1. CSC 276 Object-oriented Software Design

Section 20

The student and instructor will meet once per week for no more than 60 minutes.

2. Instructor

Xiang Huang
239B Reilly Hall
huangxx@lemoyne.edu

Office Hours (RH 239B)

2:30-4:00 PM on Monday, Wednesday; Or by appointment.

. Course Description & Rationale

College Catalog Description

This course deals with the general topic of object-oriented software design. Design strategies (e.g., compositional) and concepts (e.g., functional independence) are discussed in the context of a software design model that contains four elements- architecture, data, interfaces, and components. Different object-oriented software design techniques (e.g., UML), software design metrics (e.g., coupling), and software quality assurance techniques (e.g., review) are discussed and applied to software designs. The course will also introduce human-computer interaction, information models and database systems. Each student will produce small and medium-sized design models and will produce one medium-sized design model and a prototype implementation.

Prerequisite

CSC 176 or permission of the program director.

Rationale

This course emphasizes abstraction as a critical thought process. The students' knowledge learned in 175 and 176, particularly the definition and use of classes, is extended to encompass a more abstract (and a more complete) way of describing elements found within a software product. This course completes the foundational software engineering knowledge requisite for a computer science major. This course covers the following knowledge areas and topics in the latest computer science curriculum guidelines.

- Human Computer Interaction (HCI): New Interactive Technologies, and Programming Interactive Systems.
- Information Assurance and Security (IAS): Defensive Programming, Foundational Concepts in Security, Principles of Secure Design, Secure Software Engineering, and Threats and Attacks.
- Information Management (IM): Data Modeling.
- Programming Languages (PL): Event-driven and Reactive Programming, and Object-oriented Programming.
- Software Development Fundamentals (SDF): Development Methods, and Fundamental Data Structures.
- Software Engineering (SE): Software Construction, Software Design, Software Project Management, Software Verification and Validation, and Tools and Environments.
- System Fundamentals (SF): Cross-layer Communications, and State and State Machines.

4. Course Materials & Resources

Text Books

The instructor has a third draft of a text book that will be used in this course.

Resources

The following identifies additional resources that may be used for this course:

- Materials will be posted on the Canvas course pages throughout the semester.

5. Learning Goals and Objectives

Program Learning Goals

Le Moyne College expects its computer science graduates to:

1. Use critical thinking skills, problem solving techniques, and abstraction to develop computational solutions while understanding how theory and practice influence each other and appreciating the value of good design.
2. Apply the knowledge they have learned to solve real world problems, realize that there are multiple solutions to a given problem and be able to assess the benefits and weaknesses of different approaches.
3. Be prepared for the rapid pace of advances in the computing field and for continued growth as a computing professional.
4. Be able to communicate their knowledge to others in an ethically responsible manner and be prepared to work individually or in a collaborative environment.

Course Learning Objectives

Upon completion of this course, the student shall be able to:

- Evaluate human-computer interaction (HCI) in existing applications.
- Design and develop a text-based user interface for a simple application.
- Implement HCI using a model-view-controller framework.
- Develop logical and physical data models.
- Develop text-based XML files for a simple application.
- Embed Document Object Model language statements to use an XML file within a simple application.
- Understand and apply software quality assurance techniques to evaluate design models.
- Understand testing fundamentals and apply test cases to a software implementation.
- Develop design artifacts that describe elements of a simple application.
- Develop effective technical documentation that consolidates design models into a coherent design artifact.
- Use web-based resources to support learning.
- Use digital library resources to support learning.

6. Assessment & Evaluation of Learning

Assessment

A student actively participates in the assessment of their learning. Assessment activities include:

- Student discussions of course information in formal and informal settings.
- Reviewing instructor feedback to help gauge which learning activities are most helpful.
- Being honest with yourself about what your strengths and weaknesses are as it relates to learning the course materials.
- Being open to new ideas regarding the best way(s) for you to learn this material.

In addition, an evaluation approach called *mastery learning with contract grading* will be used by the instructor to evaluate your learning. While the instructor will use this evaluation approach to grade your assignments, ***a student should use this to identify their strengths and weaknesses***.

