call Scenario

AT&T 5ESS Switch

Modern day phone calls?

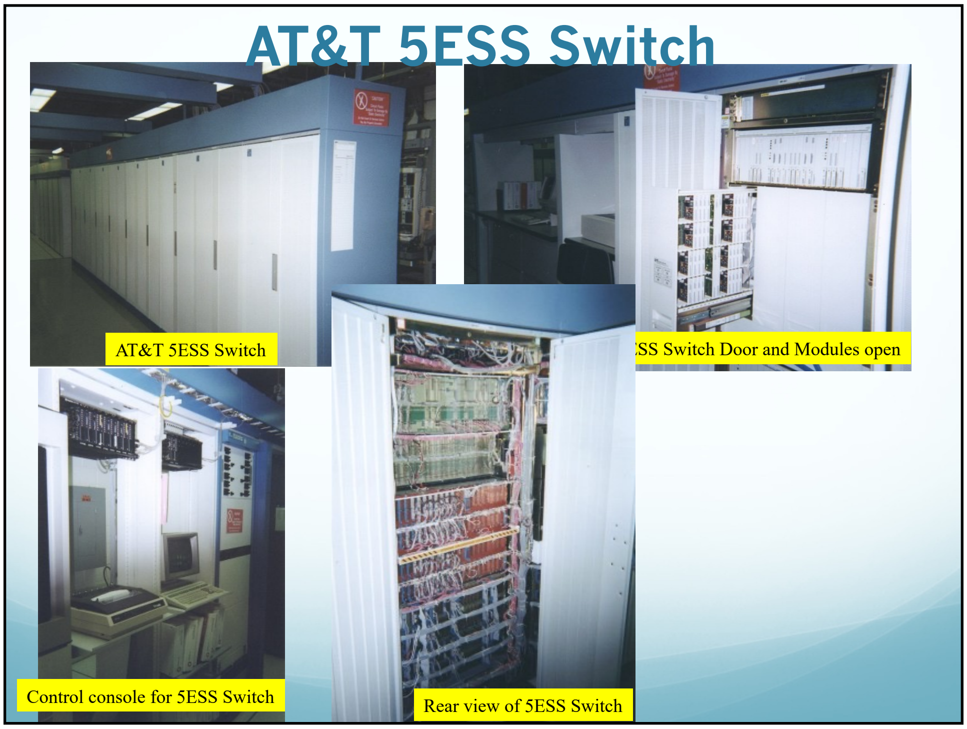
What is a landline phone?

Use your imagination…

PC: Just Jared
PC: telephoneworld.org
PC: Pats Pulpit

PC: Lost and Found Props

61



AT&T 5ESS Switch

AT&T 5ESS Switch

SS Switch Door and Modules open

Control console for 5ESS Switch

Rear view of 5ESS Switch

62

Let’s simplify things…

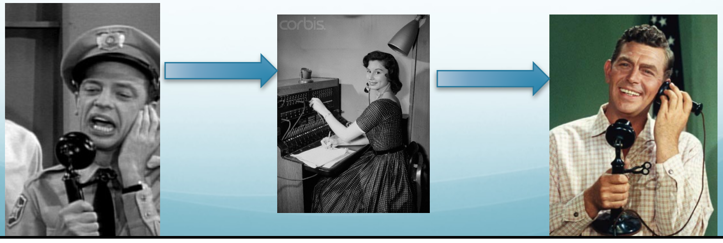
PC: Amazon.com

63

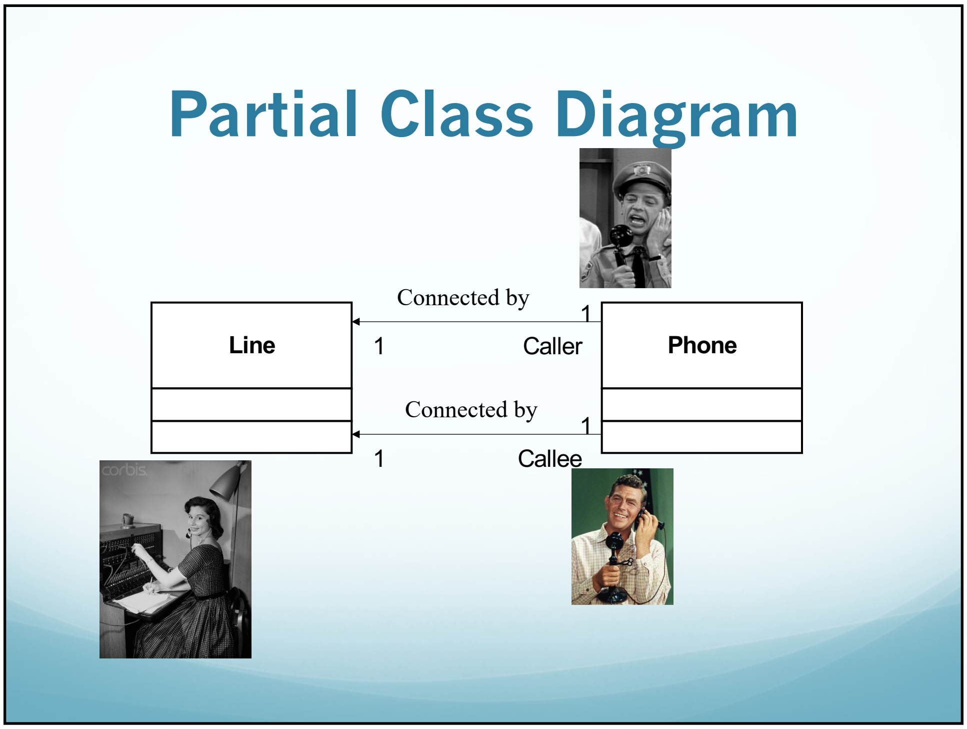
Making a Phone Call Scenario

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.

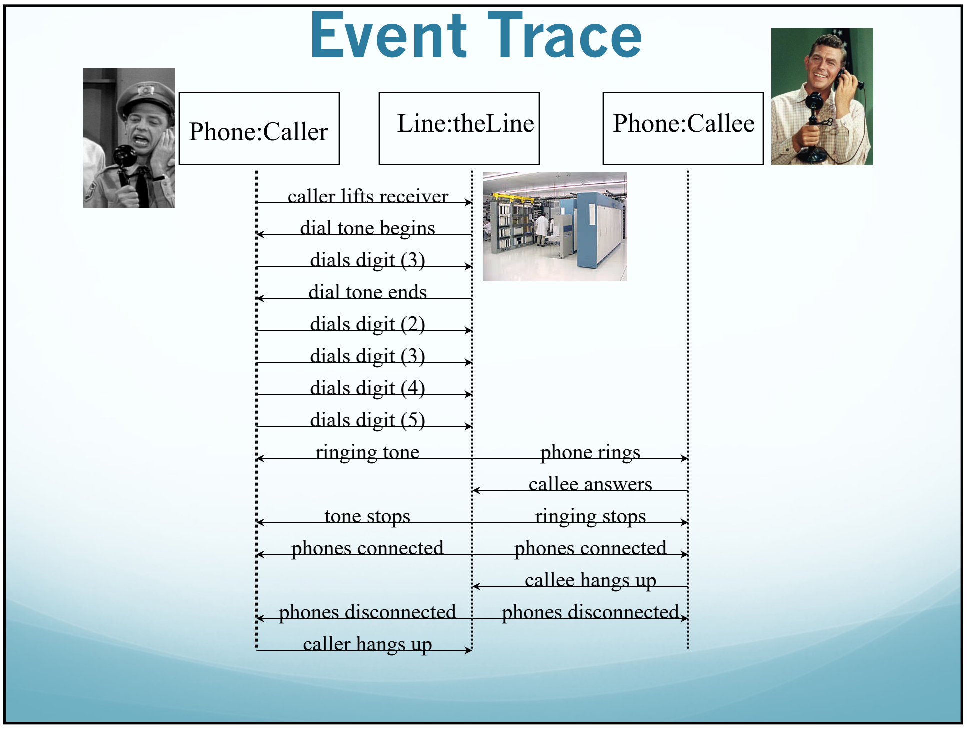


64



Partial Class Diagram

67



Event Trace

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

callee answers

tone stops — ringing stops

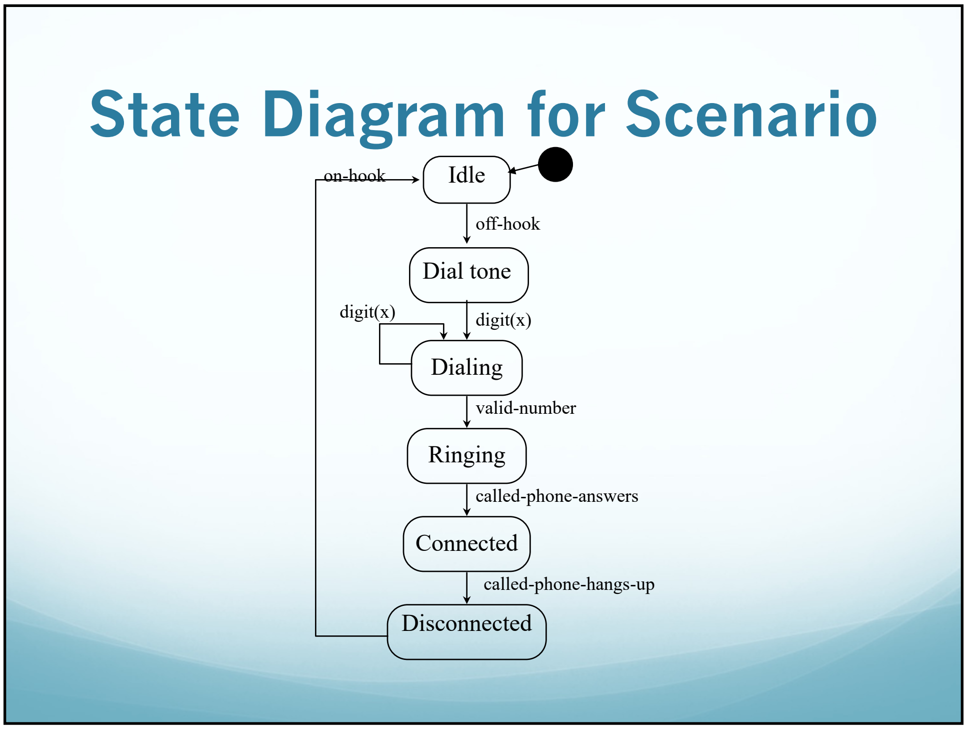
phones connected — phones connected

callee hangs up

phones disconnected — phones disconnected

caller hangs up

68



State Diagram for Scenario

Idle

off-hook

Dial tone

digit(x)

digit(x)

Dialing

valid-number

Ringing

called-phone-answers

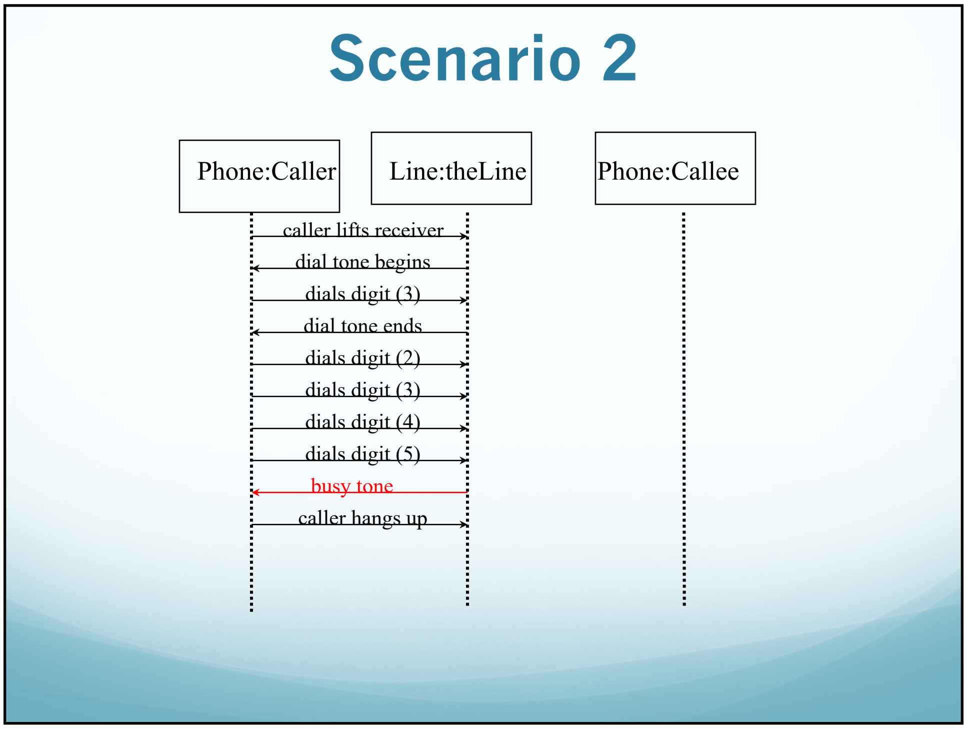
Connected

called-phone-hangs-up

Disconnected

on-hook

69



Scenario 2

Phone:Caller

Line:theLine

Phone:Callee

caller lifts receiver

dial tone begins

dials digit (3)

dial tone ends

dials digit (2)

dials digit (3)

