Recap

What is Software Engineering?
- The complete process of specifying, designing, developing, analyzing, and maintaining a software system.

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

- Modeling and abstraction
- Software architecture vs. software design
- UML crash course

Software development: the high-level problem

**One solution:** “Here happens a miracle”

**Another solution:** Modeling the 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 dependencies?
Different levels of abstraction

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

Call graph

Layer diagram

Architecture vs. design

What's the difference?
Architecture vs. design

Architecture (what is developed?)
- High-level view of the overall system:
  - What components do exist?
  - What type of storage, layers, etc.?

Design (how are the components developed?)
- Considers individual components:
  - Data representation
  - Interfaces
  - ...

Architecture and Design
- Simplify communication
- Allow effort estimation and progress monitoring

Software architecture: examples

Pipe and Filter
- \texttt{grep CS520 grades.csv | cut -f 1 -d ',' | sort | uniq -c}
- A,CS520,Joe
- A,CS320,Joe
- B,CS520,Jane
- ...
- 28 A
- 12 B
- ...

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

Software architecture: examples

Client-server / n-tier
- Client
- Presentation layer
- Business logic layer
- Data access layer
- DB

Simplifies reusability, exchangeability, and distribution.
Software architecture: examples

**Model View Controller (MVC)**

- **View**
- **Controller**
- **Model**

Client sees
uses
manipulates
updates

Separates data representation (Model), visualization (View), and client interaction (Controller)

Model View Controller (MVC): example

**Simple 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 max: 35° F</td>
</tr>
</tbody>
</table>

Temp. sensor

Reset

Reset history button

Modeling architecture and design: challenges

Great visualization but what are the semantics?
UML crash course

The main questions
- What is UML?
- Is it useful, why bother?
- When to (not) 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
  - ...

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
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: Name and Student ID
  - Car: Make and Model

Object
- Entity from the real world.
- Instance of a class
  - Student: Joe (4711), Jane (4712), ...
  - Car: Audi A6, Honda Civic, ...

UML class diagram: basic notation

```
MyClass
- attr1 : type
# attr2 : type
+ attr3 : type
+ bar(a:type) : ret_type
- foo() : ret_type
```

visibility
- private
~ package-private
# protected
+ public

Attributes
<visibility> <name> : <type>

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

Static attributes or methods are underlined

Classes, abstract classes, and interfaces

<table>
<thead>
<tr>
<th>MyClass</th>
<th>MyAbstractClass</th>
<th>&lt;&lt;interface&gt;&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>{abstract}</td>
<td>MyInterface</td>
</tr>
</tbody>
</table>

Level of detail may vary and depends on context and purpose.
Classes, abstract classes, and interfaces

```java
public class MyClass {
    public void op() {
        ...
    }

    public int op2() {
        ...
    }
}
```

```
public interface MyInterface {
    public void op();
    public int op2();
}
```

UML class diagram: inheritance

```
public class MyClass extends SuperClass implements MyInterface
```

```
public class MyClass {
    public void op() {
        ...
    }

    public int op2() {
        ...
    }
}
```

```
public interface MyInterface {
    public void op();
    public int op2();
}
```

UML class diagram: aggregation and composition

### Aggregation

- **Part** has-a relationship **Whole**
  - 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** has-a relationship **Whole**
  - 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!
Aggregation or composition?

**Composition**
- LinkedList
- Stack

**Aggregation**
- Customer
- Bank

UML class diagram: navigability

- A
- B
- Navigability: not specified
- Navigability: unidirectional
  - “can reach B from A”
- Navigability: bidirectional

UML class diagram: multiplicity

- A
- B
- 1
- 1
  - Each A is associated with exactly one B
  - Each B is associated with exactly one A

- A
- B
- 1..2
  - *
  - Each A is associated with any number of Bs
  - Each B is associated with exactly one or two As

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 patterns.