SWEN 262

Engineering of Software Subsystems

General Course Information

Logistics

- During each week of the semester, SWEN 262 will meet 2-3 times for 50-75 minutes each time.
- Each class session will include some combination of the following:
  - A brief lecture covering new material (usually ~15 minutes)
  - An in-class assignment (individual or small team)
  - A discussion about a previous assignment
  - Project time
  - Presentations
- Many assignments are due at the start of class so that we can have a class discussion.
  - Please pay attention to due dates and times.

As much as possible, class time will be devoted to problem solving as individuals or on teams of 2-6 students.

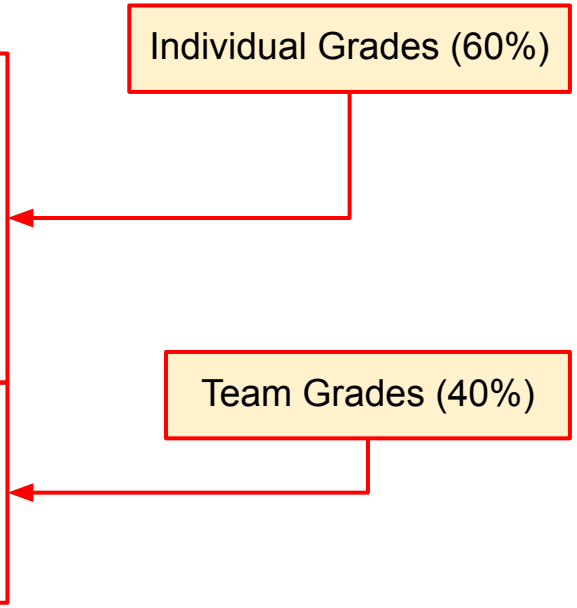
Grades

- A minimum grade of C- is required for Software Engineering majors to qualify for coop.
  - SE majors are also ***required*** to take the SE Co-op Seminar (SWEN-099) before co-op.
- Your instructor *may* consider rounding grades up.
  - For example a 79.6% may qualify a student for a B-.
  - This is up to the discretion of the individual instructor.
  - In general, grades will not be rounded up more than half a percentage point.
- Students that regularly attend class, respond to feedback, visit their instructor during office hours, and generally seem to *try* are more likely to benefit.

Grade	Percentage Range
A	at least a 93
A-	at least a 90
B+	at least an 87
B	at least an 83
B-	at least an 80
C+	at least a 77
C	at least a 73
C-	at least a 70
D	at least a 60
F	less than 60

Grading

- Activities - 10%
- Mini-Designs - 10%
- Midterm Exam 1 - 10%
- Midterm Exam 2 - 10%
- Final Exam - 20%
- Design Project R1 - 17.5%
- Design Project R2 - 17.5%
- Refactoring Project - 5%



Note that individual grades for team projects may be adjusted based on the instructor's feedback or feedback from other members of the team.

Class Activities

- Most class activities will require that you download a document, answer some questions, and upload your answers to a drop box on MyCourses.
- Each will be graded on a scale of 0-5.
  - 0 for no submission.
  - 1,2,3,4 (grader's discretion) based on partial/incorrect solutions.
  - 5 for a correct solution.
- This means that it will be hard to get a good class participation grade simply by submitting anything at all at the last minute.

Mmm...tastes like learning!

Individual Learning

This course is designed to give the learner every opportunity to, well, *learn*...

- In class activities (short answer or problem solving).
- Hands on mini-design exercises (refactorings).
- Team Projects - Apply your knowledge to large design problems.

It will also give your instructor a *lot* of data about how much you have learned and how well you can apply your knowledge.

- Part of the instructor's job is to evaluate your preparedness for a co-op.
  - Critical thinking.
  - Communication and writing skills.
  - Code mastery.
  - Ability to work on a team.
  - Design and engineering skill.

Please don't consider this class the last "check box" you need to get out into the big wide world.

Instructors can (and *have*) recommended that students that technically pass the course are not considered "ready" for co-op.

Instead, consider this a real opportunity to learn valuable skills for your career.

