CS 460 Software Engineering

Fall 2020

Instructor:	Gruia-Catalin Roman
Credit:	3 units
Prerequisites:	CS 351
Time:	Tu. & Th. 2:00 – 3:15 pm
Location:	remote class

State-of-the-art, industrially tested, specification and design techniques are presented, illustrated on small examples, and applied to realistic problems. The objective of the course is to develop an understanding of the technical and organizational complexities involved in software development and to teach key concepts and techniques used to manage these complexities. The students are required to design and build systems through team effort. Emphasis is placed on emulating the realities of an industrial organization. The classes are conducted in an intensive workshop atmosphere. The projects cover the principal system development life-cycle phases from requirements analysis, to software design, and to final implementation. Issues relating to human factors, dependability, performance, operating costs, maintainability and many others are addressed and resolved in a reasonable manner.

Desirable Entry Level Skills

Basic requirements

- programming proficiency (100-300 lines of dependable code) in Java or C++
- operating systems principles (drivers, mutual exclusion, scheduling)
- database systems
- socket-based communication
- specification, design, implementation distinctions
- basic technical writing abilities
- low-level code documentation (headers, in-line assertions)
- sets, functions, predicates, state machines
- semantic models

Elementary software engineering skills

Basic functional design

- data structures
  - high-level description using words and pictures
  - initial state
  - commonly used operations
- procedure
  - high-level algorithm specification
  - pseudocode (10/1 ratio of code to text)
- procedural abstraction
  - axiomatic specification (pair of assertions)
  - operational specification (pseudocode)
  - functional decomposition and recursion

Object-oriented concepts

- simple objects
  - proper encapsulation of data structures and devices
  - clean abstraction (application-oriented)
  - specification: abstract state, operations over the abstract state
  - design documentation: data structures and procedures

Learning Outcomes

Software engineering skills to be acquired

Object-oriented design

- composite objects
- design diagrams
- design strategies
- design patterns
- accommodating languages (e.g., C++ vs. Java)
- coping with design constraints (e.g., off-the-shelf packages)
- factoring performance
- generalization (classes and templates)

System level design

- design rules
- processing and data allocation
- design and specification of system interfaces
- distributed algorithms design and verification
- performance evaluation and estimation
- feasibility analysis

System level requirements

- elicitation strategies
- models
- documentation methods
- validation
- formal specification
- human interface design

Technical management of large software projects

- life-cycle
- software development strategies (macro-level)
- technical reviews
- written and oral communication
- planning and commitments
- group and corporate dynamics
- system integration