# CS 520

Theory and Practice of Software Engineering
Fall 2019

**Best and worst programming practices**

September 12, 2019

## Reminder

Class website:
https://people.cs.umass.edu/~brun/class/2019Fall/CS520/

**Schedule:**

(subject to change; check regularly)

| week | date | day | topic | homework | project |
|---|---|---|---|---|---|
| Week 1 | Sep 3 | Tu | Course introduction | | |
| | Sep 5 | Th | Software fairness | Homework 1 Due: Tu September 17, 2019, 9:00AM EDT | |
| Week 2 | Sep 10 | Tu | Software architecture and design | | |
| | Sep 12 | Th | Best and worst programming practices | | |
| Week 3 | Sep 17 | Tu | Object oriented design principles | | |
| | Sep 19 | Th | Object oriented design patterns | | Project topic selection due: Tu Oct 8, 2019, 9:00AM EDT |
| Week 4 | Sep 24 | Tu | Version control systems | | |
| | Sep 26 | Th | Advanced uses of git | | |
| | Oct 1 | Tu | Collaboration and pair programming | | |

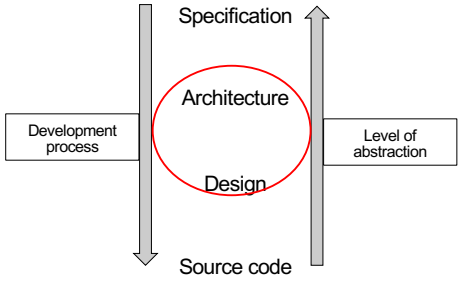
## Reminder

Class website:
https://people.cs.umass.edu/~brun/class/2019Fall/CS520/

Instructor: Yuriy Brun
office: 302 computer science building
office hours: Tuesday 11:15PM–12:00PM
email: brun@cs.umass.edu

TA: Joshua Levine
office: CS 207 (cube 1)
office hours: Wednesday 1:00PM–2:00PM
email: joshualevine@cs.umass

## Recap: software architecture vs. design

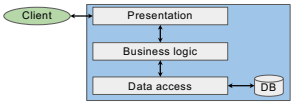



What's the difference?

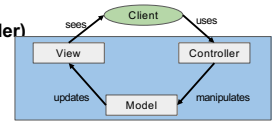
## Recap: software architecture examples

- **Pipe and filter**

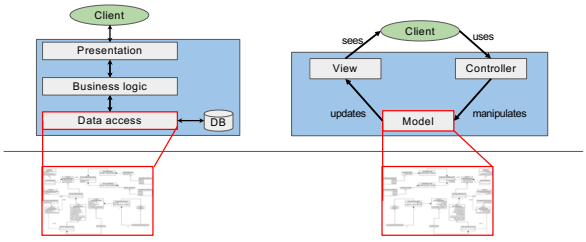

- **N-tier / client-server**




- **MVC (Model-View-Controller)**




## Recap: software architecture and design goals

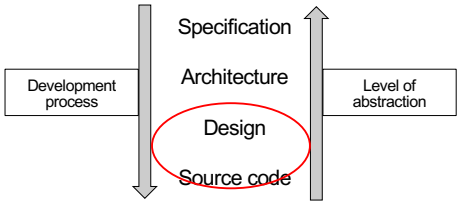



**Architecture and design goals**

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

## Today

- An in-class discussion on best and worst programming practices.




## setup and goals

- 4-person teams
- 6 code snippets
- 4 rounds
  - **First round**
    - For each of 3 code snippet, decide whether it represents good or bad practice.
    - **Goal:** discuss and reach consensus on good or bad practice.
  - **Second round** (known solutions)
    - For each code snippet, try to understand why it is good or bad practice.
    - **Goal:** come up with one or more explanations or a counter argument.

  and then repeat with 3 more code snippets

## Round 1: good or bad?

## Snippet 1: good or bad?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

## Snippet 2: good or bad?

```
public void addStudent(Student student, String course) {
   if (course.equals("CS520")) {
      cs520Students.add(student);
   }
   allStudents.add(student)
}
```

## Snippet 3: good or bad?

```
public enum PaymentType {DEBIT, CREDIT}
public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
       throw new IllegalArgumentException("Unexpected payment type");
    }
}
```

## Solutions

- Snippet 1: bad
- Snippet 2: bad
- Snippet 3: good

## Round 2: why is it good or bad?

## Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

## Snippet 1: this is bad! why?

```
public File[] getAllLogs(Directory dir) {
   if (dir == null || !dir.exists() || dir.isEmpty()) {
      return null;
   } else {
      int numLogs = … // determine number of log files
      File[] allLogs = new File[numLogs];
      for (int i=0; i<numLogs; ++i) {
         allLogs[i] = … // populate the array
      }
      return allLogs;
   }
}
```

```
File[] files = getAllLogs();
for (File f : files) {
   …
}
```

Don't return null; return an empty array instead.

## Snippet 2: short but bad! why?

```
public void addStudent(Student student, String course) {
   if (course.equals("CS520")) {
      cs520Students.add(student);
   }
   allStudents.add(student)
}
```

## Snippet 2: short but bad! why?

```
public void addStudent(Student student, String course) {
   if (course.equals("CS520")) {
      cs520Students.add(student);
   }
   allStudents.add(student)
}
```

Defensive programming: write the literal first (or add an explicit assertion).

## Snippet 3: this is good, but why?

```
public enum PaymentType {DEBIT, CREDIT}
public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
       throw new IllegalArgumentException("Unexpected payment type");
  }
}
```

## Snippet 3: this is good, but why?

```
public enum PaymentType {DEBIT, CREDIT}
public void doTransaction(double amount, PaymentType payType) {
  switch (payType) {
    case DEBIT:
       … // process debit card
       break;
    case CREDIT:
       … // process credit card
       break;
    default:
       throw new IllegalArgumentException("Unexpected payment type");
  }
}
```

Type safety using an enum; throws an exception for unexpected cases (e.g., future extensions of PaymentType).

## Round 3: more snippets

## Snippet 4: good or bad?

```
public int getAbsMax(int x, int y) {
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

## Snippet 5: good or bad?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

## Snippet 6: good or bad?

```
public class Point {
   private final int x;
   private final int y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

## Solutions

- Snippet 1: bad
- Snippet 2: bad
- Snippet 3: good
- Snippet 4: bad
- Snippet 5: bad
- Snippet 6: good

## Round 4: why is it good or bad?

## Snippet 4: also bad! huh?

```
public int getAbsMax(int x, int y) {
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

## Snippet 4: also bad! huh?

```
public int getAbsMax(int x, int y){
  if (x<0) {
    x = -x;
  }
  if (y<0) {
    y = -y;
  }
  return Math.max(x, y);
}
```

Method parameters should be final; use local variables to sanitize inputs.

## Snippet 5: Java API, but still bad! why?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o) {
      …
   }
   …
}
```

## Snippet 5: Java API, but still bad! why?

```
public class ArrayList<E> {
   public E remove(int index) {
      …
   }
   public boolean remove(Object o){
      …
   }
   …
}
```

```
ArrayList<String> l = new ArrayList<>();
Integer index = new Integer(1);
l.remove(index);
```

Avoid method overloading, which is statically resolved. Autoboxing/unboxing adds additional confusion.

## Snippet 6: this is good, but why?

```
public class Point {
   private final int x;
   private final int y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

## Snippet 6: this is good, but why?

```
public class Point {
   private final int x;
   private final int y;
   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return this.x;
   }
   public int getY() {
      return this.y;
   }
}
```

Good encapsulation; immutable object.