Recap: Software Engineering

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

Why is it important?
- Software is everywhere and complex.
- Software defects are expensive and range from annoying to life threatening.

Goals
- Decompose a complex engineering problem.
- Organize processes and effort.
- Improve software reliability.
- Improve developer productivity.

Today
- Modeling and abstraction
- Software architecture vs. software design
- UML (Unified Modeling Language) crash course
Software development: the high-level problem

**One solution:** “Hope for the best”

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

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

Software architecture vs. design

**Architecture (what components are developed?)**
- Considers the system as a whole:
  - High-level view of the overall system.
  - What components exist?
  - What type of storage, database, communication, etc?

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

Development process
- Specification
- Architecture
- Design
- Source code
- Level of abstraction
- What's the difference?
A first example: Goal

Goal: group and count CS520 grades.

Software architecture: Pipe and Filter

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

Software architecture: Client-Server / n-tier

Simplifies reusability, exchangeability, and distribution.

Software architecture: Model View Controller

Separates data representation (Model), visualization (View), and client interaction (Controller)
Real World Example: UMass Amherst CICS Weather Station

- Located on top of the CS building
- Here are two different User Interfaces (UIs)

Identifying architecture patterns

1. A web browser (e.g., Chrome, Edge, Safari) applies which of the following architecture patterns?
   a) Client-Server
   b) Model View Controller
   c) Pipe and Filter
Identifying architecture patterns (cont.)

2. Here is a common compiler architecture. Which architecture pattern is being applied?
   a) Client-Server
   b) Model View Controller
   c) Pipe and Filter

Summary: Software architecture vs. design

Architecture and design goals
- Lower complexity: separation of concerns, well defined interfaces
- Simplify communication
- Allow effort estimation and progress monitoring

UML crash course

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

What is UML?
- Unified Modeling Language.
- Developed in the mid 90’s, improved since.
- Standardized 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?

- Unified Modeling Language.
- Developed in the mid 90’s, improved since.
- Standardized 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
  - ...

Are UML diagrams 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 source code.

Code generation
- Generating source code from diagrams is challenging.
- Code generation may be useful for skeletons.

In this class, we will use UML class diagrams mainly for visualization and discussion purposes.
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
- Come from the real world.
- Instance of a class
  - Student: Juan (4711), Jane (4712), ...
  - Car: Audi A6, Honda Civic, Tesla S, ...

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>

Name
Attributes
visibility : name : type

Methods
visibility : name(\param\*) : return type
\param\ : name : type

Visibility
- private
- package-private
- protected
- public
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

UML class diagram: concrete example

public class Person {
}

public class Student extends Person {
  private int id;
  public Student(String name, int id) {
    ...
  }
  public int getId() {
    return this.id;
  }
}

private class MyClass {
  public void op1() {
    ...
  }
  public int op2() {
    ...
  }
}

Level of detail in a given class or interface may vary and depends on context and purpose.
UML class diagram: Inheritance

```
public class SubClass extends SuperClass implements AnInterface
```

UML class diagram: Aggregation and Composition

- **Aggregation**
  - Part
    - has-a relationship
  - Whole

- **Composition**
  - Part
    - has-a relationship
  - Whole

- Existence of Part does not depend on the existence of Whole.
- Lifetime of Part does not depend on Whole.
- No single instance of whole is the unique owner of Part (might be shared with other instances of Whole).
- Part cannot exist without Whole.
- Lifetime of Part depends on Whole.
- One instance of Whole is the single owner of Part.

Aggregation or Composition?

```
Room

Customer

Building

Bank
```

```
Room

Customer

Building

Bank
```

What about class and students or body and body parts?
Questions about the UML class diagram example

1. Which classes implement TimedDevice?
2. For class AbstractCGMReceiver:
   a. How many fields?
   b. How many methods?
3. Which class extends AbstractCGMReceiver?
4. What is the relationship between AbstractCGMReceiver and Alert?
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.