CSCE 240: Introduction to Software Engineering

1. Course number and name: CSCE 240: Introduction to Software Engineering

2. Credit: 3-hrs; Contact: 3 lectures of 50 minutes each or 2 lectures of 75 minutes each per week

3. Instructor: Fall 2010: Homayoun Valafar and Pahal Kamlesh Dalal (GS)
   Spring 2011: Homayoun Valafar

4. Text book: Walter Savitch, *Absolute C++,* 4th edition, Addison Wesley, 2009.

5. Specific course information
   a. Catalog description: Fundamentals of software design and development; software implementation strategies; object-oriented design techniques; ethics in software development.
   b. Prerequisites: CSCE 215, grade of C or better in CSCE 146
   c. Required in all curricula

6. Specific goals for the course
   a. Specific outcomes of instruction:
      - Independently design and implement C++ programs in a Unix environment.
      - Demonstrate mastery of pointers, iterators, memory management including object creation and destruction, and parameter passing in C++.
      - Demonstrate mastery of object oriented programming concepts including: inheritance, polymorphism, operator overloading, template functions and classes, and the use of STL containers.
      - Develop object oriented models using UML
      - Engage in program design and implementation in a team environment.
      - Use a source control tool in a team environment.

   b. Relation of course outcomes to Student Outcomes: CE: see page 2; CS & CIS: see page 3

7. Topics covered and approximate weight (14 weeks, 3 hours/week, 42 hours total)
   1. Unix Programming Environment: Unix tools, C preprocessor, Make, Shell, I/O redirection, debugging.
   2. Pointers: Pointer manipulation, functions and function pointers, virtual functions.
   3. Basic class management: constructors, destructors, data hiding, container classes.
   4. Memory management: object creation and destruction, memory leak.
   5. Advanced C++ features: operator overloading, iteration, special containers, inheritance, code reuse, multiple inheritance, virtual functions, polymorphism, templates, template libraries.

6. Introduction to UML and object oriented modeling: use-case models, object identification, specifying static behavior, activity diagrams, collaboration diagrams and sequence diagrams, specifying relationships: generalization/specialization, aggregation, associations including multiplicity and roles, dynamic behavior using state diagrams.
7. Introduction to Source Control and Distributed Source Control, for example, using git.

c.

Computer Engineering

Relation of Course Outcomes to EAC Student Outcomes*

Course Outcomes (CE)	**Student Outcomes**											
	(a) apply knowledge of mathematics, science, and engineering	(b) design and conduct experiments, … interpret data	(c) design a system, component, or process to meet desired needs …	(d) function on multidisciplinary teams	(e) identify, formulate, and solve engineering problems	(f) an understanding of professional and ethical responsibility	(g) communicate effectively	(h) the broad education to understand the impact of engineering solutions …	(i) a recognition of the need for, and an ability to engage in life-long learning	(j) a knowledge of contemporary issues	(k) use the techniques, skills, and modern engineering tools ….	(CE) demonstrate knowledge of discrete mathematics [CE]
Criteria	a	b	c	d	e	f	g	h	i	j	k	CE
1. Demonstrate mastery of object oriented programming concepts: inheritance, polymorphism, and operator overloading.												
2. Demonstrate mastery of pointers, iterators, memory management including object creation and destruction, and parameter passing in C++ and Java.												

3. Develop C++ code in a Unix Environment utilizing the C preprocessor, the debugger (gdb), make, source code revision systems (sccs), utilities such as those for transferring files to and from Unix and Windows.												
4. Demonstrate mastery of template functions and classes; understand underlying implementation of major containers in the STL.												
5. Develop object oriented models using UML and the Unix Programming Environment: Unix tools, C preprocessor, Make, Shell, I/O redirection, debugging.												

* 3 = major contributor, 2 = moderate contributor, 1 = minor contributor; blank if not related

d.

Computer Science & Computer Information Systems

Relation of Course Outcomes to CAC Student Outcomes*

Course Outcomes (CS & CIS)	**Student Outcomes**											
	All									**CS**		**CIS**
	(a) apply knowledge of computing and mathematics appropriate to the discipline	(b) analyze a problem, and identify and define the computing requirements …	c) design, implement, and evaluate a computer-based system, …	(d) function effectively on teams to accomplish a common goal	(e) An understanding of professional, ethical, legal, … responsibilities	(f) communicate effectively with a range of audiences	(g) analyze the local and global impact of computing on … society	(h) Recognition of the need for … continuing professional development	(i) current techniques, skills, and tools necessary for computing practice	(j) apply mathematical foundations, algorithmic principles, and CS theory …	(k) apply design and development principles	(j) An understand processes that support the information systems environment.
Criteria	a	b	c	d	e	f	g	h	i	j	k	j
1. Demonstrate mastery of object oriented programming concepts: inheritance, polymorphism, and operator overloading.												
2. Demonstrate mastery of pointers, iterators, memory management including object creation and destruction, and parameter passing in C++ and Java.												

3. Develop C++ code in a Unix Environment utilizing the C preprocessor, the debugger (gdb), make, source code revision systems (sccs), utilities such as those for transferring files to and from Unix and Windows.												
4. Demonstrate mastery of template functions and classes; understand underlying implementation of major containers in the STL.												
5. Develop object oriented models using UML and the Unix Programming Environment: Unix tools, C preprocessor, Make, Shell, I/O redirection, debugging.												

* 3 = major contributor, 2 = moderate contributor, 1 = minor contributor; blank if not related