CS 520/620
Advanced Software Engineering
Spring 2016

January 21, 2016
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.
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
Software development: the high-level problem

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?
Different levels of abstraction

Source code

Call graph

Example: Linux Kernel

- 16 million Lines of Code!
- What does the code do?
Different levels of abstraction

Source code

Call graph

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

Source code

Kernel architecture

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

Pipe and Filter

```
grep CS520 grades.csv | cut -f 1 -d ',' | sort | uniq -c
```

```
A,CS520,Joe
A,CS320,Joe
B,CS520,Jane
...
```

```
28 A
12 B
...
```
Software architecture: examples

Pipe and Filter

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

```
A,CS520,Joe
A,CS320,Joe
B,CS520,Jane
...

grep CS520 grades.csv | cut -f 1 -d ',' | sort | uniq -c

28 A
12 B
...
```
Software architecture: examples

Model View Controller (MVC)
Software architecture: examples

Model View Controller (MVC)

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><strong>25° F</strong></td>
<td>![Temperature Graph]</td>
</tr>
<tr>
<td><strong>-3.9° C</strong></td>
<td>min: <strong>20° F</strong> max: <strong>35° F</strong></td>
</tr>
</tbody>
</table>

- **Temp. sensor**
- **Reset history button**

- **Reset** 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
  - ...
Architecture vs. design

- Specification
- Architecture
- Design
- Source code
Architecture vs. design

Development process

Specification

Architecture

Level of abstraction

Design

Source code
Modeling architecture and design: challenges

- Client sees
- Client uses
- View updates
- Controller manipulates
- Model
Modeling architecture and design: challenges

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
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

MyClass

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

Visibility

- private
~ package-private
# protected
+ public

Static attributes or methods are underlined

Attributes

\(<visibility> \ <name> : \ <type>\)

Methods

\(<visibility> \ <name>(<param>*) : \ <return type>\)
\(<param> := \ <name> : \ <type>\)
Classes, abstract classes, and interfaces
public class MyClass extends SuperClass implements MyInterface
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!
UML class diagram: navigability

Navigability: not specified

Navigability: unidirectional “can reach B from A”

Navigability: bidirectional
UML class diagram: multiplicity

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

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 and architecture patterns.