ILLINOIS INSTITUTE OF TECHNOLOGY

Transforming Lives. Inventing the Future.

www.iit.edu

SOFTWARE ENGINEERING

CS 487

Dennis Hood

Computer Science

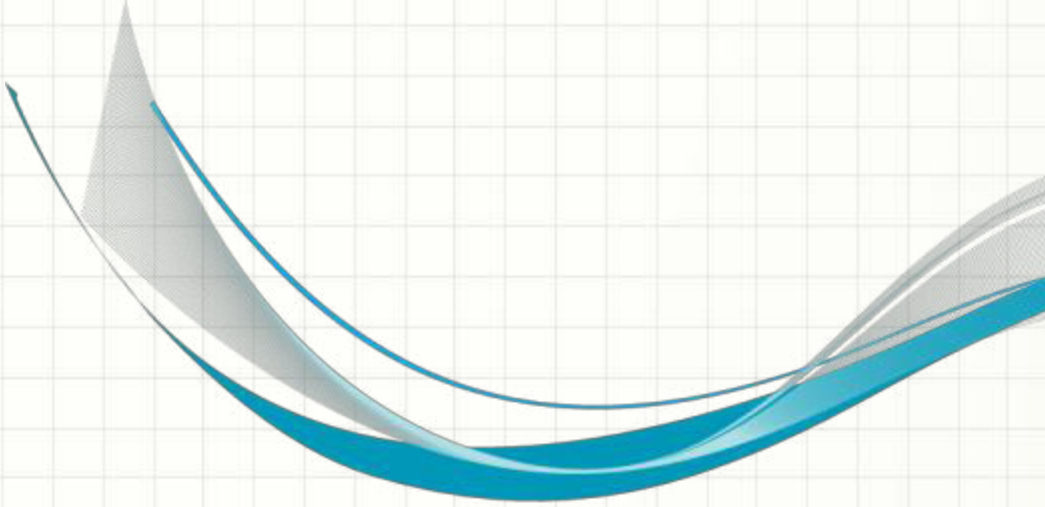
Summer ‘19

Homework #4

- Information Modeling
  - You are designing World Cup penalty kick simulator
- Deliverable – analysis and design of information
  - Briefly describe the penalty kick scenario: focusing on 5 objects – kicker, goalie, ball, net, and scoreboard
  - Describe each object by listing its attributes (including data type) and methods (including parameters and return values)
  - Use pseudo-code to depict the scenario – be sure your code supports the interactions between your objects
  - Enhance the code to simulate a championship tie-breaker scenario (2 teams, 5+ rounds)
- Please submit by 8/1/19

Lesson Overview

- Specification: Modeling the Object Space
- Reading
  - Ch. 12 – Safety Engineering
  - Ch. 13 – Security Engineering
  - Ch. 14 – Resilience Engineering
- Objectives
  - Revisit the threat of ambiguity and the importance of clear, concise communication in successful software development
  - Examine object-oriented software engineering with respect to previous topics (e.g., modeling with state-transition diagrams, reuse, testing, evolution, etc.)



Lecture 7
Object-oriented Software Engineering

Object Orientation

- Developing software solutions involves two significant translations:
  - The user's environment -> a system design which models it
  - The system design -> an executable application
- Object-oriented analysis identifies the entities, referred to as objects, that make up the user's environment
- Object-oriented design then develops a model of these objects including their attributes and behaviors
  - This forms the basis of the system's design
- Object-oriented programming implements this design as an application (using a language such as Java)

Analysis

- In analysis, the software engineer works to understand the perspective of the user
- The goal is to capture what the application will be required to do and how it should do it
- The end result historically has been a requirements specification
- In object-oriented analysis (OOA), we seek to capture the entities which make up the user's world and their interactions

Design

- In design, the software engineer works to structure a solution system which will meet the stated requirements
- The goal is to create an architecture which will not only satisfy the requirements, but will do so with efficiency and facilitate future growth (increases in number of users, changes in other system components such as OS, etc.)
- The end result historically has been a set of system models, with each capturing a perspective (UI, information architecture, workflow, state transitions, etc.)
- Object-oriented design (OOD), uses diagrams and modeling languages similar to previous methodologies

Unified Modeling Language (UML)

- A system's design is complex and abstract
- Expressing it is difficult using "human" language (e.g., English) which presents a significant risk of misinterpretation
- As software engineering has evolved as a discipline, various models have emerged to capture different system perspectives
- The Unified Modeling Language (UML) tied together several of these models with a supporting language to allow engineers to specify design with greater clarity

Classes

- The nouns and noun phrases found in the user's description of their environment indicate entities which must "exist" in the application in order for the application to be both logical and useful
- In object-oriented development, classes are templates which capture these entities
- Objects are instances of a class which can then be made to behave in a manner which simulates the user's operating environment

Attributes

- The descriptive terms used by users to further explain the entities of their environment can be incorporated into class definitions as attributes
- Attributes should be specified in a constrained way with respect to their purpose and limitations (type)
- Access to attributes can be restricted in to hide detail, protect privacy, and maintain integrity

Behavior

- The action words (verbs) used by the user in describing their environment indicate the behavior of the entities
- Methods are included in class definitions to provide the means of simulating the behavior in instantiated objects
- UML supports the modeling of user interactions (Use Cases) with objects as well as the flow of behavior (Sequence Diagrams)

Supporting Methods

- Class definitions must include methods which support behavior and thereby allow objects to more completely and accurately simulate entities
  - Constructors which instantiate objects by assigning values
  - Accessors which provide the ability to read the value of an object's attributes
  - Mutators which provide the ability to altering an object's attributes in an acceptable manner

Things to Model

- Context Models
  - The relationships among system components
  - The “big picture” view of the user’s environment
- Use Cases
  - A capture of user interactions with system components
- Sequence Diagrams
  - The flow of interactions and transactions
- State-Transition Diagrams
  - A capture of the states the environment can be in and the catalysts which cause the environment to change
  - Helps to identify and capture exceptions

Encapsulation

- Instance variables are usually declared to be *private*, which means that they cannot be accessed directly outside the class
- Users (clients) of the class can only reference the private data of an object by calling methods of the class
- In this way, the methods provide a protective shell around the data known as *encapsulation*
- Benefit: the class' methods can ensure that the object data is always valid

Reusability

- Reuse has tremendous potential benefits
  - Designing, coding, debugging, etc. are difficult, time consuming, expensive, risky, etc.
  - Reuse gives us more return on that investment
- Impact on new development
  - The time required to build a new application will be shortened if we can reuse existing components
  - Less time means lower cost
  - The code will be more reliable since the reused components have already been tested and used
  - Working with already designed classes helps with the understanding of requirements and design