CS 520

Theory and Practice of Software Engineering
Fall 2018

Software architecture and design/UML crash course

September 6, 2018

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 annoying).

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

Recap: Software Engineering

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

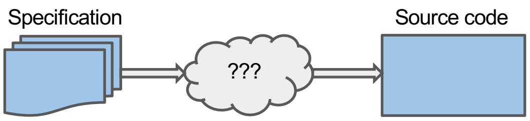
Why is it important?
- Software is everywhere and complex.
- Software defects are expensive (and annoying).

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

Today

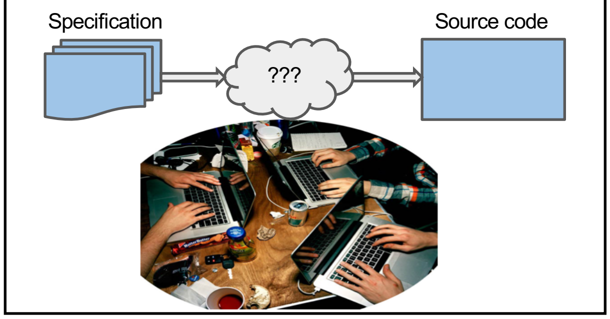
- Modeling and abstraction
- Software architecture vs. software design
- UML crash course

Software development: the high-level problem



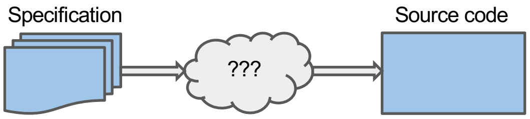
Software development: the high-level problem

One solution: "*Here happens a miracle*"



Software development: the high-level problem

Another solution: Modeling the architecture and design


Specification

???

Source code

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

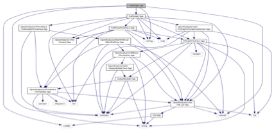
Example: Linux Kernel

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

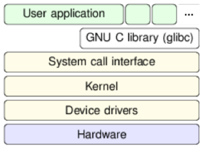
Different levels of abstraction

Source code

Call graph



Layer diagram


User application

...

GNU C library (glibc)

System call interface

Kernel

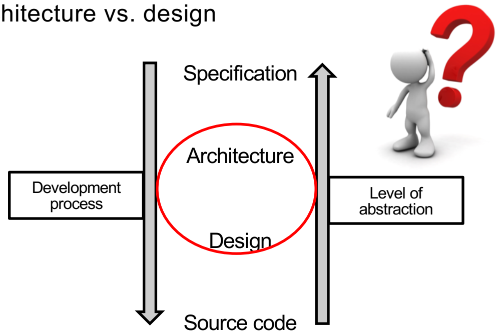
Device drivers

Hardware

Example: Linux Kernel

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

Architecture vs. design


Specification

Architecture

Design

Source code

Development process

Level of abstraction

What's the difference?

Software architecture vs. design

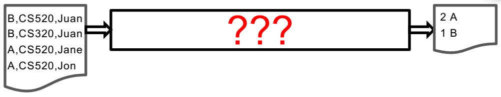
Architecture (what components are developed?)

- 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?)

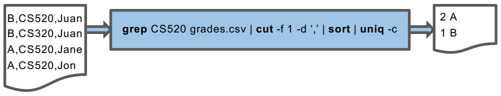
- Considers individual components:
    - Data representation
    - Interfaces, Class hierarchies
    - …

A first example



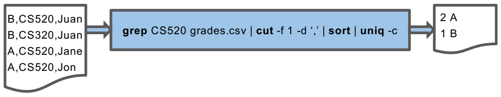
Goal: group and count CS520 grades.

Architecture or design pattern?



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

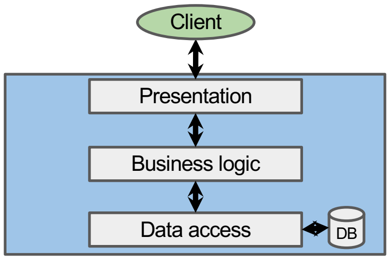
Software architecture: Pipe and Filter



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

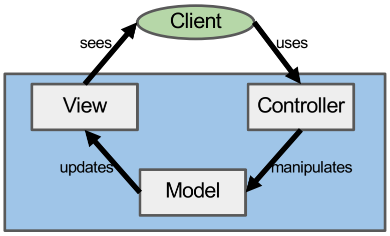
The architecture doesn't specify the design or implementation details of the individual components (filters)!

Software architecture: Client-server / n-tier



Simplifies reusability, exchangeability, and distribution.