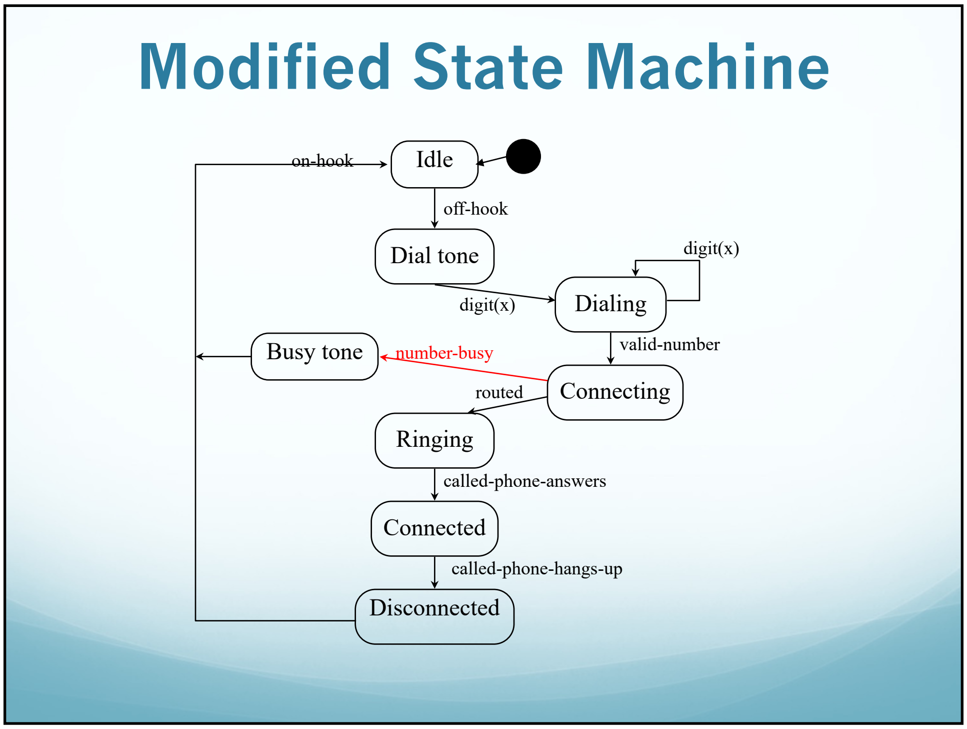
dials digit (4)

dials digit (5)

busy tone

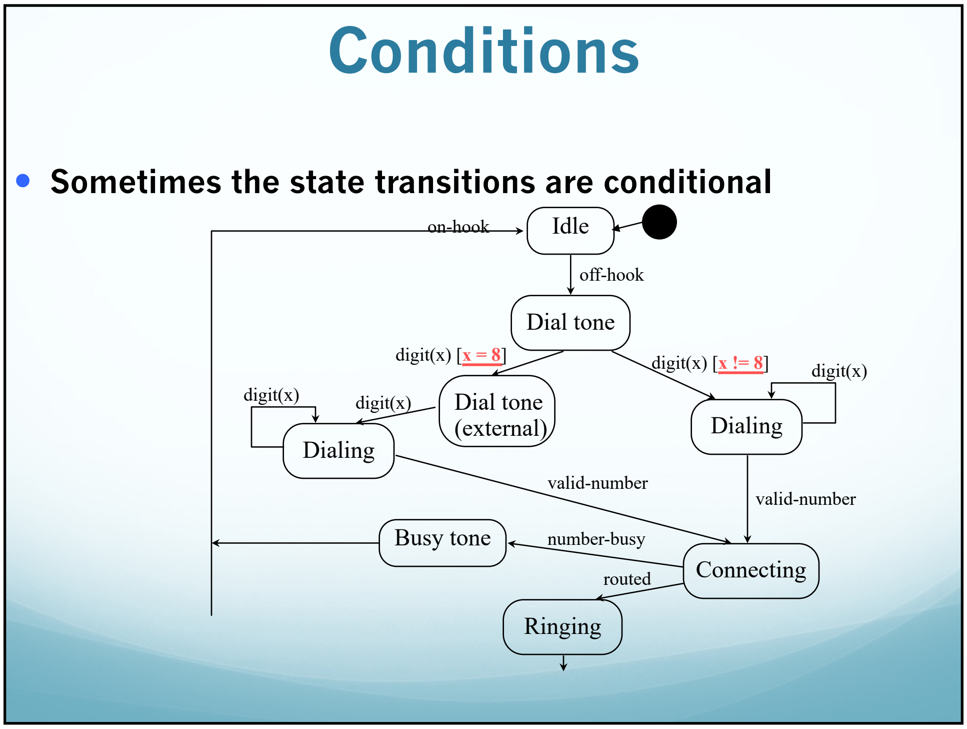
caller hangs up

70



Modified State Machine

71



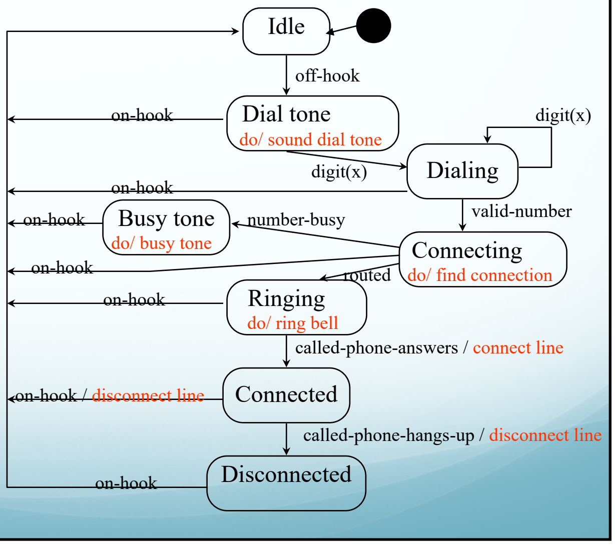
Conditions

- Sometimes the state transitions are conditional

72

Operations (AKA Actions)

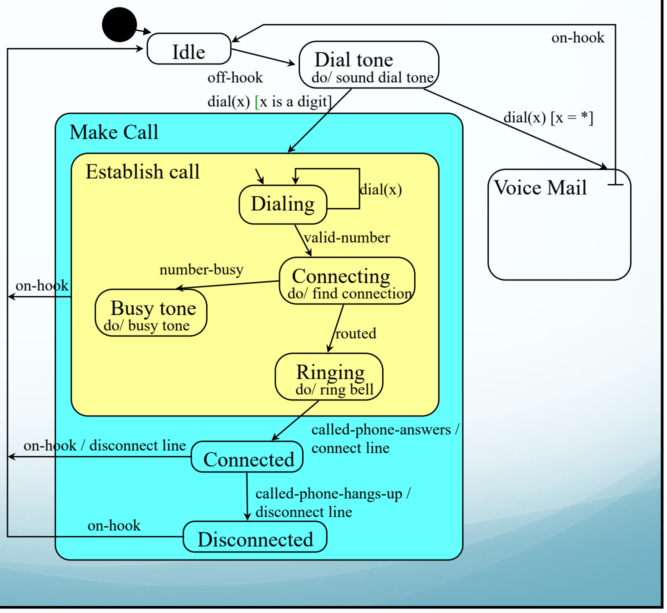
- Actions are performed when a transition is taken or performed while in a state
- Actions are terminated when leaving the state