Evaluation

While students shall actively assess their learning, the evaluation of a students' learning is done solely by the instructor. Your evaluation, which results in a grade, shall include the following elements:

Assignments	80%	All assignments are evaluated using ***mastery learning with contract grading*** described below.
Participation	20%	Attendance and discussions at the weekly meetings.

Final grades will be assigned according to the following scale. Grades are rounded to determine the final grade, so 92.5% or higher is an A (and so on).

	A (4.0): 93-100%	A- (3.7): 90-92%
B+ (3.3): 87-89%	B (3.0): 83-86%	B- (2.7): 80-82%
C+ (2.3): 77-79%	C (2.0): 73-76%	C- (1.7): 70-72%
	D (1.0): 60-69%	
	F (0.0): 0-59%	

Evaluation Terms (mastery learning, contract grading)

A description of each term used in the title of this section is found below. The ***words in bold italics*** highlight the differences in these two evaluation approaches.

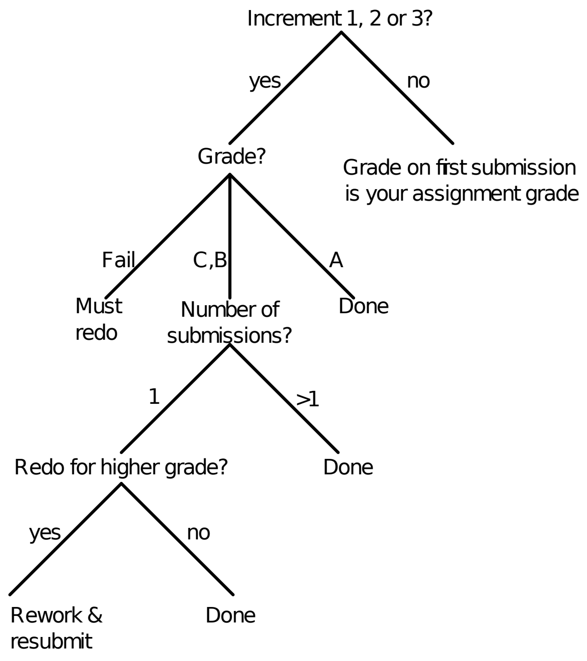
- Mastery learning — An evaluation approach that requires a student to demonstrate a mastery of the concepts before being allowed to continue to the next assignment.
  1. The instructor defines a minimum level of ***mastery*** that results in a passing grade (e.g., an A-level grade).
  2. Any grade less than the minimum level results in the student failing the assignment.
  3. A student must redo a failed assignment until they've passed the assignment by reaching or exceeding the minimum level of ***mastery***.

- Contract grading — An evaluation approach that allows a student to choose the grade they earn for an assignment while ensuring that a minimum level of competency has been demonstrated.
  1. The instructor defines a minimum level of ***competence*** that results in a passing grade (e.g., a C-level grade).
  2. Any grade less than the minimum level results in the student failing the assignment.
  3. A student must redo a failed assignment until they've passed the assignment by reaching or exceeding the minimum level of ***competency***.
  4. ***A student may also redo a passed assignment in order to improve their grade on the assignment***.

Evaluation using Mastery Learning with Contract Grading

An instructor may choose to evaluate student work using only mastery learning or using only contract grading. ***In this course, the instructor is combining mastery learning with contract grading***. The following describes how these two evaluation approaches shall be applied in this course:

- Each type of software artifact you produce will have a rubric.
- Each rubric will describe the evaluation criteria for achieving a C, a B, and an A.
- You must earn at least a C to pass the assignment.
- ***For increments 1 through 3:***
  - You have 2 attempts to pass an assignment.
  - When you fail an assignment on the first attempt, the grade you earn on your redo (i.e., your second attempt) is your grade for this assignment.
  - When you pass an assignment on the first attempt, you may redo the assignment once to improve your grade.
- ***For increments 4 through 7:***
  - Your grade on your first submission is your grade for this assignment.
- Your final grade for each assignment is the highest grade that you've earned.
- The decision tree to the right explains the choices you will need to make.



Evaluation using Software Artifact Rubrics

Each type of software artifact listed below will have a rubric that will describe the evaluation criteria a student much meet in order to achieve the corresponding grade.

- Plan
- Human-computer interaction
- Data model
- Design
- Code

In all of these rubrics:

- The acceptable (C) grade represents the minimum level of ***competency***.
- The best (A) grade represents the minimum level of ***mastery***.

