Recap

What is Software Engineering?
- Much more than just writing code!
- The complete process of designing and developing a software system that meets its specification.

Why is Software Engineering important?
- Decomposes a complex engineering problem.
- Organizes processes and effort.
- Improves software reliability.
- Improves developer productivity.

Today
- Modeling and abstraction
- Software architecture
- Software design
- UML overview
Software development: the high-level problem

 Specification

 Source code

 One solution

 Specification

 Source code

 “Here happens a miracle”

 Software development: the high-level problem

 Another solution

 Specification

 Source code

 Modeling the software architecture and design

 What is modeling?

 Building an abstract representation of reality

 - Ignoring (insignificant) details.
 - Level of abstraction depends on viewpoint and purpose:
   - Communication
   - Verification
   - Code generation
 - Focusing on the most important aspects/properties.

 Is abstraction == simplification?
Different levels of abstraction

Source code

Example: Linux Kernel
- 16 million Lines of Code!
- What does the code do?
- Are there different layers?

Call graph

Different levels of abstraction

Source code

Example: Linux Kernel
- 16 million Lines of Code!
- What does the code do?

Kernel architecture
The architecture doesn’t specify the design or implementation details of the individual components (filters)!

Separates data representation (Model), visualization (View), and client interaction (Controller)
Software architecture: examples

MVC example: weather station

<table>
<thead>
<tr>
<th>Current</th>
<th>30 day history</th>
</tr>
</thead>
<tbody>
<tr>
<td>25° F</td>
<td></td>
</tr>
<tr>
<td>-3.9° C</td>
<td>min: 20° F</td>
</tr>
<tr>
<td></td>
<td>max: 35° F</td>
</tr>
</tbody>
</table>

Temp. sensor

Reset history button

Architecture vs. design

Architecture
- Considers the overall software system.
- Considers interactions between components.
- High-level view of the software system
  - shows individual development tasks,
  - allows better effort estimation and progress monitoring,
  - simplifies communication.

Design
- Considers individual components
  - data representation
  - interfaces
  - ...
Boxes, arrows, and annotations are great for visualization, but what are the semantics?

UML crash course

The main questions
- What is UML?
- Is it useful, why bother?
- When to use UML?

What is UML?
- It’s not UMass Lowell.
- Unified Modeling Language.
- Developed in the mid 90’s, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
  - Use case diagrams
  - Component diagrams
  - Class and Object diagrams
  - Sequence diagrams
  - Statechart diagrams
  - ...
What is UML?

- It’s not UMass Lowell.
- Unified Modeling Language.
- Developed in the mid 90’s, improved since.
- Unifies existing, disparate notations.
- Standardizes the notation for modeling OO systems.
- A collection of diagrams for different viewpoints:
  - Use case diagrams
  - Component diagrams
  - Class and Object diagrams
  - Sequence diagrams
  - Statechart diagrams
  - ...

Is UML useful?

Communication
- Forward design (before coding)
  - brainstorm ideas (on whiteboard or paper)
  - draft and iterate over software design

Documentation
- Backward design (after coding)
  - obtain diagram from code

Code generation
- Automatically derive code from diagrams

Classes vs. objects

Class
- Grouping of similar objects.
  - Student
  - Car
- Abstraction of common properties and behavior.
  - Student: Student ID
  - Car: Make and Model

Object
- Entity from the real world.
- Instance of a class
  - Student: Joe, Jane, …
  - Car: Audi A6, Honda Civic, …
UML class diagram: basic notation

<table>
<thead>
<tr>
<th>MyClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>- attr1 : type</td>
</tr>
<tr>
<td># attr2 : type</td>
</tr>
<tr>
<td>+ attr3 : type</td>
</tr>
<tr>
<td>+ bar(a: type) : ret_type</td>
</tr>
<tr>
<td>- foo() : ret_type</td>
</tr>
</tbody>
</table>

Attributes
- <visibility> <name> : <type>
- Static attributes or methods are underlined

Methods
- <visibility> <name>(<param>*) : <return type>
- <param> := <name> : <type>

Visibility
- private
# package-private
# protected
+ public

UML class diagram: inheritance

public class MyClass extends SuperClass implements MyInterface

is-a relationship

UML class diagram: aggregation and composition

Aggregation
- Existence of Part does not depend on the existence of Whole.
- Whole does not own Part.
- Part might be shared with other instances of Whole.

Composition
- Part cannot exist without Whole.
- The lifetime of Part is controlled by Whole.
- Whole is the single owner of Part.

Don’t confuse an is-a relationship with a has-a relationship!
Summary UML

- Unified notation for modeling OO systems.
- Allows different levels of abstraction.
- Suitable for design discussions and documentation.
- Generating code from diagrams is challenging.

In this class, we will use UML class diagrams mainly for visualization and discussion of design and architecture patterns.