So, What Can I do to Get a Better Grade?

- The work.

Seriously, we get asked this question every semester, in every class, by at least one or two students, usually around week 12.

The most straightforward way to get a good grade in this class is to do the assignments, and **respond to feedback***.

Sometimes a small amount of extra credit (up to 2%) is offered near the end of the semester, but that is not guaranteed.

* “respond to feedback” means that, if you are given feedback about something you should do differently, you should actually...do it differently the next time.

Course Overview

Course Overview

- This course discusses 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.
  - No longer thinking strictly in code, or at the level of individual classes and objects.
- What to do when a pattern does not fit exactly.
  - Patterns are not code.
  - Patterns are *recipes* that can be followed to solve a problem that has been solved before.
  - Sometimes the recipe needs to be slightly modified to fit a specific problem.
  - Learn to evaluate the options and analyze the trade-offs.

Be wary of modifying a pattern too much.

This is a good sign that you may have picked the wrong pattern.

Beware the *Golden Hammer*...

Course Overview

At the code level, you are already familiar with some standard patterns.

Q: How would you walk through an array in Java?

```
for(int i=0; i<array.length; i++) {
  Object element = array[i];
  // do whatever you need to do with element i
}
```

Alternatively...

```
for(Object element: array) {
  // do whatever you need to do with the element
}
```

Course Overview

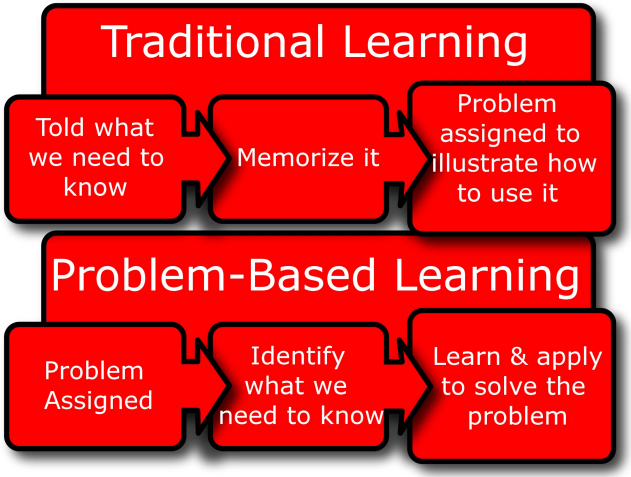
The level of discussion in this course is small subsystems of 3 to 10 classes.

- Higher than what you may have done before.
  - Not specific data structures
  - Not algorithmic approaches
  - Not *nose-to-the-screen* coding when all you think about is the current statement.
- But also *lower* than whole architecture systems or frameworks.
  - Not financial systems
  - Not air-traffic control
  - Not J2EE
  - Not Django or Spring
  - Not a view from 300 miles up.

Course Overview

This course uses a problem-based learning (PBL) methodology.

- Solving problems motivates your learning.
- Planned lectures are minimal (for the most part).
- This is better because:
  - The learner (*you*) actively engages in the material. *You* do it rather than listen to me explain how it is done.
  - Deeper learning occurs when the learner motivates the need for knowledge.
  - This more closely resembles a true career situation.



Problem-Based Learning provides you with the tools and resources you need to solve fuzzy problems without giving you the answers.

In many cases there is (many) more than one solution to a problem!

Course Overview

The instructor can help you overcome perceived PBL negatives.

- Thinking is hard.
- Making mistakes is discouraging.
- This is not a passive sport anymore.
  - ***You*** need to identify needed knowledge.
  - ***You*** need to initiate requests for additional guidance.
- “I don’t know enough to know what I don’t know.”
- Not getting money’s worth from the instructor.

Course Overview

Success in this course requires a different strategy than other courses.

- Keep work moving forward.
  - You are given lots of time to complete each assignment. Waiting until the last minute will affect your success.
  - In other news, the sky is blue.
- Seek feedback every class.
  - Your instructor will *attempt* to guide you to make your work *less worse*.
- Bring up struggles for discussion.
  - I can help you get through the rough patches.
  - Other students not nearly as brave as you are will benefit from your bravery.