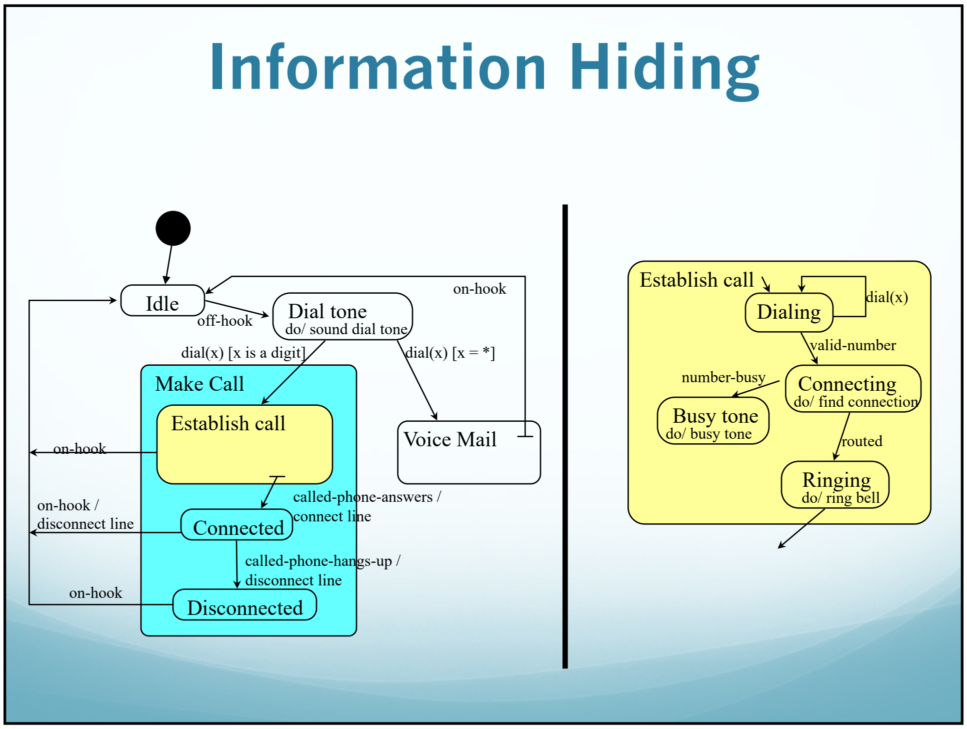
73

Hierarchical State Machines

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

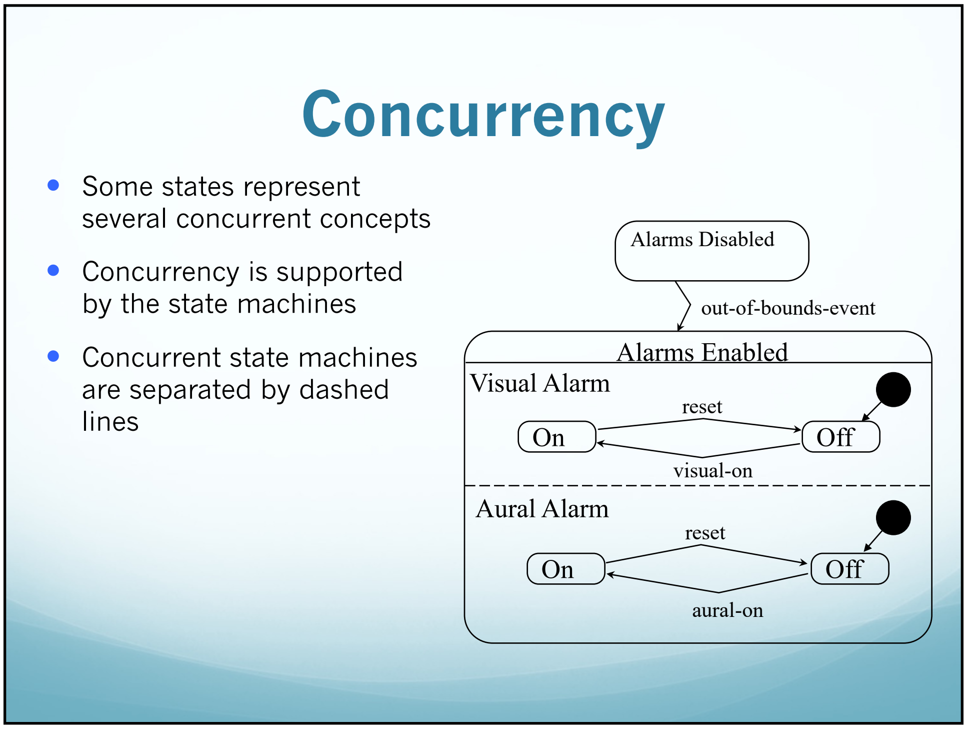


74



Information Hiding

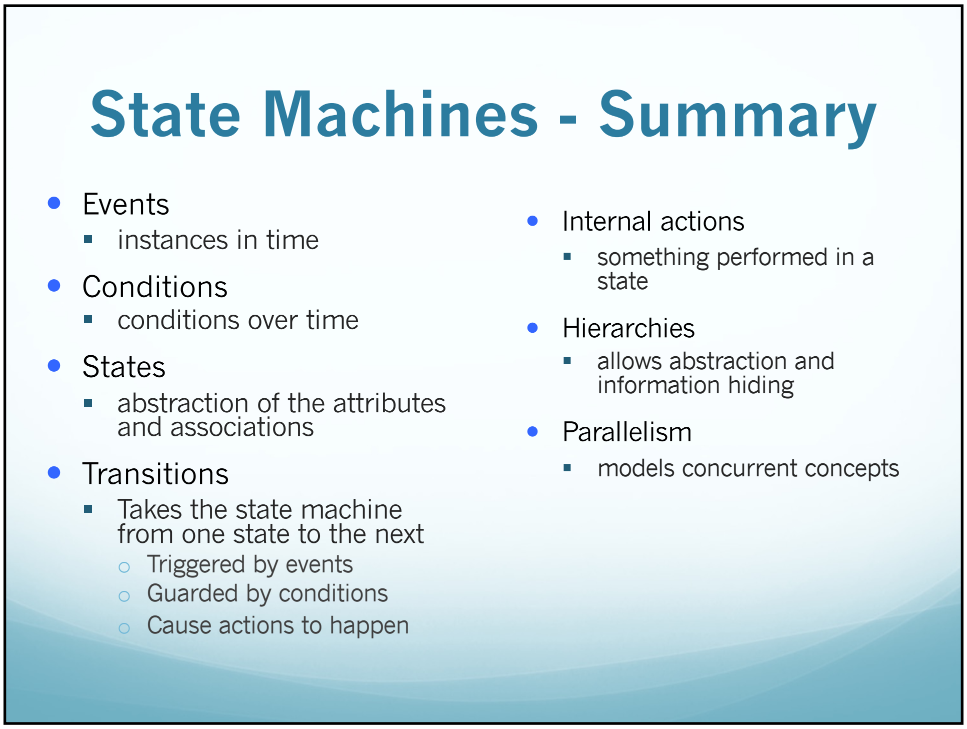
75



Concurrency

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

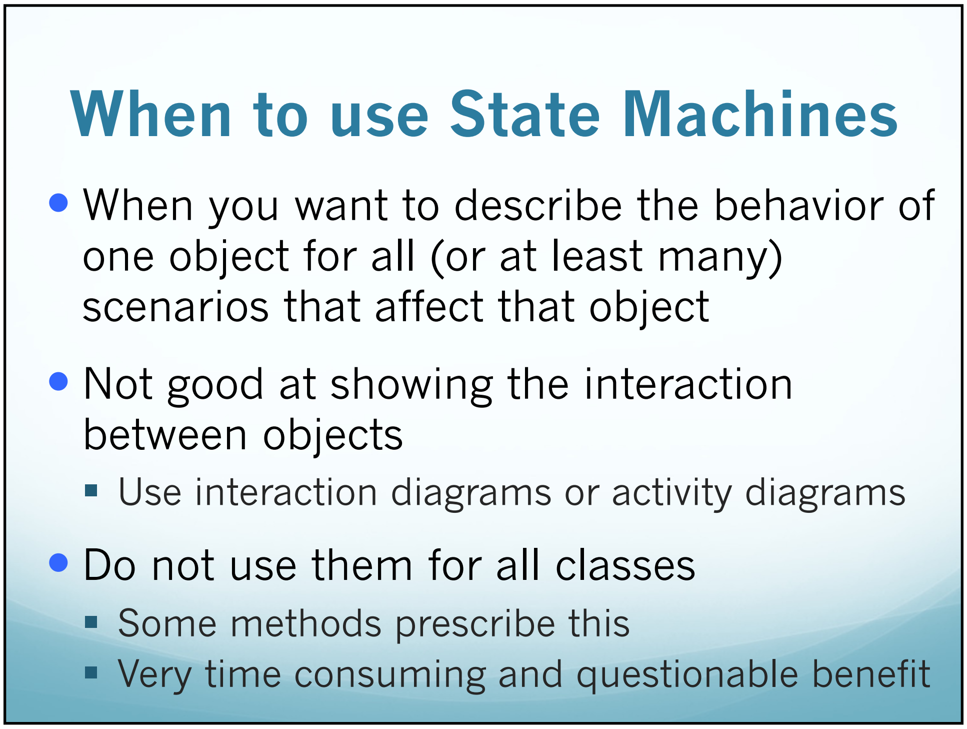
77



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

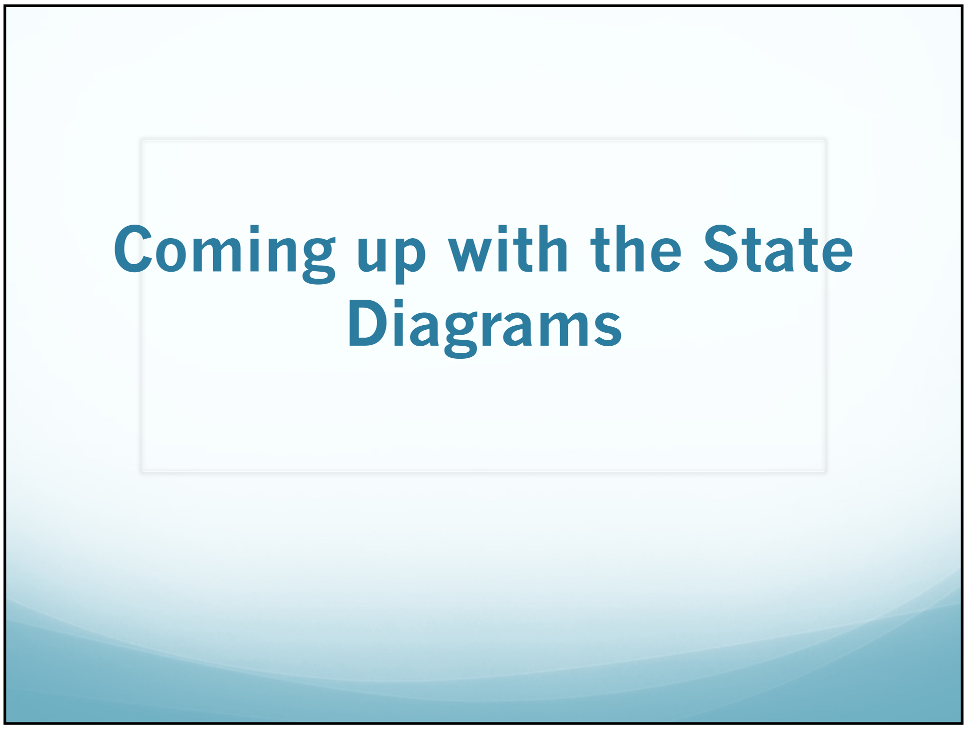
78



When to use State Machines

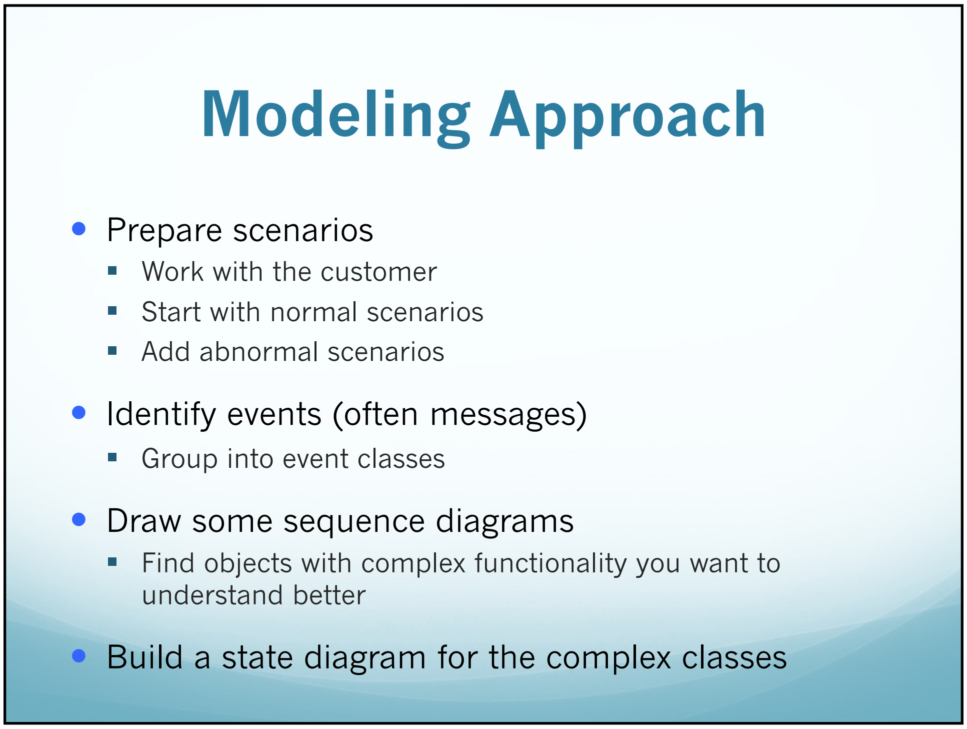
- When you want to describe the behavior of one object for all (or at least many) scenarios that affect that object
- Not good at showing the interaction between objects
  - Use interaction diagrams or activity diagrams
- Do not use them for all classes
  - Some methods prescribe this
  - Very time consuming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