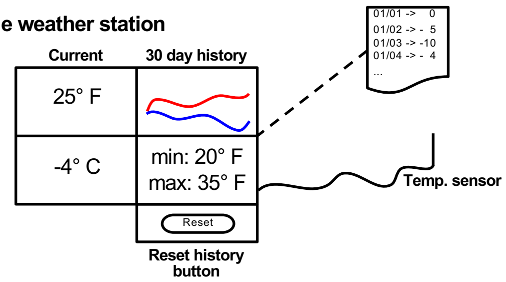
Software architecture: **M**odel **V**iew **C**ontroler



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

Model View Controler: example

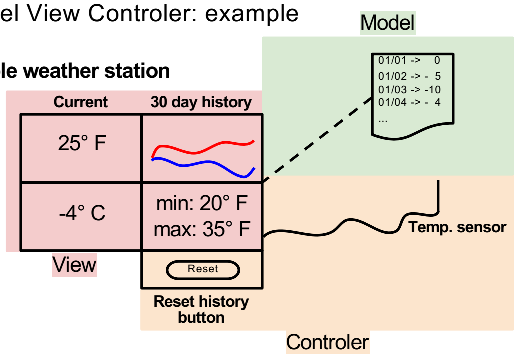
Simple weather station



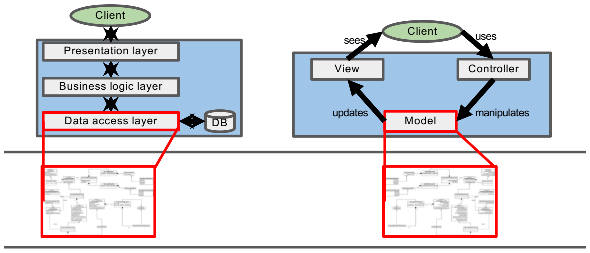
```
01/01 ->   0
01/02 -> -  5
01/03 -> -10
01/04 -> -  4
...
```

Model View Controler: example

Simple weather station



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

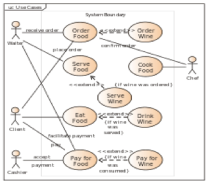
- What is UML?
- Is it useful, why bother?
- When to (not) use UML?

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
  - ...

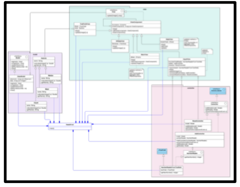
What is UML?

- 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
  - ...

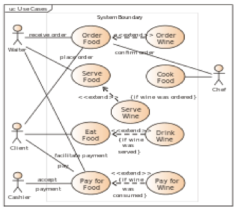


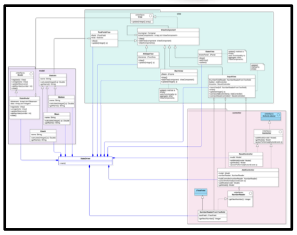
What is UML?

- 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?





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

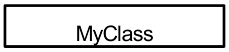
Class

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

Object

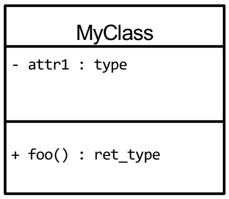
- cfrom the real world.
- Instance of a class
  - Student: Juan (4711), Jane (4712), …
  - Car: Audi A6, Honda Civic, Tesla S,...

UML class diagram: basic notation



MyClass

UML class diagram: basic notation



MyClass

```
- attr1 : type
```

```
+ foo() : ret_type
```

Name

Attributes

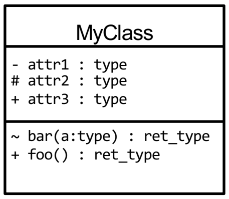
<visibility> <name> : <type>

Methods

`<visibility> <name>(<param>*) : <return type>`

<param> := <name> : <type>

UML class diagram: basic notation



MyClass

```
- attr1 : type
# attr2 : type
+ attr3 : type
```

```
~ bar(a:type) : ret_type
+ foo() : ret_type
```

Name

Attributes

<visibility> <name> : <type>

Methods

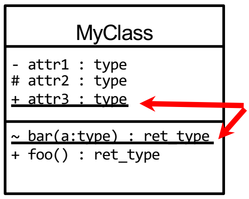
`<visibility> <name>(<param>*) : <return type>`

<param> := <name> : <type>

Visibility

- private
~ package-private
# protected
+ public

UML class diagram: basic notation



Name

Attributes

<visibility> <name> : <type>

Static attributes or methods are underlined

Methods

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

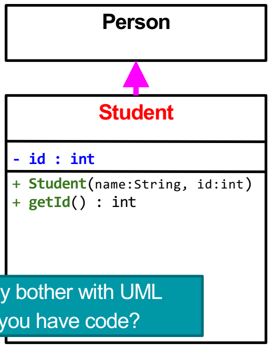
Visibility

- private
~ package-private
# protected
+ public

UML class diagram: concrete example

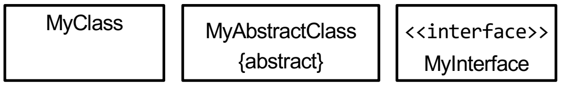
```
public class Person {
  ...
}
```

```
public class Student
    extends Person {

  private int id;

  public Student(String name,
                 int id) {
    ...
  }

  public int getId() {
    return this.id;
  }
}
```

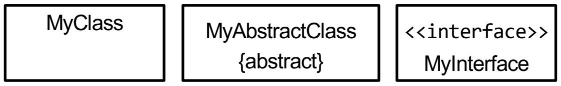


So why bother with UML when you have code?