Computing a numeric grade for an assignment is done as follows:

1. Each rubric includes many evaluation criteria, describing how you can achieve a C, a B, and an A. Thus, using one rubric generates many letter grades for your assignment.
2. An assignment may use many rubrics to evaluate your work.
3. Each letter grade from rubric evaluation criteria is converted to a numeric grade, using the table below.

Unacceptable	Acceptable (C)	Better (B)	Best (A)
50%	75%	85%	100%

4. All of the numeric grades are then averaged to determine your assignment grade.
5. Determining whether you have passed or failed an assignment is determined using the table found in the ***Evaluation*** section above. Note that, for purposes of the decision tree described above, a grade of C or better (i.e., 72.5% or higher) is a passing grade.

How much time should I devote to this class?

Many students are able to earn a B+ or higher in this course. This is for three reasons:

- The student takes advantage of the *Mastery Learning with Contract Grading* described above.
- The student pays attention to the rubric evaluation criteria and the instructor's grading comments.
- The student is willing to spend the hours necessary to improve their software designs. Prior students have indicated that they've spent anywhere from 5-20 hours per increment to improve their designs.

7. Course Procedures & Policies

Assignments

Each assignment has a due date and time (e.g., Tuesday, August 30, 2011 by end-of-day i.e., 11:59:59 PM). An assignment solution submitted after the due date/time shall have a late penalty applied to the grade. Note that there is no due date/time associated with when you submit your second attempt for increments 1 through 3.

Based on the following notation, the formula below explains the assignment late penalty policy.

- **`DDT`** the Due Date/Time.
- **`SDT`** the Submission Date/Time.
- **`+n`** the nth class session after DDT.

Formula	Late Penalty
`DDT+n-1 < SDT <= DDT+n`	-5n%

When n=1, the assignment is one day late and a 5% late penalty is applied. When n=2, the assignment is two days late and a 10% late penalty is applied. And so on. The maximum late penalty is 50%; any assignment submitted after ten days receives the maximum late penalty.

Any College announcement indicating that classes have been cancelled affects the above policy *only if* the day classes were cancelled is also the DDT. In this case see Error: Reference source not found (below) for details.

Attendance

Attendance is crucial. I would strongly suggest that you not miss any weekly meeting. As described under ***Grading***, participation is twenty percent of your final grade.

Backups

You should use your H: drive to store all of the assignment files for this course. A benefit to doing this is that the Information Technology Office does a backup of your H: drive each evening. Should you accidentally delete an assignment file, the IT help desk can help you get that file back (assuming the file has existed for at least 24 hours).

Cancelled Classes

In the event that the college cancels classes, the following conditions apply:

- If an assignment is due, it is still due on the date classes were cancelled.

E-mail Communication

When the instructor sends an e-mail out to the class, the instructor will use your Le Moyne e-mail address. Should you send an e-mail to the instructor; the instructor will reply using the e-mail address you used to send the e-mail.

When sending the instructor an e-mail using a non-Le Moyne e-mail address, please identify yourself by name. It is sometimes very difficult to determine who you are based on an e-mail address (e.g., catboo@yahoo.com).

Pandemic Preparedness
In the event that the outbreak of a virus that is widespread during the spring semester, the following academic initiatives will be undertaken.

- Courses will *not* be canceled; the College will *not* shutdown.
- Material will be presented via Canvas for those students that are unable to attend class.
- If the instructor becomes ill with a virus, then the instructor will post instructions via Canvas as to how the course shall proceed, and to indicate when he'll restart classroom-based instruction.