79



Coming up with the State Diagrams

83

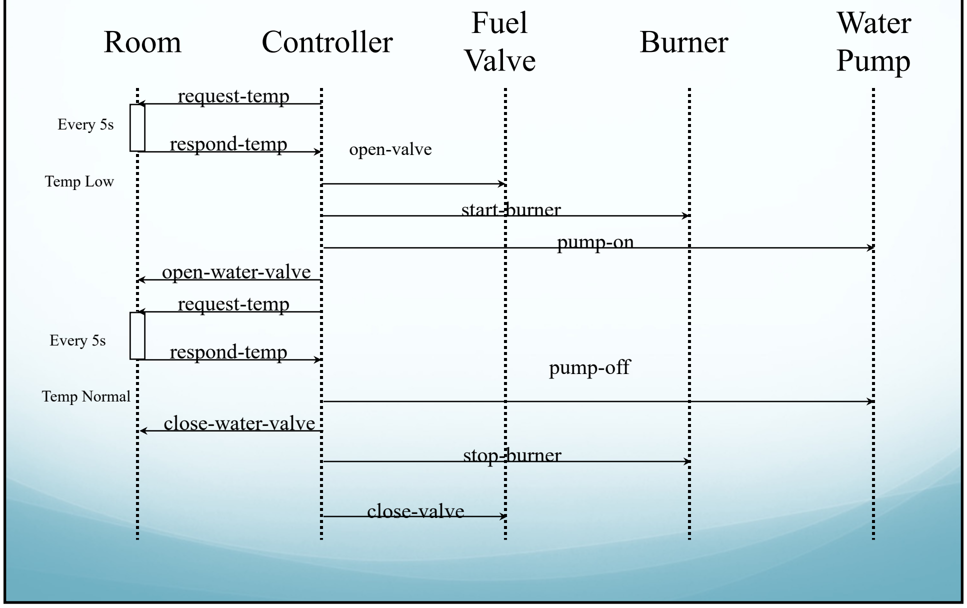


Modeling Approach

- 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

84

Scenario-1



Room

Controller

Fuel Valve

Burner

Water Pump

request-temp

Every 5s

respond-temp

open-valve

Temp Low

start-burner

pump-on

open-water-valve

request-temp

Every 5s

respond-temp

pump-off

Temp Normal

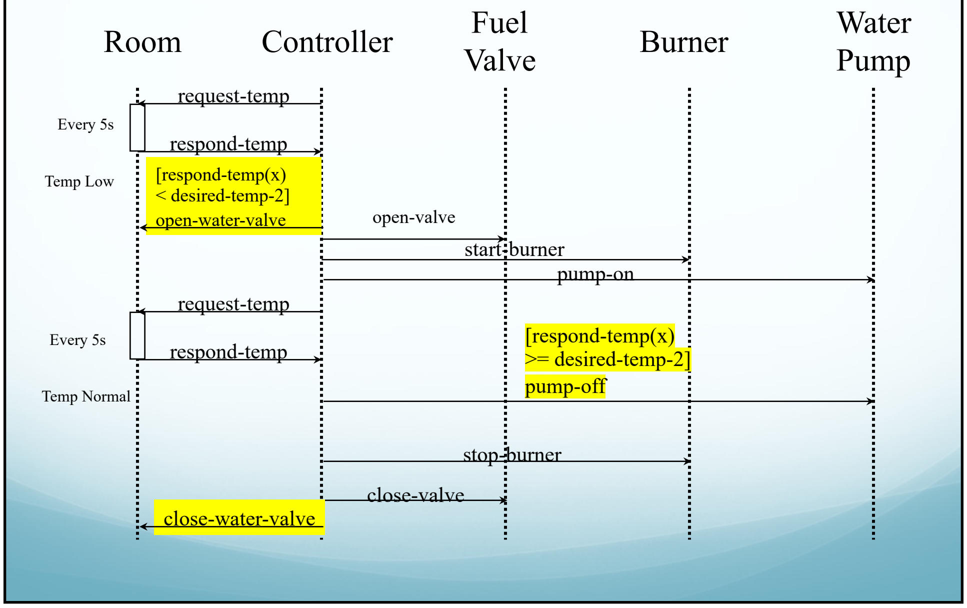
close-water-valve

stop-burner

close-valve

85

Scenario-1, v2b



Room

Controller

Fuel Valve

Burner

Water Pump

request-temp

Every 5s

respond-temp

Temp Low

[respond-temp(x)
< desired-temp-2]
open-water-valve

open-valve

start-burner

pump-on

request-temp

Every 5s

respond-temp

[respond-temp(x)
>= desired-temp-2]
pump-off

Temp Normal

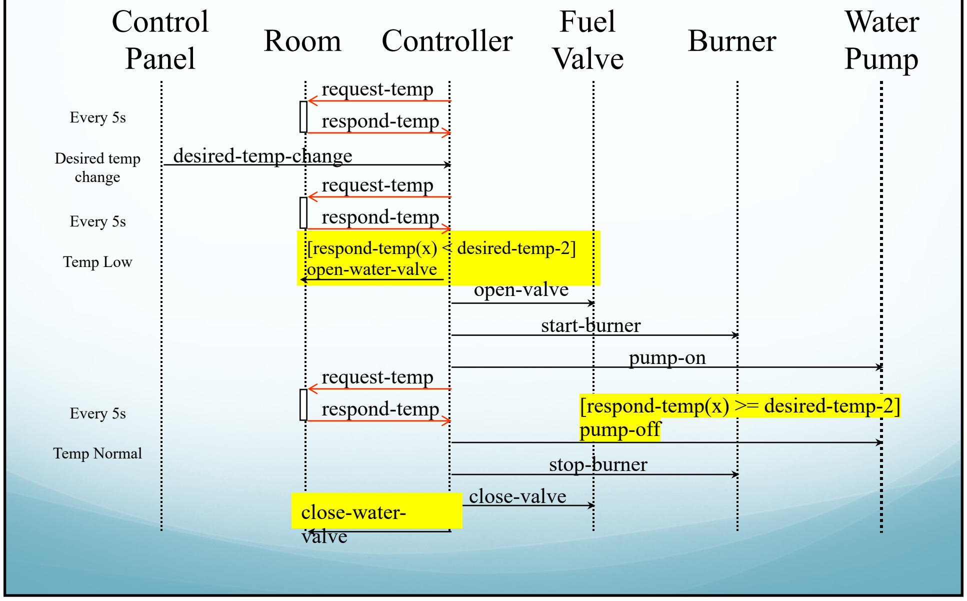
stop-burner

close-valve

close-water-valve

87

Scenario-2, v2



Control Panel

Room

Controller

Fuel Valve

Burner

Water Pump

Every 5s

request-temp

respond-temp

Desired temp change

desired-temp-change

Every 5s

request-temp

respond-temp

Temp Low

[respond-temp(x) < desired-temp-2]
open-water-valve

open-valve

start-burner

pump-on

Every 5s

request-temp

respond-temp

[respond-temp(x) >= desired-temp-2]
pump-off

Temp Normal

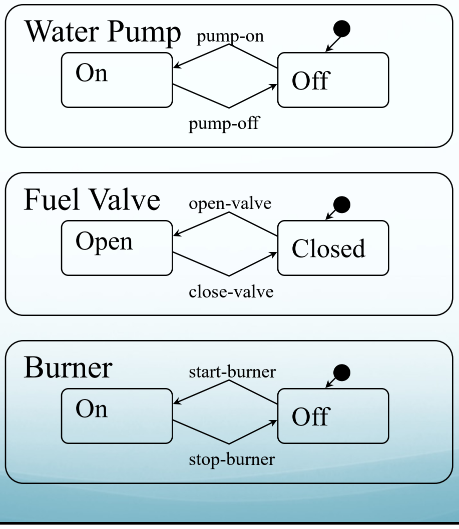
stop-burner

close-valve

close-water-valve

89

Dynamic Model



Water Pump

pump-on

On

Off

pump-off

Fuel Valve

open-valve

Open

Closed

close-valve

Burner

start-burner

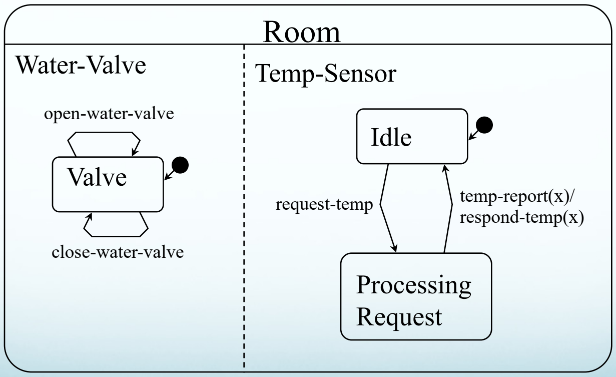
On

Off

stop-burner

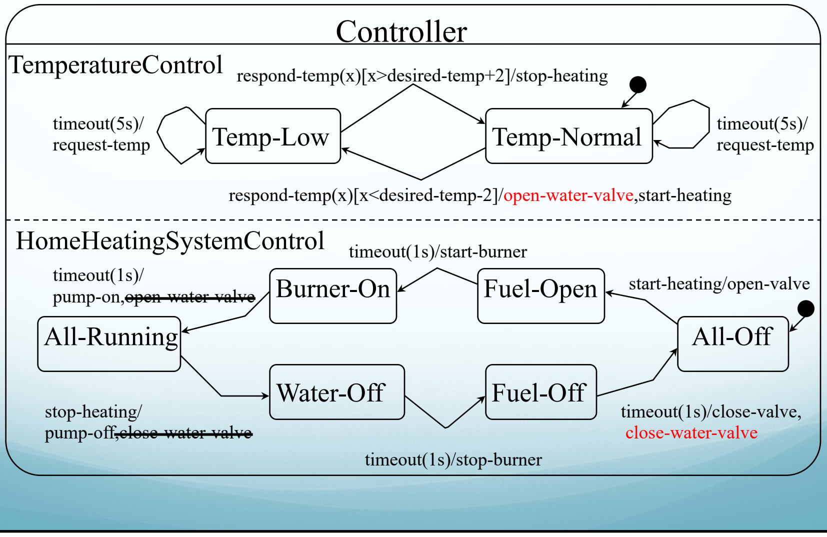
91

More Dynamic Model



92

Even More Dynamic Model, v2



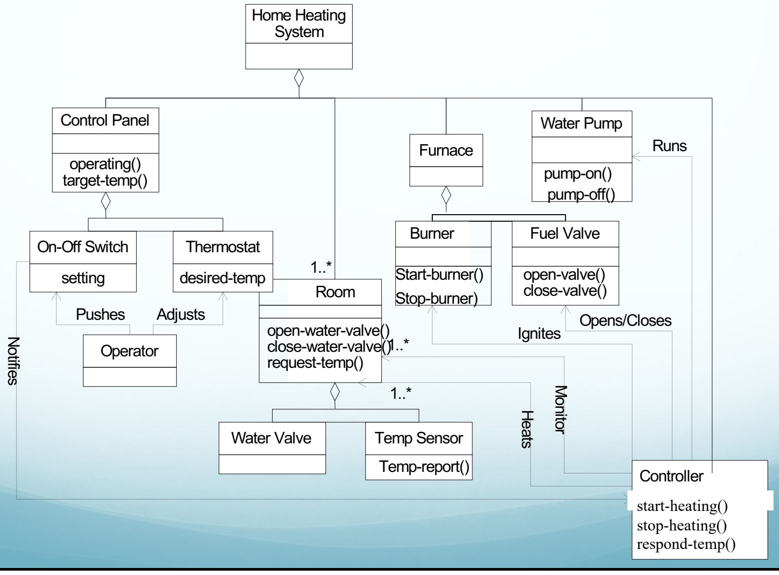
95

Identify Key Operations

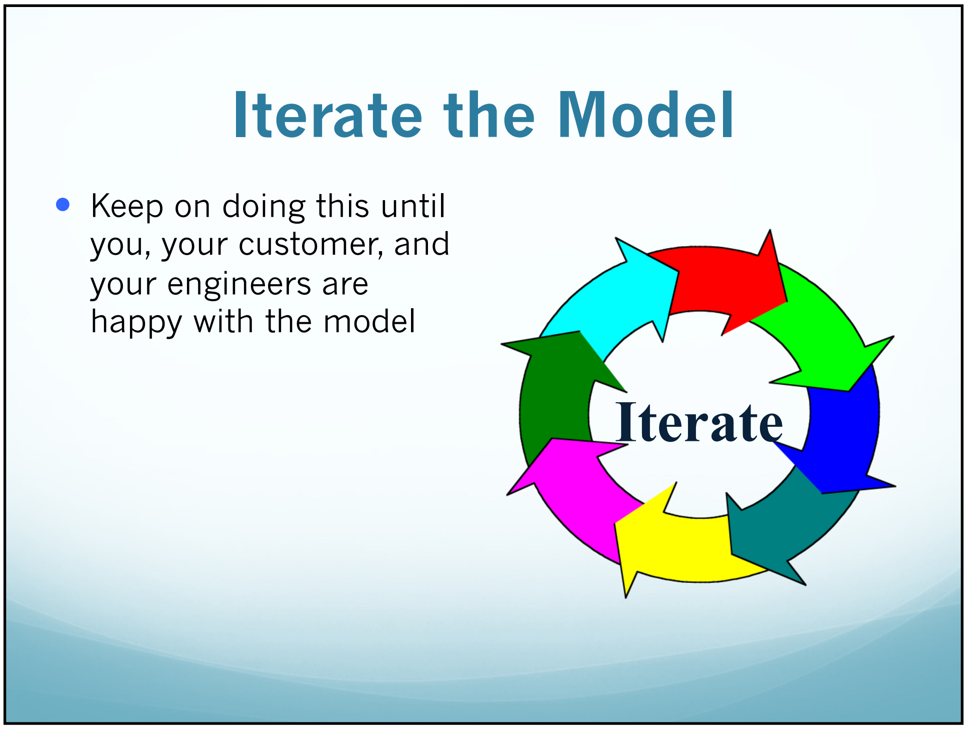
- Operations from the object model
  - Accessing and setting attributes and associations (often not shown)
- Operations from events
  - All events represent some operation
- Operations from actions and activities
  - Actions and activities represent some processing activity within some object
- Operations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

