Concordia University
Department of Computer Science and Software Engineering

SOEN 341 — Software Process
Course Outline — Fall 2008 — Section H

Instructor: Greg Butler (gregb@cse)
Tutor: Bahman Zamani (b_zamani@cse)

Web page: http://users.encs.concordia.ca/~gregb/home/soen341-f2008.html

Introduction

The purpose of SOEN 341 is to introduce the basic concepts of industrial software development, especially the various approaches to the software development process, and to prepare you for subsequent software engineering courses such as:
SOEN 342 — Software Requirements and Specifications
SOEN 343 — Software Design
SOEN 384 — Management and Quality Control in Software Development

Prerequisite Knowledge

You should have had some previous experience in programming, preferably object-oriented programming using C++ or Java (COMP 249 or equivalent), some knowledge of data structures and algorithms (COMP 352 or equivalent), and some knowledge of the principles of technical documentation (ENCS 282 or SOEN 282 or equivalent).

Textbook:Craig Larman. *Agile and Iterative Development: A Manager's Guide.* Addison-Wesley. ISBN 0-13-111155-8, 2003.

Course Outline

Need for a software process. Software processes and life cycles. Models of systems and software. Process modeling. Software development tools. Process components: requirements, analysis, design, implementation, testing. Project management and quality control. The course will discuss different alternative processes for software development, including the Waterfall model, iterative/incremental models, the Rational Unified Process, and Extreme Programming. A complete list of the lecture topics, along with course notes, is available on the course web page.

Lectures

Week 1: introduction, development phases

Week 2: software process, extreme programming

Week 3: agile software development, build-and-fix model, waterfall model, incremental models

Week 4: prototyping, spiral model

Week 5: unified process

Week 6: unified process

Week 7: requirements, use case approach

Week 8: specifications

Week 9: design, architectural design, conceptual design, module interfaces, detailed design

Week 10: testing, testing phases, unit testing, integration testing, system testing

Week 11: implementation, coding standards

Week 12: [slack for catch-up in schedule]

Week 13: capability maturity model, CMM process areas

Computing Facilities

You will do practical work in a computer laboratory equipped with PCs that run Windows and Linux operating systems. Programming environments for C++ and Java are available on both platforms. The computer laboratory assigned to this course is H-817. It will be booked at the same hours as your tutorial, as some of the tutorials might be held in the laboratory for the purpose of tutoring on software tools to be used in the project. Even though you are allowed to use other resources for the completion of the project, note that the final project demonstration will have to occur on the downtown campus.

Project

The major practical component of the course will be the development of an object-oriented program designed and implemented by a team. Each team will consist of approximately six to nine students. Each team will implement an application whose description will be provided during lecture and tutorial time in the first week of classes. It is very important to understand that the role of this project is to demonstrate the importance and the application of a software process. It is not to be approached as a regular programming project.

Evaluation

Quizzes (3)	10% + 15% + 15%	=	40%
Homework Assignments (3)	5% + 5% + 5%	=	15%
Team Project Deliverable 0 (team information)			0%
Team Project Deliverable 1 (incr. 1 + scope + plan)	(code: 3%) + (document: 7%)	=	10%
Team Project Deliverable 2 (incr. 2 + design)	(code: 3%) + (document: 7%)	=	10%
Team Project Deliverable 3 (incr. 3 + testing)	(code: 10%) + (document: 5%)	=	15%
Team Project Deliverable 4 (final project demo.)			5%
Individual Project Report			5%

Quizzes (10%+15%+15%) and assignments are individual work, and the project is team work, where each team member shares the exact same grade as his/her team mates. The three homework assignments (5%+5%+5%) will be primarily theoretical and designed as exercises to help

you prepare for the quizzes. The project is providing the practical component in the form of three project deliverables including an incremental delivery of the system (10%+10%+15%) and a final project demonstration (5%). Finally, a short individual project report (5%) is required at the end of the project to enquire on your experience as an individual in the project. Late assignments or project deliverables will be assessed a 50% penalty for each late working day. The exact schedule for assignment due dates and quizzes are available on the web page.

Project

The project is the game of Battleship Solitaire (http://en.wikipedia.org/wiki/Battleship_(puzzle)). The warships are located on a 10-by-10 grid. There is one battleship of four squares, two cruisers of three squares, three destroyers of two squares, and four submarines of one square. Each ship occupies a number of contiguous squares on the grid, arranged either horizontally or vertically. The boats are placed so that no boat touches any other boat, not even diagonally.

The goal of the puzzle is to discover where the ships are located. The player can ask two kinds of questions: (1) pick a square, and (2) pick a row or column.

For question (1) the computer answers: (a) *sub*; (b) *water*; (c) *ship* followed by *north*, *south*, *east*, *west*, or *middle*.

For question (2) the computer returns a count of the number of squares in the row or column that are occupied by ships.

There are three programs that make the complete application:

1. An interactive game player where the user can select a game and play it.
2. A game generator which returns a random game.
3. An advisor to guide the user playing a game.

The project is an incremental project involving three versions of these programs:

1. A minimal version which provides basic functionality and a workable interface.
2. A version which incorporates at least one difficult feature, such as infinite undo and redo, real game generation, and encapsulated strategies.
3. A usable, robust, tested version providing the most important features that time allows.

For the third and last version, each group should also integrate the three programs into one game application. It is suggested that team members take the roles of Implementor, Tester, and Documenter, and share the role of devising the requirements and the design at all times.

It is required that team members rotate to a new program for each version, and rotate to a new role for each version, so that by the end of the project you will have worked on each program and performed each role.