8. Course Outline
The following provides an approximate schedule for this course.

Schedule	Chapters		Assignments (due date)
Weeks 1-2	1	Course Introduction	
	2	Introduction to Software Design	
	3	Program Design Criteria and Simple Design Models	Increment 1 (Feb 4)
	4-OOP	Case Study: Using Program Design Criteria & Simple OOP Models	
	5	Program Design and Performance	
	6-OOP	Case Study: Considering Performance	
	7	Program Design and Security	
	8-OOP	Case Study: Considering Performance	
Weeks 3-4	9	Characteristics of Good Software Design	Increment 2 (Feb 13)
	10-OOD	Case Study: Transition to Software Design	
	11	Introduction to Model-View-Controller	
	12-OOD	Case Study: Model-View-Controller	
Weeks 5-7	13	Introduction to HCI Design	Increment 3 (Feb 25)
	14-OOD	Case Study: TUI	Increment Planning (Mar 8)
	15	MVC: TUI versus GUI	
	16-OOD	Case Study: GUI	
	17	Is Your Design Clear, Concise, and Complete?	
		Spring break	
Weeks 8-15	18	Software Design & Security	Increment 4 (Mar 25)
	19-OOD	Case Study: More Security Requirements	
	20	Introduction to Design Patterns	Increment 5 (Apr 8)
	21-OOD	Case Study: Design Patterns	
		Easter break	
	22	Modeling Persistent Data	Increment 6 (Apr 22)
	23	Persistent Data Storage	
	24-OOD	Case Study: Persistent Data Storage	
	25	Software Design Document	Increment 7 (May 6)
	26	What's Next?	
		Course/instructor evaluations	
		Distribute take-home final exam	

9. Accommodations

Students with Disabilities
In coordination with the disability support services (DSS), reasonable accommodations are provided for qualified students with disabilities. Please register with the disability DSS office for disability verification

and determination of reasonable accommodations. You can either stop by the DSS (Library, 1st floor) or call (445-4597 voice mail, or 445-4104 TDD) to make an appointment with the staff. Please meet with the instructor to review your accommodation form and discuss your needs as warranted.

Observance of Religious Holidays
As provided by New York State Education Law, a student has the right to miss a class event because of his or her religious beliefs. Any student that wishes to apply this law must notify the instructor at least two weeks in advance of missing the class event.

10. Academic Standards

Cheating Versus Collaboration
You are encouraged to discuss your ideas and approaches to assignments with your classmates, tutors, and the instructor. Any solution that you submit should be written by you, or by your group/team for group-based assignments. You are expected to fully understand everything submitted, even when submitting a group-based solution. (Exception for the Senior capstone course: you will need to understand enough of your group members' contributions to create an integrated solution; you are not responsible for understanding all of the details of your group members' contributions.) If asked to do so, the instructor expects that a student can explain any part of their solution that the instructor deems suspicious or confusing. This includes explaining any line of code or design feature. When a student is unable to explain their solution this will be deemed an occurrence of cheating. Letting others copy your work constitutes cheating and is subject to the same penalties described in section ***6 Assessment & Evaluation of Learning***. Depending on the severity and frequency of cheating, cheating may result in a grade of zero on the assignment/quiz/exam, formal notification to the Dean, and/or an F in the course.

Motivation for Cheating Policy
Below are some of the motivations for why the CS program has developed this cheating policy.

- Employers look to the CS degree as an indication that a student can write software. When a company interviews a freshly graduated candidate for a software developer position, and the company determines that the candidate cannot write code, the company will be cautious about other students from the same school. If the company found that more than one candidate was unqualified in the technical interview, this company will likely start screening out candidates from that school. Thus, students that graduate with a CS degree without learning the skills hurt everyone else at their school.
- The CS faculty believe that the most effective way of learning software design, implementation, testing and debugging is to *take a hands on approach by grappling with concepts and being actively involved in code writing*, *design and problem solving*. This learning style is described as ***active play***, and involves *being creative by experimenting with various approaches to solving a problem*.
- Discussions are an important aspect of collaboration. This is true in both college and professional environments. The CS faculty *encourage you to discuss the structure and rationale of your solution with your peers*.
- Note that, many employers' policies encourage software code reuse. However, 99.9% of the time when a professional reuses code, the existing code requires modification as it either solves a slightly different problem or only a part of the new problem. Thus a professional would still need to (1) formulate an idea of how they want to solve their problem; (2) be able to understand the code they are copying; and (3) be able to change it only where needed. The ability to accurately, effectively, and efficiently reuse portions of existing code in a project is a skill learned though experience and active play. These skills are not built by copying parts of the solution of the same problem from other students.

- *In upper level CSC courses, programming is used to help* ***reinforce fundamental concepts***. Note that programming is a great example of ***active play***, which involves *being creative by experimenting with various approaches to solving a problem*. The CS faculty expect that upper level students are able to apply their programming skills to help demonstrate the concepts being covered.