96

Complete OO Model



97



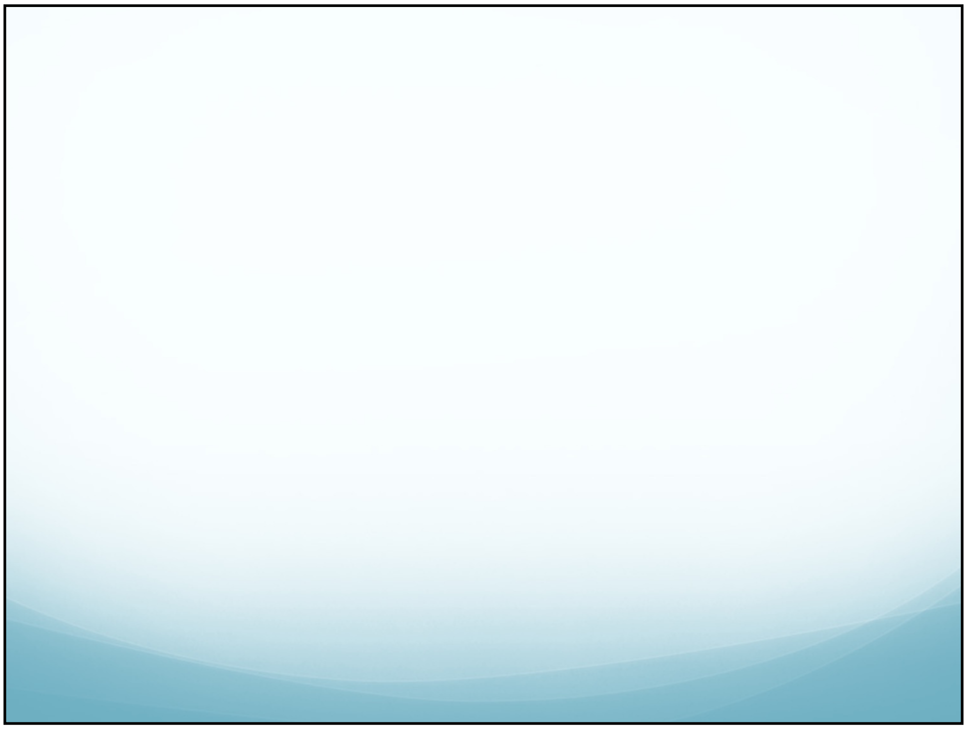
Iterate the Model

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

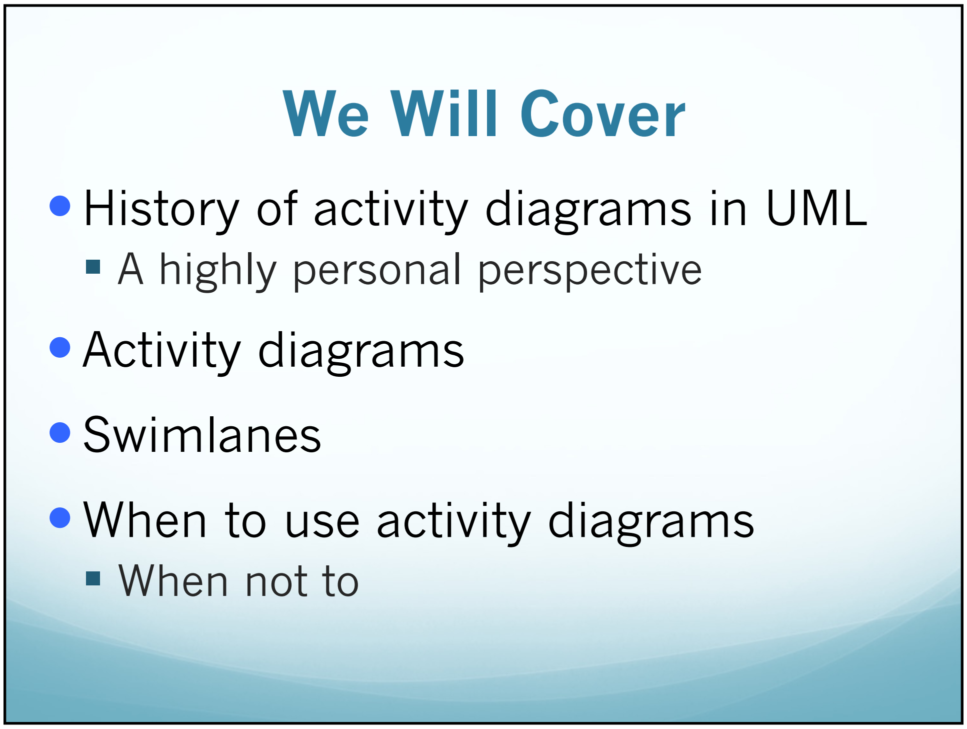
99

Activity Diagrams

101



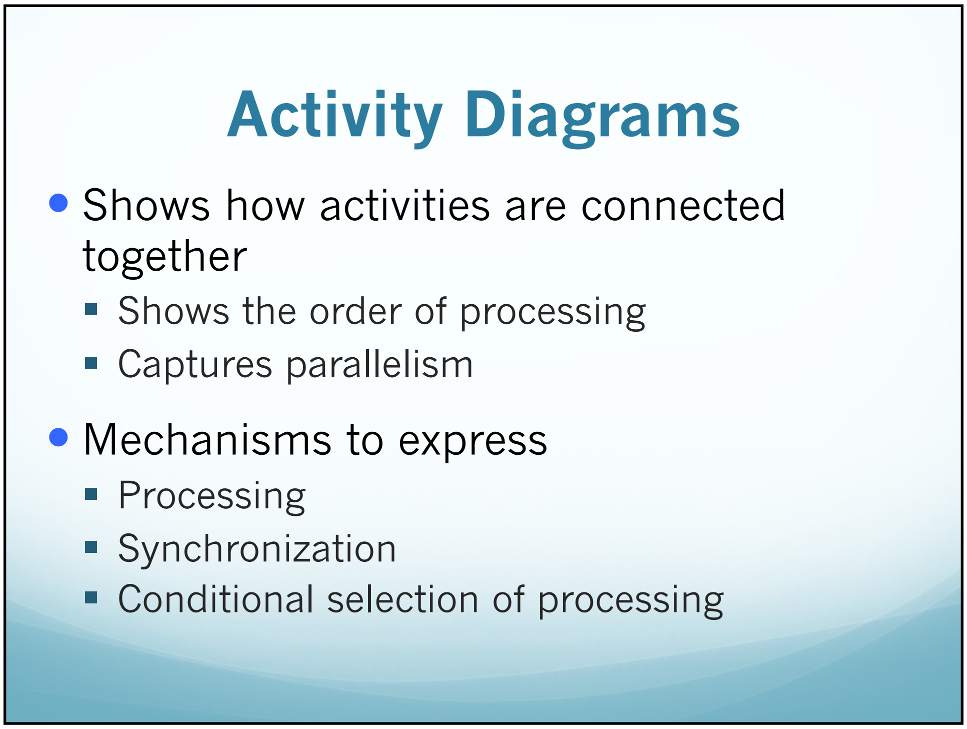
102



We Will Cover

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

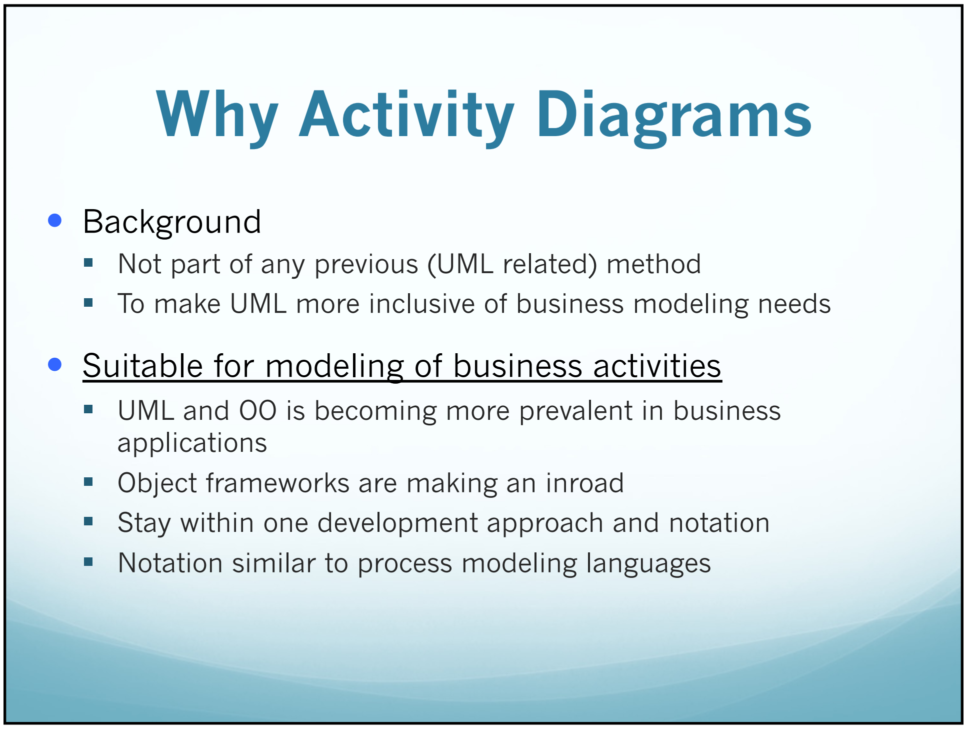
103



Activity Diagrams

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

104

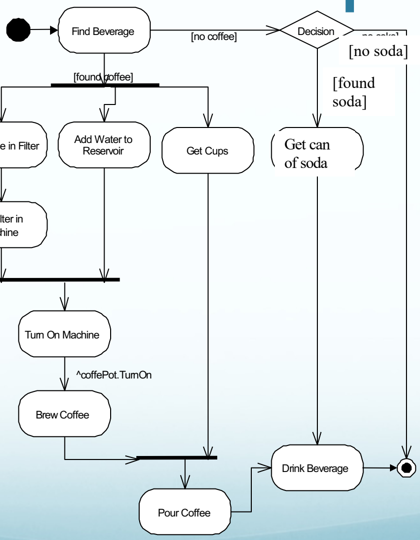


Why Activity Diagrams

- Background
  - 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

105

Coffee Example



MH

106

HACS Use-Cases

Use case: **Distribute Assignments**

Actors: Instructor (initiator), Student

Type: Primary and essential

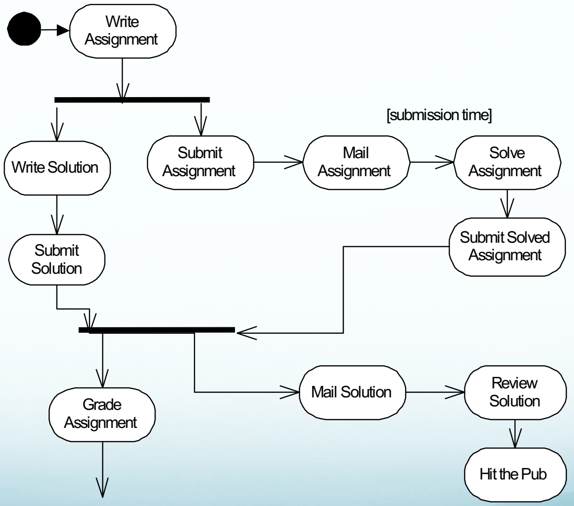
Description: The Instructor develops an assignment and submits it to the system. The instructor will also submit the delivery date, due date, and the class for which the assignment is assigned. Once the student has submitted the solution on the due date, the system will mail the solution to the student.

Cross Ref.: Requirements XX, YY, and ZZ

Use-Cases: *Configure HACS* must be done before any user (Instructor or Student) can use HACS

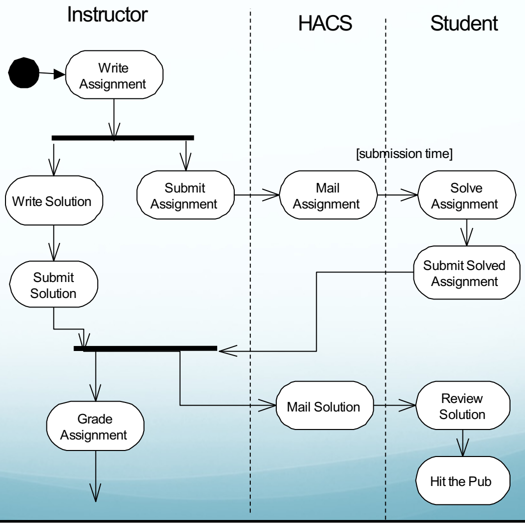
107

Activity Diagrams for Use Cases

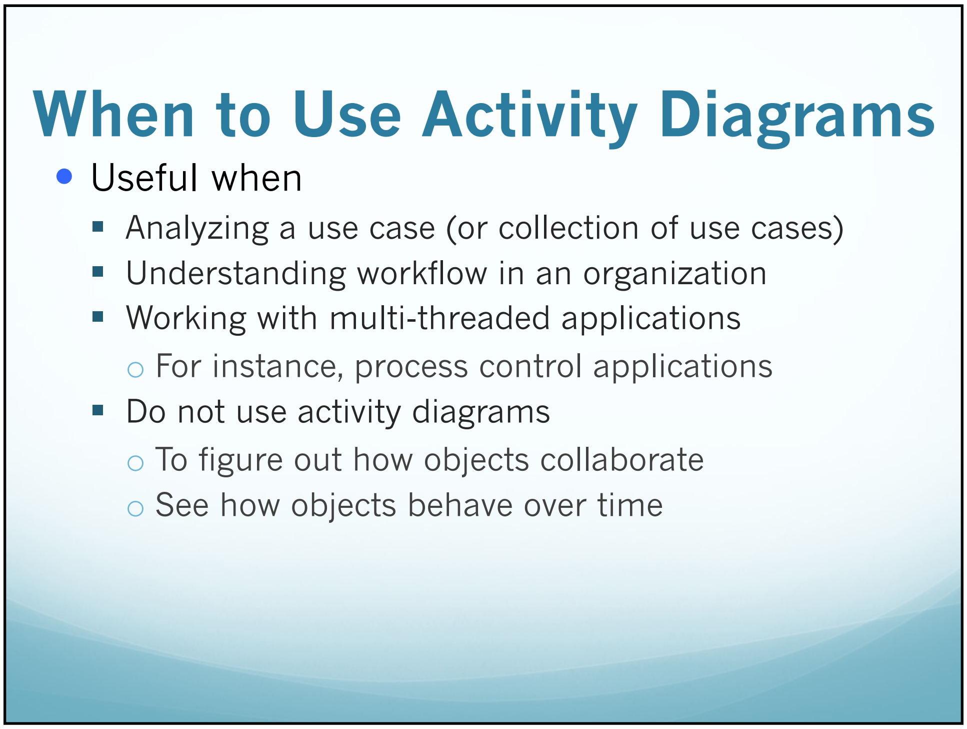


108

Swimlanes (Who Does What?)



109



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

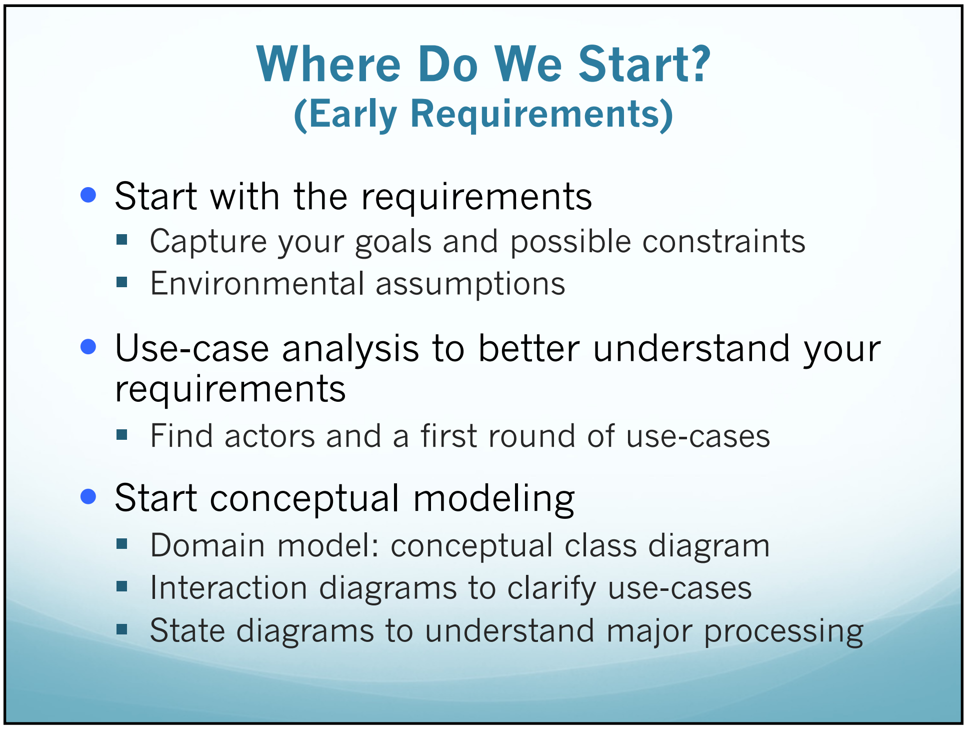
111

Approaching a Problem

Where do we start?

How do we proceed?

