Engineering of Software Subsystems Course Overview
Up to this point you have only spent a little time talking about software design in general.

That is about to change because this course is about **Design**
But we must digress to handle an administrative matter.

- Do you have an active account in the SE domain?
- Can you login to the machine?
- Do you have access to your section’s myCourses website 4010-362-xx?
- Can you access the 362 course information on the SE web server?

http://www.se.rit.edu/~se362
By this point you each have some procedure that you follow to create an object-oriented design.

- What procedure do you follow?
- Go to the ~se362 website schedule for the first class. Open the Design Process document and capture your process in one or two sentences for each question.
- Deposit this in the Design Process dropbox.
You have learned low-level OOP design in CS courses and some larger design in SE361.

- **CS1 – 3: first principles of OOP**
  - *Find the nouns* → objects/state
  - *Find the verbs* → behaviors; methods/functions
  - Encapsulation, inheritance
  - Programming

- **SE361: larger design problem**
  - *Some design principles and trade-offs*
  - *Introduction to design patterns*
  - *Introduction to static and dynamic modeling*
This course discusses standard patterns of structure and interaction between classes.

- Standard patterns of structure and interaction between classes
  - Design patterns
- How to apply them to your application
  - Deal with subsystems at the higher level of abstraction provided by the patterns
- What to do when it does not fit exactly
  - Evaluate options and analyze the trade-offs
At the code level you know some standard patterns.

- How do you walk through an array in Java?

```java
for (i = 0; i < array.length; i++) {
    // use the array element
}
```
Our level of discussion for this course is a small subsystem of 3 to 10 classes.

- Higher than what we've done before
  - *Not specific data structures*
  - *Not algorithmic approaches*

- Lower than whole architectures or frameworks
  - *Not financial systems*
  - *Not air-traffic control*
  - *Not J2EE*
The next level of design requires an awareness of the principles that underlie “good” designs.

- All engineering is based on principles that have been learned over time and many applications and some failures.
- What software design principles have you seen?
- Go to the ~se362 website schedule for the first class. Open the Two Design Principles document and refresh your memory by answering the questions.
- Deposit this in the Two Design Principles dropbox.
There are some key object-oriented design concepts that we will stress.

- Increase cohesion where possible
- Decrease coupling where possible
- Behaviors follow data
- Prefer type (interface) inheritance over class (implementation) inheritance. “Program to the interface, not the implementation.”
- Prefer composition to inheritance
- Use delegation to “simulate” runtime inheritance.
- Law of Demeter “Only talk to your friends.”
What Are Patterns?

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Christopher Alexander

A pattern is a *general* solution to a *problem* in a *context*

- *general* -- outline of approach only
- *problem* -- a recurring issue
- *context* -- consider the expected design evolution
Patterns allow us to gain from the experience, and mistakes, of others.

- Design for re-use is difficult
- Experienced designers:
  - *Rarely start from first principles*
  - *Apply a working "handbook" of approaches*
- Patterns make this ephemeral knowledge available to all
- Support evaluation of alternatives at higher level of abstraction
The main classification for Gang-of-Four design patterns is by purpose of the pattern’s intent.

- **Creational**: intention is mainly about creating objects
- **Structural**: intention is mainly about the structural relationship between the objects
- **Behavioral**: intention is mainly about the interactions between the objects
A second dimension for classification is binding time.

- Using inheritance is compile-time binding or class-based
- Using delegation or composition is run-time binding or object-based

**Creational**
- `class` => defer creation to subclasses
- `object` => defer creation to another object

**Structural**
- `class` => structure via inheritance
- `object` => structure via composition

**Behavioral**
- `class` => algorithms/control via inheritance
- `object` => algorithms/control via object groups
This course uses a problem-based learning methodology.

- Solving problems motivates your learning
- Lecturing is minimal and “on-demand” when requested by students
- This is better because
  - Learner actively engages the material
  - Deeper learning when learner motivates need for knowledge
  - More closely resembles true career situation
The students were very positive about this teaching approach.
Other than wanting a passive experience the instructor can help you overcome PBL negatives.

- Negatives expressed or perceived by students
  - *Thinking is hard*
  - *Making mistakes is discouraging*
  - *Not a passive sport anymore*
    - Identify needed knowledge
    - Initiate requests for additional guidance
  - *Don’t know enough to know what I don’t know*
  - *Not getting money’s worth from the instructor*
Success in this course requires a different strategy than for other courses.

- Keep all work moving forward
  - *Do not do questions and design serially!*
  - *Questions are due first but …*
    - Less time is needed to do questions
    - If you only work on the questions until they are due you will never have enough time for the design and implementation work.
    - Work questions and design/implementation together
- Seek feedback every class
- Bring up the struggles for discussion
- Ask for lectures if that is your learning style
Course is divided into units with both individual and team activities.

- **Individual activities**
  - *Unit 1 design activity*
  - *Unit 2, 3, and 4 individual questions*
  - *Unit 2, 3, and 4 quizzes*
  - *Mid-term and final design exam*
  - *Discussion participation*

- **Team activities for units 2, 3, and 4**
  - *Answers to questions*
  - *Design and implementation exercises*
Grading is divided into team and individual components by units.

<table>
<thead>
<tr>
<th>Component</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mid-Term Design</td>
<td>10</td>
</tr>
<tr>
<td>Final Exam</td>
<td>20</td>
</tr>
<tr>
<td>Unit 1 design problem</td>
<td>5</td>
</tr>
<tr>
<td>Unit quizzes (2 * 5)</td>
<td>10</td>
</tr>
<tr>
<td>Discussion participation</td>
<td>10</td>
</tr>
<tr>
<td>Unit questions (3 * 5)</td>
<td>15</td>
</tr>
<tr>
<td>Unit design/implementation exercises (3 * 10)</td>
<td>30</td>
</tr>
</tbody>
</table>
We start you off immediately thinking about the design of a software system.

- **This class**
  - *Individually create a design for the stated problem*
  - *Collaborate with others*

- **Next class**
  - *Submit individual design at start of class*
  - *Groups of students create a consensus design*
  - *Designs will be presented*
  - *Designs will be compared and contrasted*