***CMPT 470/816 – Advanced Software Engineering***

***Course Outline***

**Meeting Times**

Lectures: Tuesday & Thursday at 10:00-11:20 AM in Thorv 205A

**Instructors**

Class Instructor: Chanchal Roy, Thorvaldson 280.4, 966-4163, *croy@cs.usask.ca*
Office Hours: By appointment and after lecture.
Teaching Assistant: Farouq AlOmari <farouq.alomari@gmail.com>

**Texts**

Readings from a variety of sources will be provided as PDFs on the course website. Also the following texts:

J.R. Cordy, "Excerpts from the TXL Cookbook", Generative and Transformational Techniques in Software Engineering, Lecture Notes in Computer Science 6491, January 2011, pp. 27-91.

Vaclav Rajlich “Software Engineering: The Contemporary Practice”, CRC Press, Nov. 2011, 315 pages.

**Course Contents:**

This course builds on the understanding of software engineering presented in CMPT 370 and (to a lesser degree) CMPT 371. The focus is on techniques to help foster *quality* software engineering. The major topics covered will range from introductory but latest software engineering concepts to advanced software quality assurance and maintenance principles and techniques. The major topics are:

**1. Source Transformation for Advanced Software Engineering**

The students will learn the importance of source transformation systems and their applications in software engineering. In particular, the students will learn the TXL source transformation language and then will apply TXL in several advanced software engineering activities such as source to source transformation (e.g., C language system to Java language systems or vice versa), pretty-printing (standard formatting of one’s code), improving the quality of the software systems including the performance, support for better software testing, software maintenance activities including clone detection, analysis and refactoring. The students will also learn the associated state of the art tools.

## 2. Design Patterns

Design patterns are one of the important aspect of quality software and thus for quality software development. The course will extensively talked about all those advanced design patterns from Gang of Four Design Patterns. In particular, the course will discuss the following design patterns: Singleton, Adapter, Facade, Observer, Builder, Bridge, Chain of responsibility, Factory, Abstract Factory, Interpreter, State, Strategy, Prototype, Mediator, Memento, Command, Flyweight, Visitor, Proxy, Composite and Decorator design patterns.

## 3. Development Anti-Patterns

As of the Design Patterns, Anti-Patterns, in particular, development Anti-Patterns are equally important for quality software development. In this course, we will also discuss in details a few of the development anti-patterns such as Spaghetti Code, Mushroom Management, Stovepipe Enterprise, Cover Your Assets, Vendor Lock-In, Design By Committee, Warm Bodies, Jumble, Wolf Ticket, Swiss Army Knife and so on.

## 4. Software Quality Assurance

Software quality assurance is the primary theme of this course and the students will learn the details of software quality, what is it, why do we need it, how to measure it, what are the different methods of providing quality assurance. The students will learn the details of different testing methods including black box, white box, grey box and mutation testing methods.

## 5. Code Smells and Refactoring

Given that software quality is the primary goal of this course, the students will also learn the details of different code smells including the number one code smell, the software clones. And then they will learn the different refactoring methods of how to remove those code smells including code clones.

## 6. Software change

This part of the lecture deals with software change, which is a foundation of most of the software engineering processes. We will first explain the classification of changes, followed by the explanation of the requirements, their analysis, and the product backlog. Software change consists of several phases, and the first one is the selection of a specific change request from product backlog. This is followed by the phase of concept location where the programmers find what specific software module needs to be changed. Then impact analysis decides on the strategy and extent of the change. Actualization implements the new functionality and incorporates it into the old code. Refactoring reorganizes software so that the change is easier (prefactoring), or cleans up the mess that the change may have created (postfactoring). Verification validates the correctness of the change. Change conclusion creates a new baseline, prepares software for the next change, and possibly releases the latest version to the users. This part of the lecture establishes a foundation on which the next part, software processes, builds.

**7. Software Process**

This part of the lecture presents the most common software processes. It explains what a software process is and what the forms are (model, enactment, performance, and plan). Then it deals with the iterative process of a solitary programmer (SIP), the team agile iterative process (AIP), directed iterative process (DIP), and centralized iterative process (CIP); these processes are primarily applicable to the stages of software evolution, servicing, but they also apply also to initial implementation and reengineering. This part then covers initial development and the final stages of software lifespan and hence it presents an overview of processes applicable to all stages of software lifespan.

**8. Advanced Software Engineering Topics**

In addition to the above topics, this course will also discuss on some other advanced software engineering topics as follows:

a. Subtyping, Subclassing and Liskov Substitute principle, Open Close principle
b. The Map/Reduce Framework
c. Garbage Collection
d. Multithreading in Java
e. Architectural Styles
f. Jini
g. Dynamic Proxies In Java
h. Remote Method Invocation (Java RMI)
i. Actor Model
j. Java IDL
k. Evolutionary Coupling

**Grading Scheme:**

- Individual Participation (grd+urd) 5%
- Assignments (grd+urd) 45%
- Final Exam (und) 50%
- Project (grad) 50%

**Project/Exploratory Topic:**

- Mandatory for graduate students, optional for undergraduates (transfer marks from final exam)
- Do a project in consultation with the instructor
- Task: Investigate a software engineering topic of your choice (subject to instructor approval)
    - Languages, tools, processes, formalisms

        - Critical synthesis and analysis expected
- Stages
    - Introductory Proposal
    - Intermediate Progress, Outline of sources & preliminary findings, description of emphasis
    - Final Report
    - Encouraged: Discussion with instructor
- Essential: Care in citing & quoting

**Assignments:**

There will be a number of assignments on the topics covered.