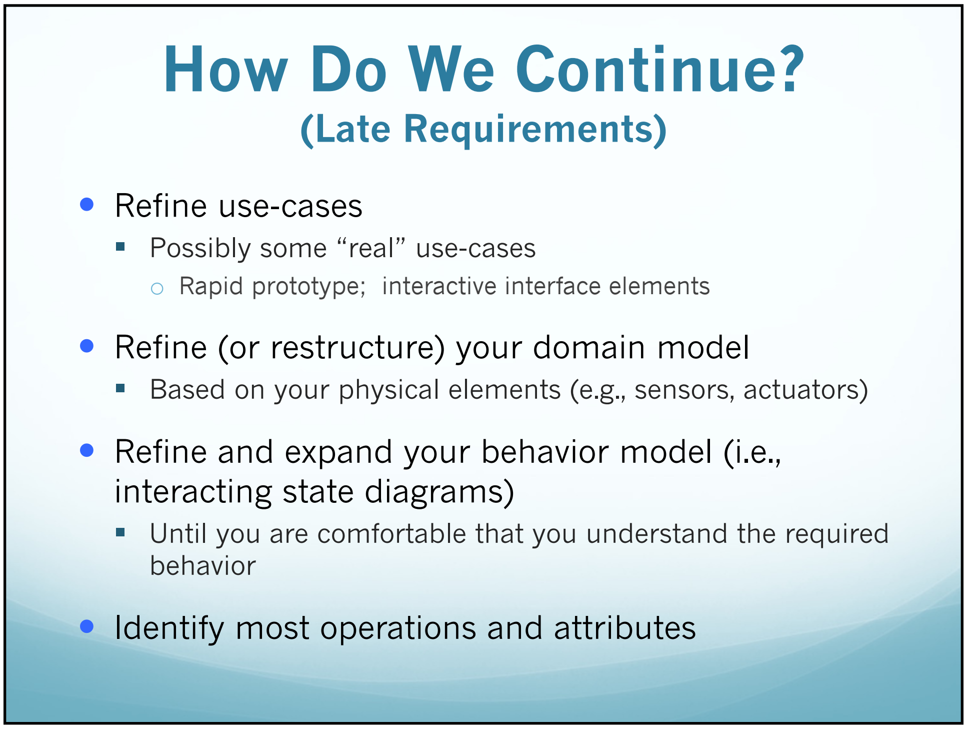
112



Where Do We Start?
(Early Requirements)

- 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
  - Domain model: conceptual class diagram
  - Interaction diagrams to clarify use-cases
  - State diagrams to understand major processing

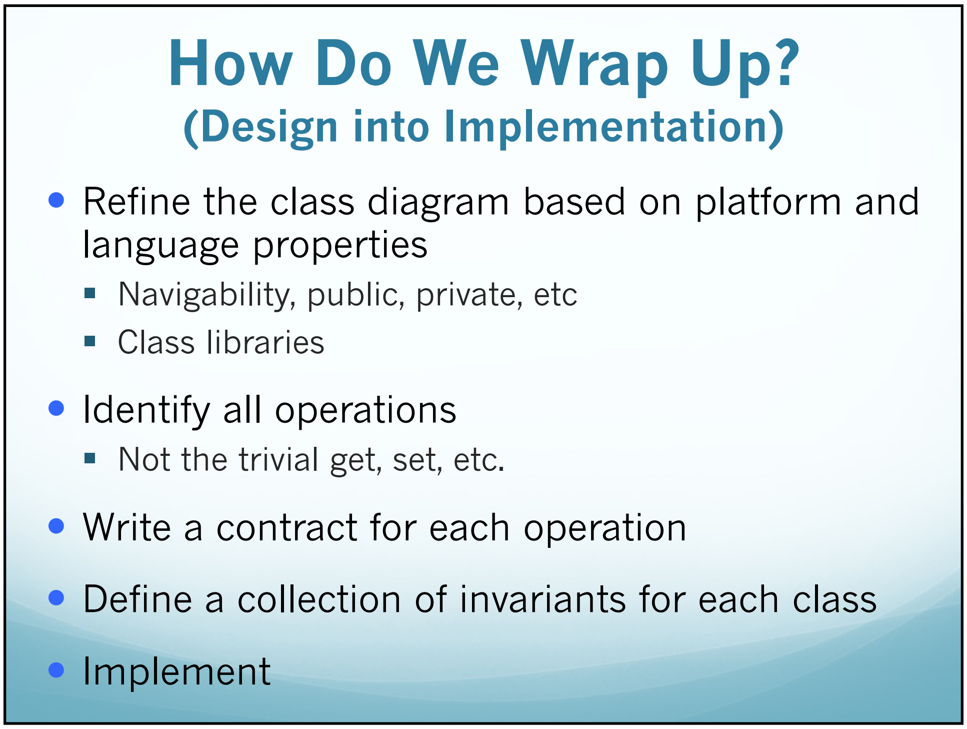
113



How Do We Continue?
(Late Requirements)

- Refine use-cases
  - Possibly some “real” use-cases
    - Rapid prototype; interactive interface elements
- Refine (or restructure) your domain model
  - Based on your physical elements (e.g., sensors, actuators)
- Refine and expand your behavior model (i.e., interacting state diagrams)
  - Until you are comfortable that you understand the required behavior
- Identify most operations and attributes

114

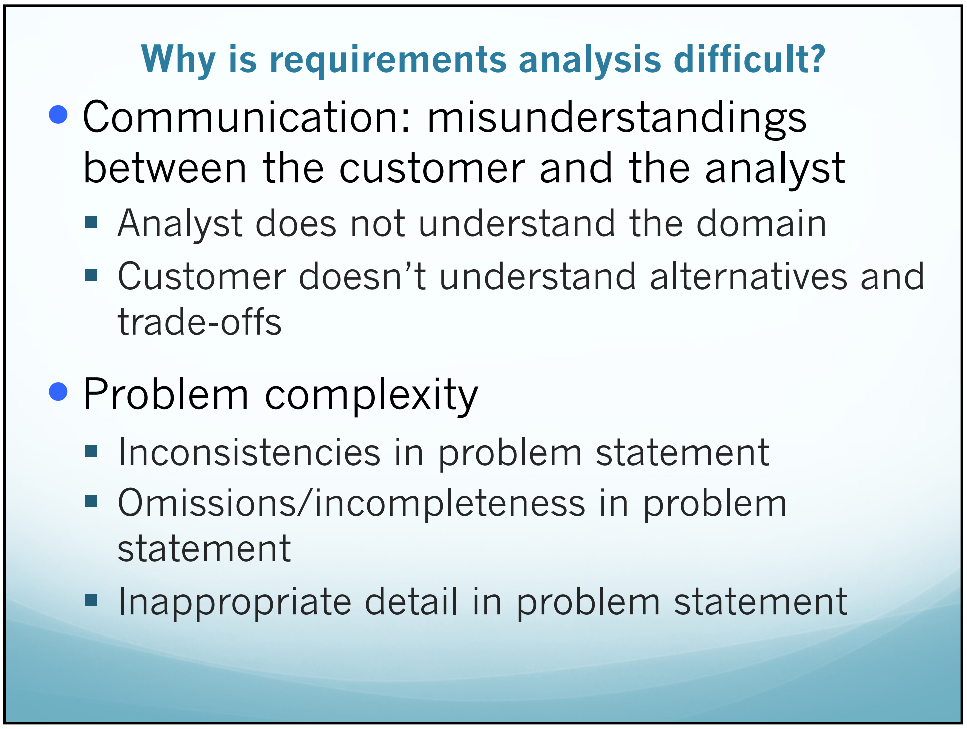


How Do We Wrap Up?

(Design into Implementation)

- 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

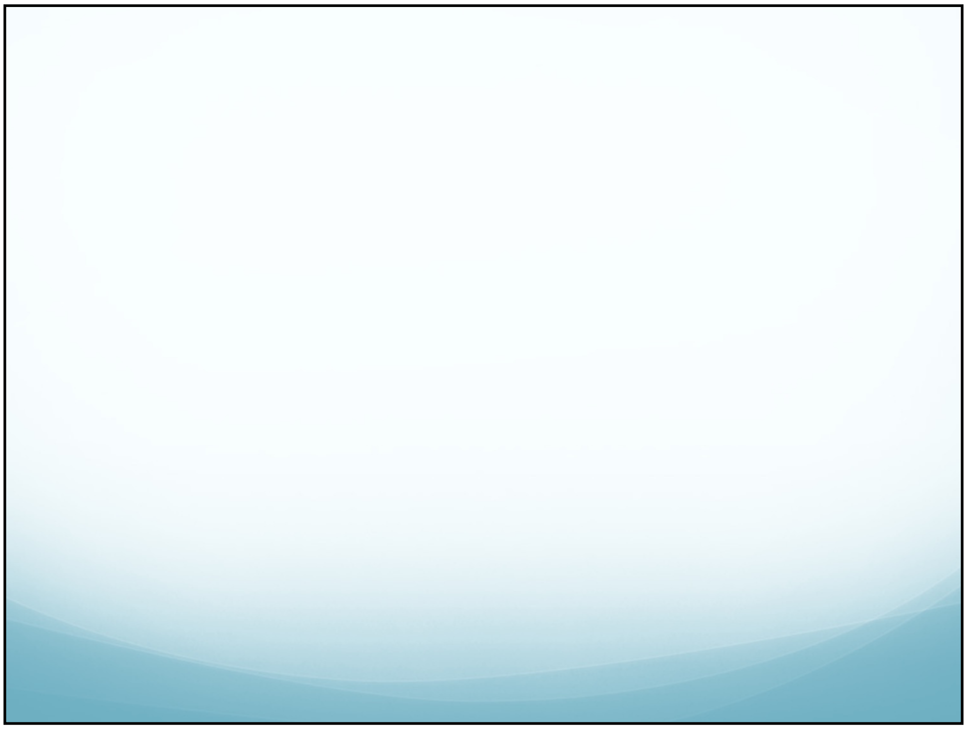
115



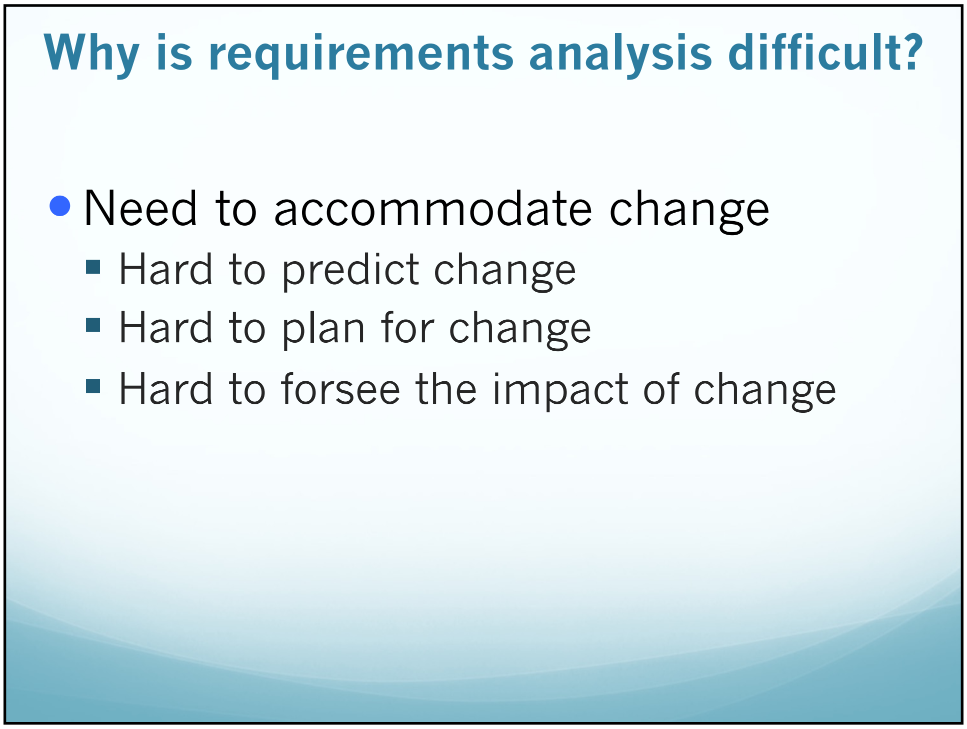
Why is requirements analysis difficult?

- Communication: misunderstandings between the customer and the analyst
  - Analyst does not 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

116



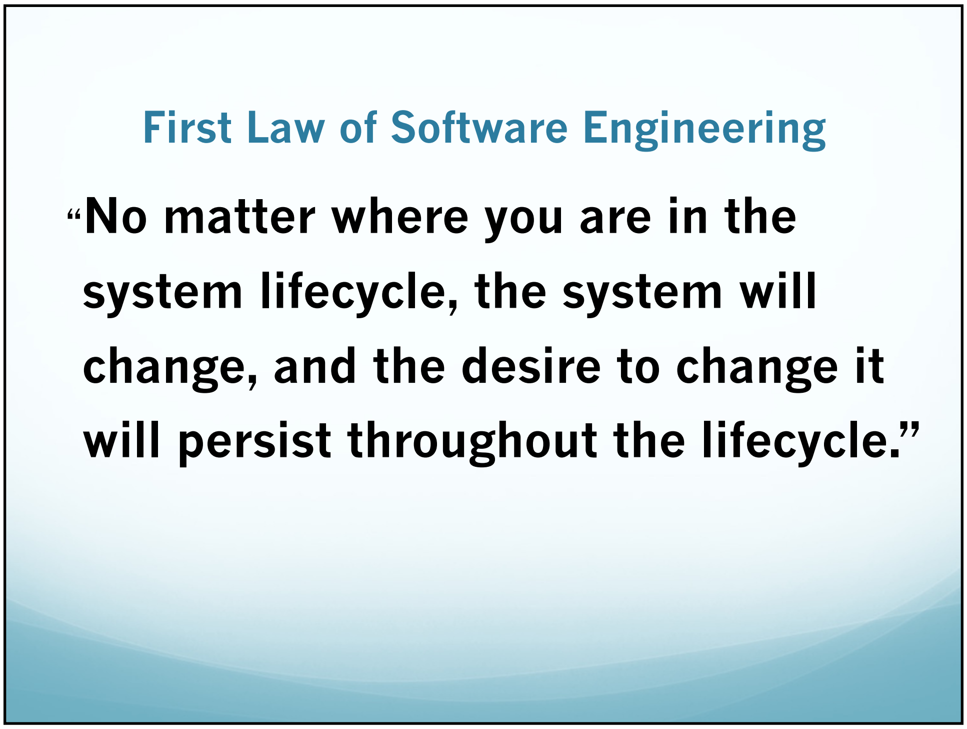
117



Why is requirements analysis difficult?