Classes, abstract classes, and interfaces



Classes, abstract classes, and interfaces



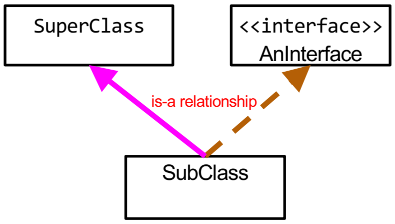
```
public class MyClass {

  public void op() {
    ...
  }

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

```
public abstract class
    MyAbstractClass {

  public abstract void op();


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

```
public interface
    MyInterface {

  public void op();


  public int op2();
}
```

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

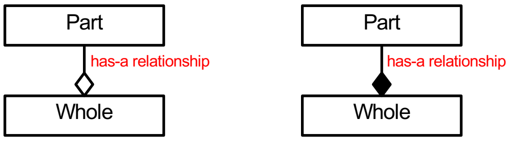
UML class diagram: Inheritance



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

UML class diagram: Aggregation and Composition

Aggregation **Composition**



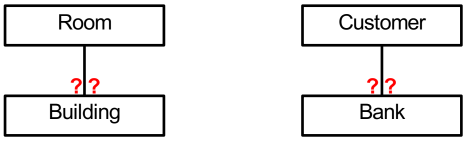
Aggregation:

- 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).

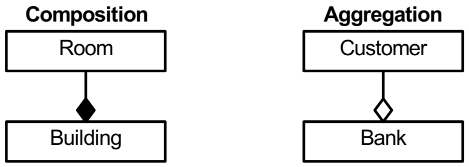
Composition:

- Part cannot exist without Whole.
- Lifetime of Part depends on Whole.
- One instance of Whole is the single owner of Part.

Aggregation or Composition?

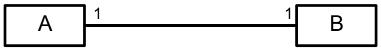


Aggregation or Composition?

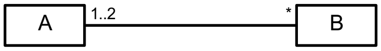


What about class and students or body and body parts?

UML class diagram: multiplicity

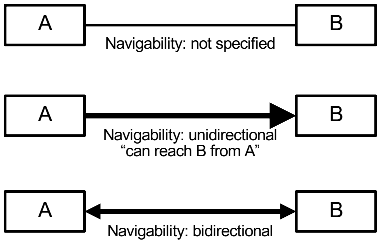


Each A is associated with exactly one B
Each B is associated with exactly one A



Each A is associated with any number of Bs
Each B is associated with exactly one or two As

UML class diagram: navigability

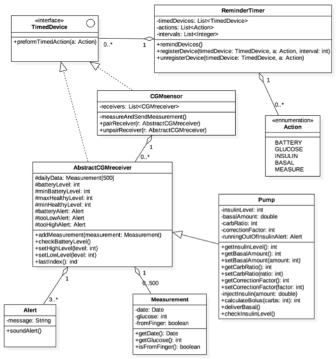


Navigability: not specified

Navigability: unidirectional
"can reach B from A"

Navigability: bidirectional

UML class diagram: example


«interface»
TimedDevice
+preformTimedAction(a: Action)

ReminderTimer
-timedDevices: List<TimedDevice>
-actions: List<Action>
-intervals: List<Integer>
+remindDevices()
+registerDevice(timedDevice: TimedDevice, a: Action, interval: int)
+unregisterDevice(timedDevice: TimedDevice, a: Action)

CGMsensor
-receivers: List<CGMreceiver>
-measureAndSendMeasurement()
+pairReceiver(r: AbstractCGMreceiver)
+unpairReceiver(r: AbstractCGMreceiver)

«ennumeration»
Action
BATTERY
GLUCOSE
INSULIN
BASAL
MEASURE

AbstractCGMreceiver
#dailyData: Measurement[500]
#batteryLevel: int
#minBatteryLevel: int
#maxHealthyLevel: int
#minHealthyLevel: int
#batteryAlert: Alert
#tooLowAlert: Alert
#tooHighAlert: Alert
+addMeasurement(measurement: Measurement)
+checkBatteryLevel()
+setHighLevel(level: int)
+setLowLevel(level: int)
+lastIndex(): ind

Pump
-insulinLevel: int
-basalAmount: double
-carbRatio: int
-correctionFactor: int
-runningOutOfInsulinAlert: Alert
+getInsulinLevel(): int
+getBasalAmount(): int
+setBasalAmount(amount: int)
+getCarbRatio(): int
+setCarbRatio(ratio: int)
+getCorrectionFactor(): int
+setCorrectionFactor(factor: int)
-injectInsulin(amount: double)
+calculateBolus(carbs: int): int
+deliverBasal()
+checkInsulinLevel()

Alert
-message: String
+soundAlert()

Measurement
-date: Date
-glucose: int
-fromFinger: boolean
+getDate(): Date
+getGlucose(): int
+isFromFinger(): boolean

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.