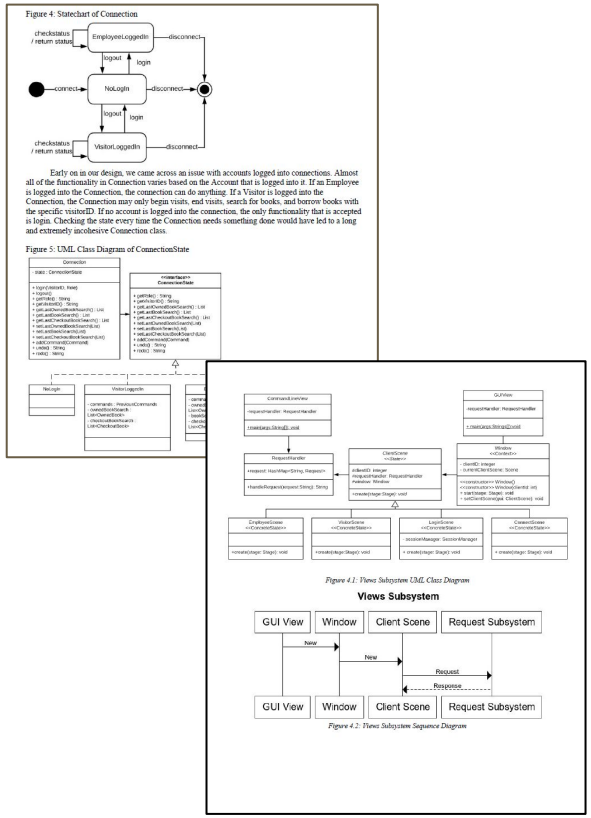
Course Overview

To move to the next level of design, you must know the principles that underlie “good” designs.

- All engineering is based on principles that have been learned over time.
  - *Discovered* rather than *invented* through the application of competent software engineering.
  - Used in many applications.
  - Sometimes through failure!

Course Overview

- Most problems do not have a single, perfect solution that everyone will agree on.
  - The larger the problem, the less likely that there is a single solution to solve it.
- Throughout this semester, students will create different designs to solve the same problem.
  - Many will be terrible *at first*.
  - But many may also solve the same problem in different, equally valid ways.
- Design Patterns are about:
  - Recognizing a category of problem that has been solved before and learning to apply the pattern(s) to solve this new iteration.
  - Learning to speak a language that helps communicate your ideas quickly and efficiently.



Words to ~~Live~~ Design By

- Use language of requirements to justify patterns (clues/breadcrumbs)
  - You will (hopefully) learn to recognize certain key phrases as identifying a requirement that is suitable for one (or maybe more!) patterns.
- Use design principles to justify design choices
  - Whenever you make decisions about competing design choices, make sure that those decisions are based firmly in design principles.
  - Often what “feels right” *is* right but you need to determine why and be able to explain it.

Functional requirement 4.a states that *“the user must be notified when an bid has been made on their sale item.”* We chose the Observer pattern here because its intent is to automatically notify a dependent when the status of its subject changes.

While the Mediator pattern may sometimes appear to be a blob, it in fact reduces the overall coupling in the system by replacing many-to-many coupling with one-to-many. In addition, the overall cohesion of the system is improved by moving the control logic out of the colleagues and into the mediator.

Professional Responsibility

- Lastly, your role in this course is to demonstrate that you are prepared to work and act like a professional software engineer.
  - Show up on time.
  - Stay until the end.
  - Meet your commitments to your instructor, your team, and yourself.
  - Notify your instructor and especially your team if you can't meet your obligations for any reason (illness, emergency, etc.).
- Your treatment of your instructor and your peers may affect your grade!

It is ultimately ***your*** responsibility to make sure that you complete your work and submit it on time.

It is course policy ***not*** to extend submission deadlines except for extenuating circumstances.

Note that *"I forgot"* is not an extenuating circumstance.

Assignment submissions are ***never*** accepted through email and will be deleted on sight.