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

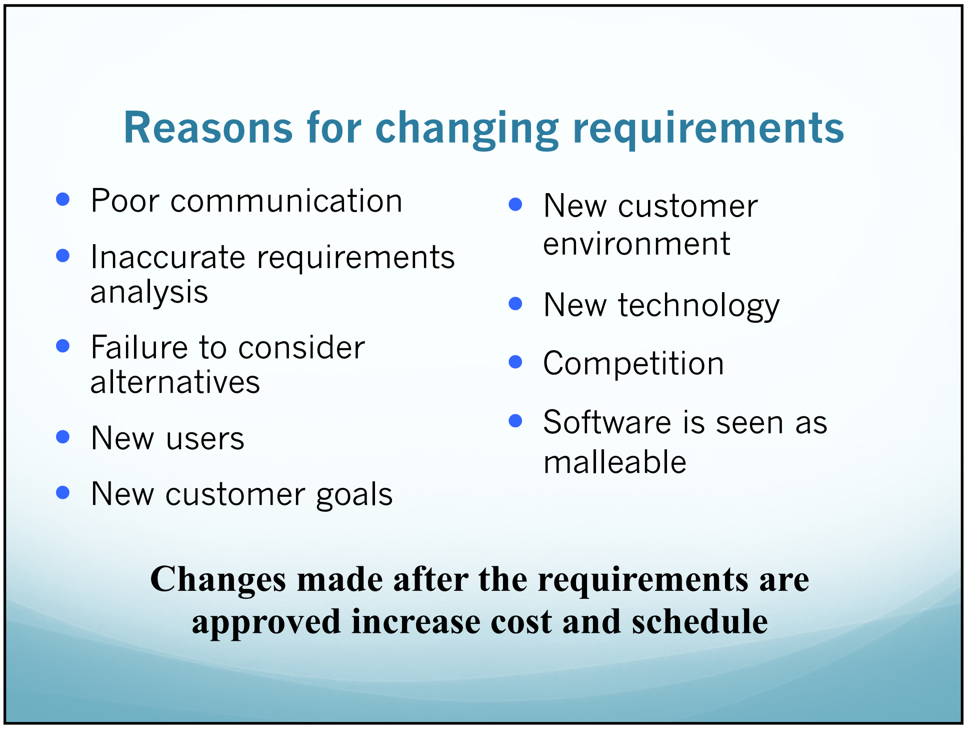
118



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.”

119

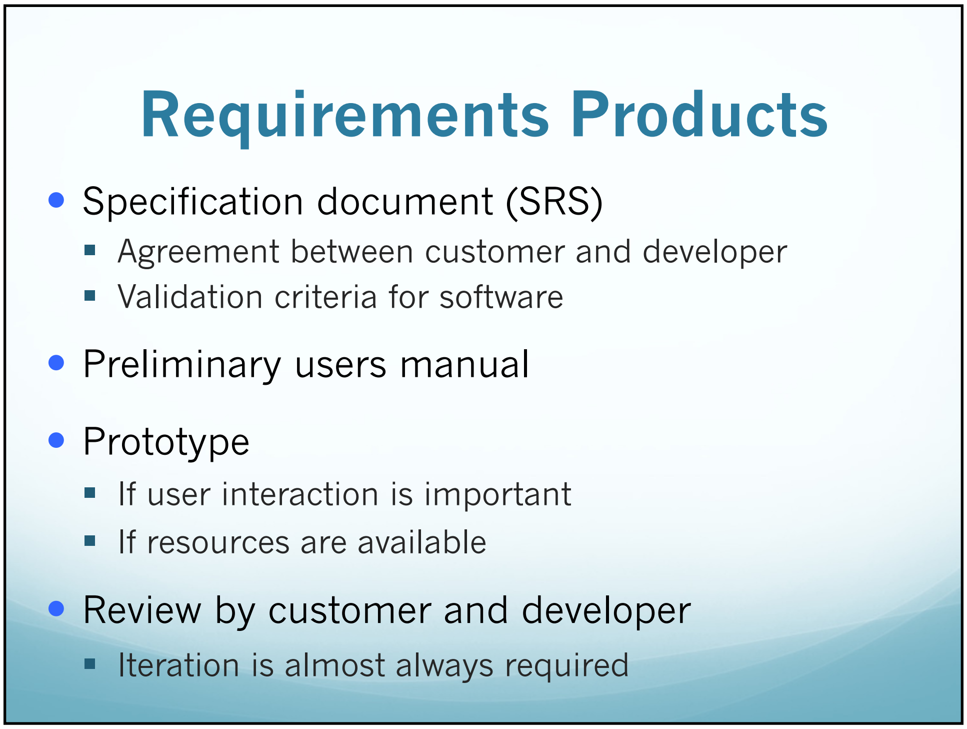


Reasons for changing requirements

- 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

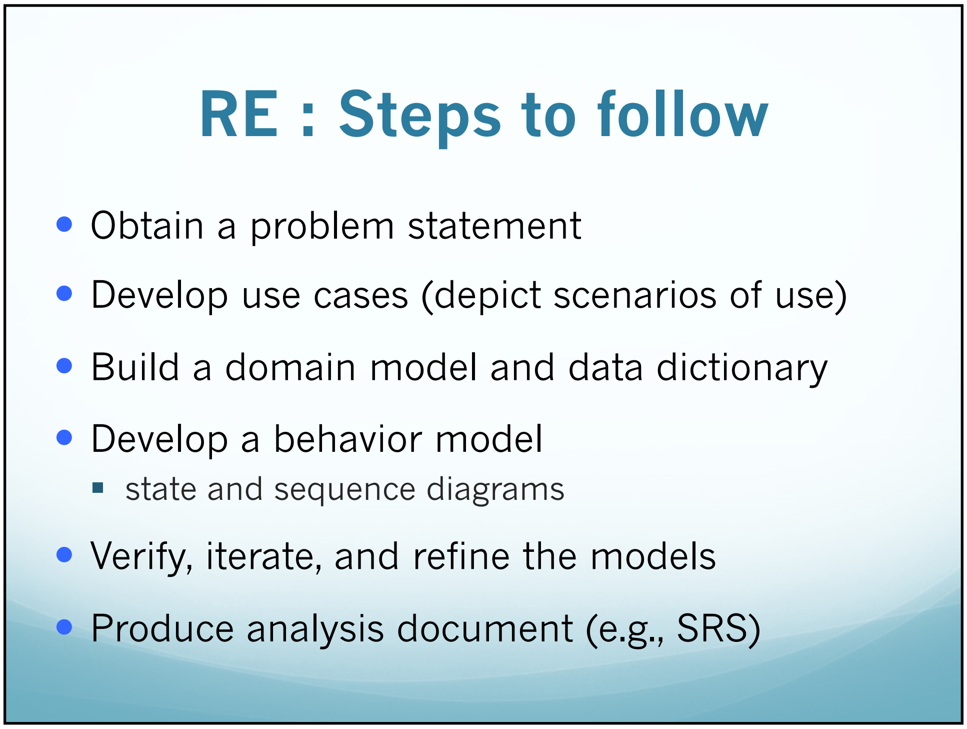
120



Requirements Products

- Specification document (SRS)
  - 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

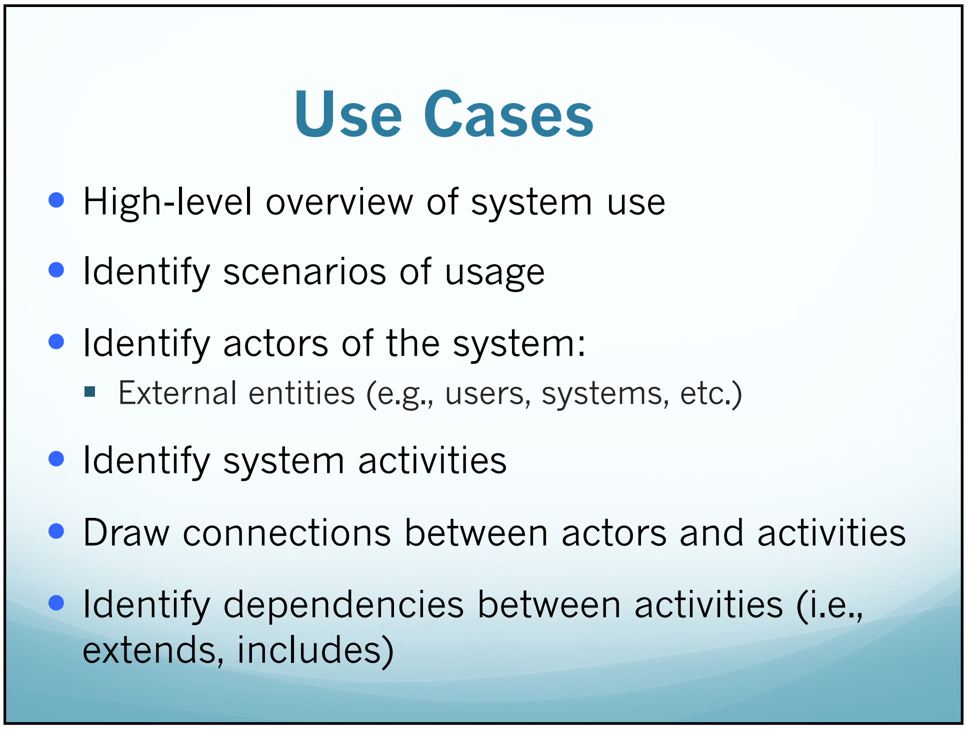
121



RE : Steps to follow

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

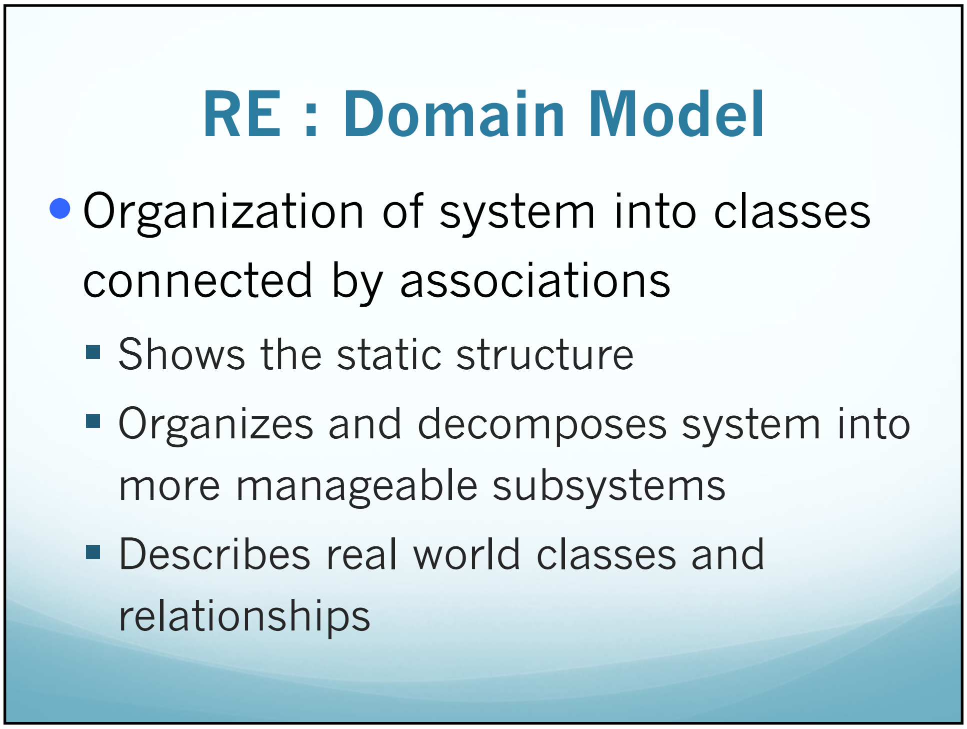
122



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)

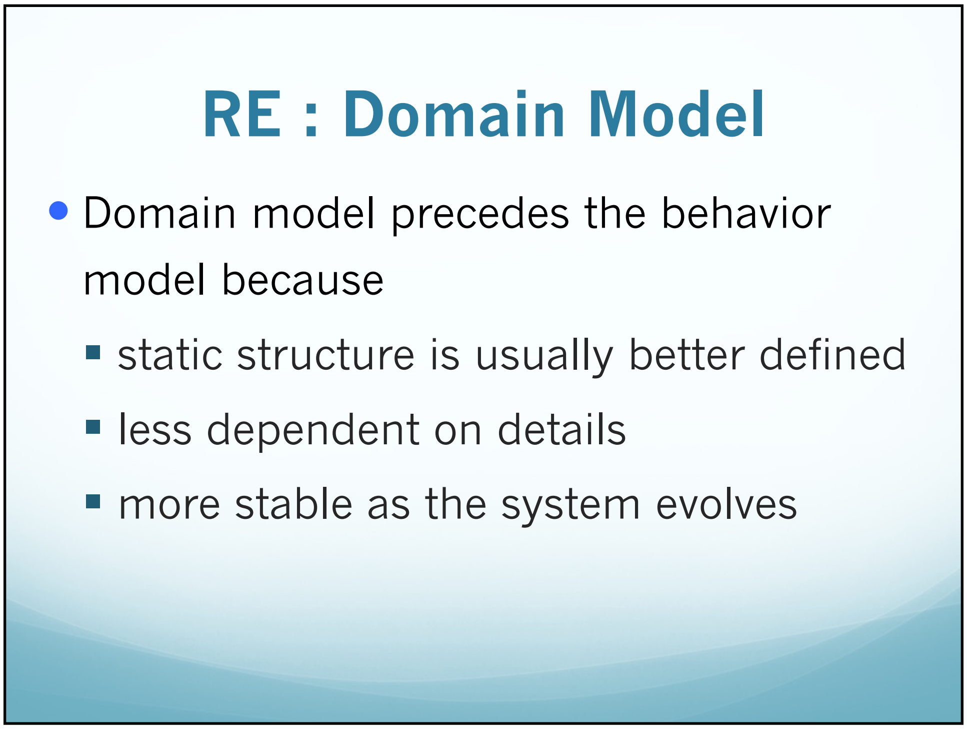
123



RE : Domain 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

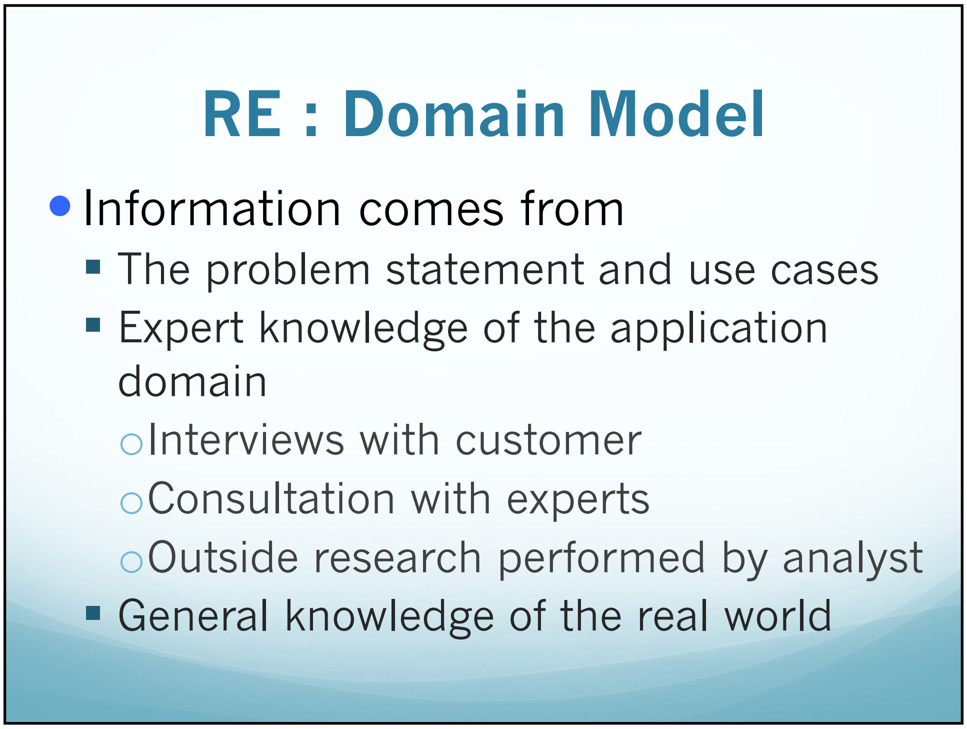
124



RE : Domain Model

- Domain model precedes the behavior model because
  - static structure is usually better defined
  - less dependent on details
  - more stable as the system evolves

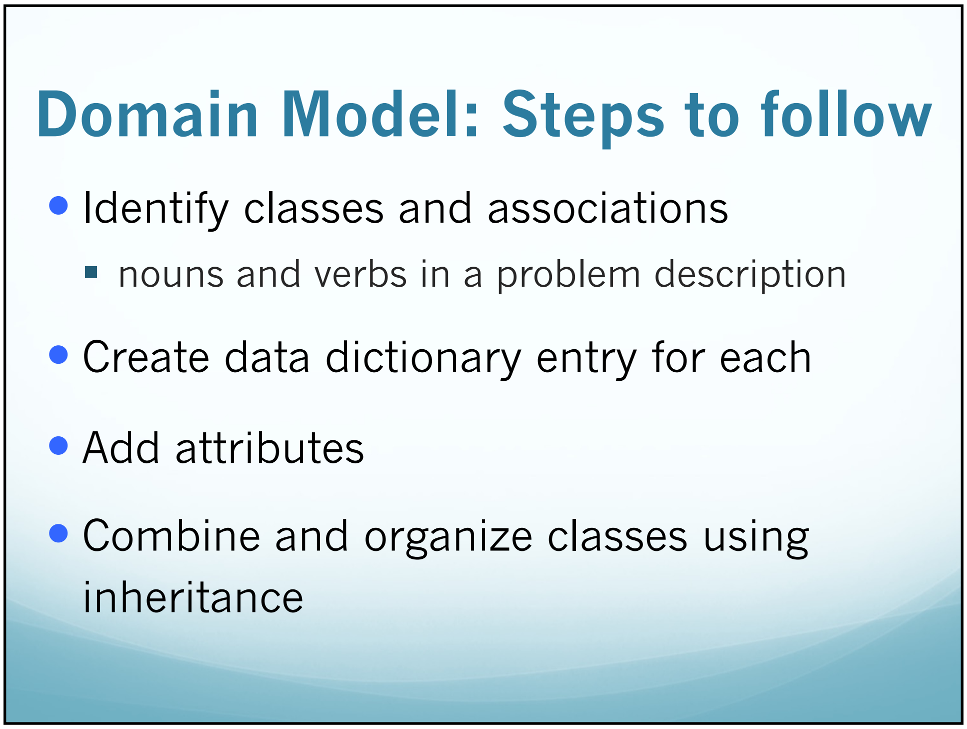
125



RE : Domain 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

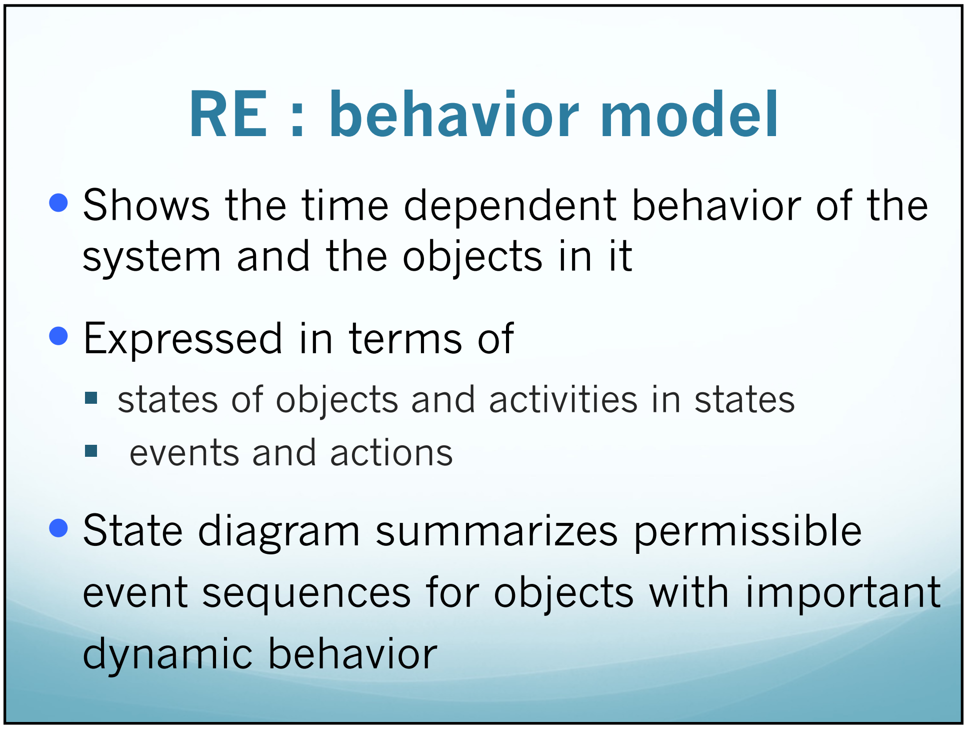
126



Domain 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

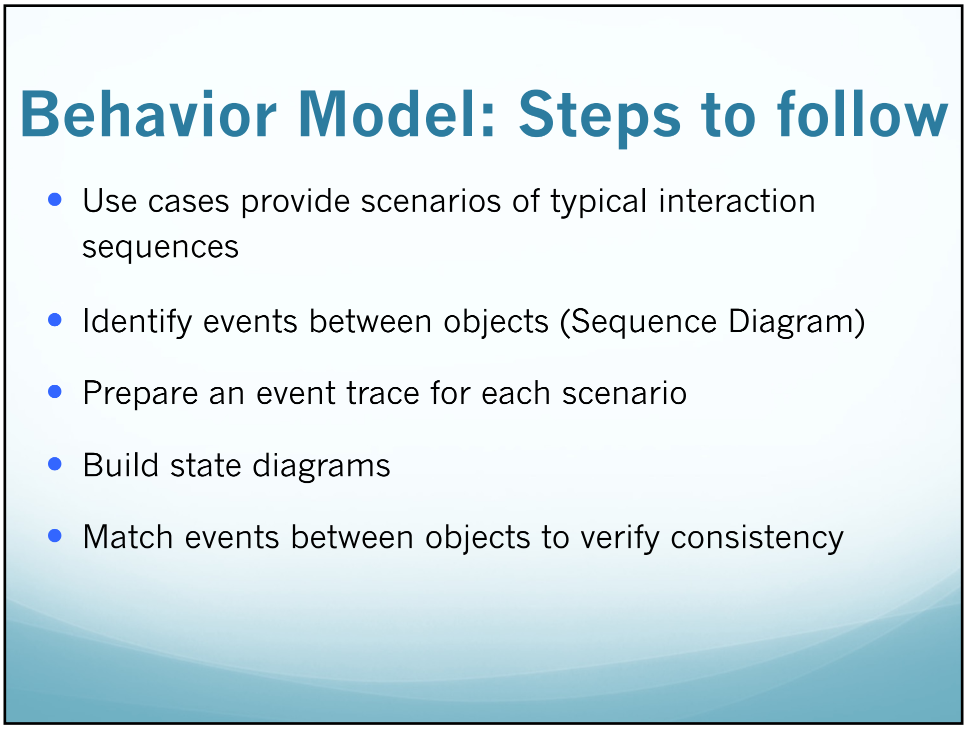
127



RE : behavior 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

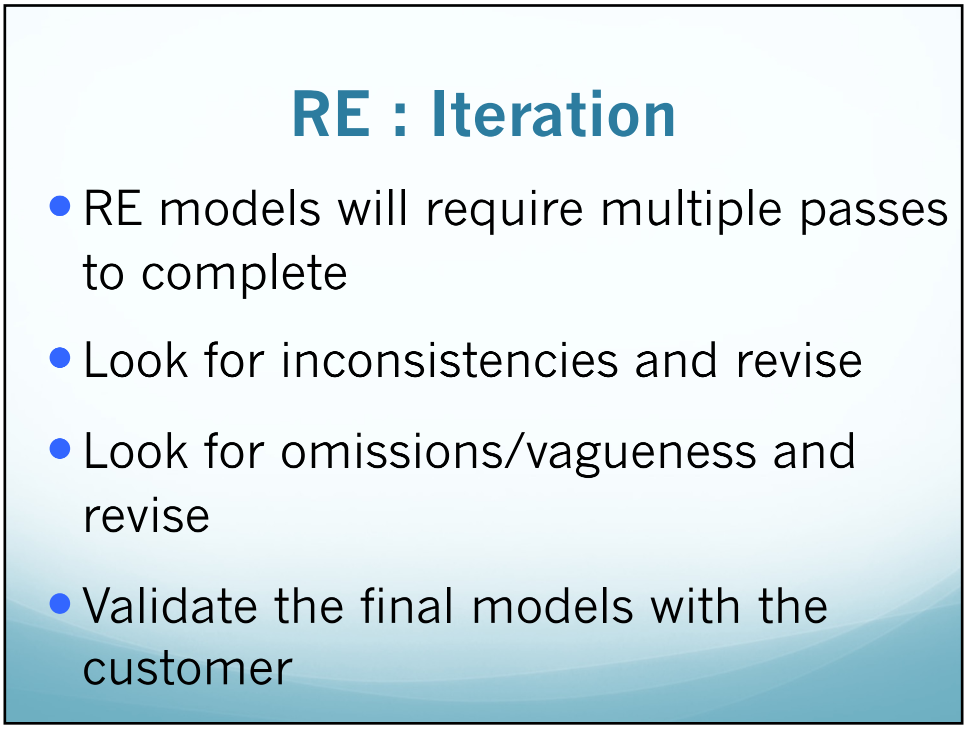
128



Behavior 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

129



RE : Iteration

- RE models will require multiple passes to complete
- Look for inconsistencies and revise
- Look for omissions/vagueness and revise
- Validate the final models with the customer

130