ECE 321 Software Requirements Engineering

Lecture 1: Introduction

Cor-Paul Bezemer

- **ICE 11-368**
- **780.492.8355**
- **bezemer@ualberta.ca**
  - I respond to e-mails between 7.30am and 4.30pm

- **Office hours**
  - Thursday, 1:00 PM – 2:00 PM
  - Or by appointment (via e-mail)

Teaching philosophy

- Lectures and participation are important
- Interactive mode: make comments and ask questions!
- Lectures are given in a ‘comfortable’ pace
  - If not – please tell me!

Schedule

- **Lectures**
  - Thursday, 11:00 – 12:50
- **Laboratories**
  - Monday OR Friday, 14:00 – 16:50, ETL E5 005
  - Starting in Week 3 (September 17)
  - Make sure to register in BearTracks!

Course basics

- **Class web site**
  - UofA eClass/Moodle at https://eclass.srv.ualberta.ca
- **Pre-requisites**
  - CMPUT 275: Tangible Computing II
- **Laboratory instructor**
  - Ahmed Chaari (chaari@ualberta.ca)
  - Mahsa Panahandeh (panahand@ualberta.ca)

Textbooks

- **NONE (kind of...)**
  - eClass with class notes, laboratory manuals, assignments, and other relevant class information
  - **Recommendations of some books** relevant to different topics (as we go through the material)

eClass

- **Here you can find**
  - Class schedule (provided in the syllabus)
  - Class notes and lab manuals
  - Assignments and solutions
  - Announcements
  - Pointers to class-related information
  - Grades
- **Make sure to check it often!**

The course according to the calendar description

Software quality attributes. Software requirements. Requirements elicitation via interviewing, workshops, prototyping and use case analysis. Vision document and Software Requirement Specification document standards. Formal software specification methods including operational and descriptive models. Design by contract. Verification and validation of requirements.

The course in a nutshell

Provides the background necessary to develop, specify, and verify software requirements

Course objectives

- To help you build **software** that is **easy** to **maintain**, **modify** and **re-use**
- To **expose** you and **introduce** you to
  - Different software engineering concepts
  - The requirements engineering field
  - Formal methods of representing software behaviour and its verification

Lecture topics guide

- **Principles of software engineering**
  - Software development models
  - Modularity, abstraction, anticipation of change
  - Information hiding

Lecture topics guide

- **Requirements engineering**
  - Analyzing the problem & understanding user needs, use cases – elicitation techniques
  - Defining the system documentation (SRS – software requirements specification, formal methods)
  - Quality measures & validation of the requirements

Learning outcomes

- Learn and understand basic software engineering concepts such as: modularity, anticipation of change, abstraction, information hiding, software development methods/models
- Understand and appreciate the value of requirements engineering
- Understand a need for formal methods, in particular be exposed to and learn Petri Nets as formal system description technique

More learning outcomes

- Analyze the users’ problems and understand their needs
- Develop use cases
- Select and apply an appropriate elicitation technique
- Prepare system requirement documents such as: Vision Document and Software Requirements Specification (SRS)

Even more learning outcomes

- Build a model of a designed system using Petri Nets, and validate it
- Determine values of quality measures describing a formal design

Course evaluation

- Your final grade will be based on a scale that is **adjusted** based on the overall class performance
- **Example** of a scale

[100% - 90%]	A+/A/A-
(90% - 80%]	B+/B/B-
(80% - 70%]	C+/C/C-
(70% - 60%]	D+/D
(60% - 0%]	F

Course evaluation

- **Four assignments** 30% (2 for 7% + 2 for 8%)
- **Group project, lab reports** 30% (15% + 5 for 3%)
- **Final exam** 40%
- **In-class participation** extra 3%

- Note: you need to score at least 50% on the final exam

Assignments

- Individual work
- No programming required
- 2-3 weeks per assignment (check the schedule)
- Two paper review assignments (7% each)
- Two problem-based assignments (8% each)

Theme of the assignments

- **Main theme:** development of a requirements specification for a software system
- **Additional theme:** questions relevant to software engineering and the course material
  - Reviews of articles and hands-on exercises

Submitting assignments

- Submitted before the **beginning** of a class in eClass
  - PDF **ONLY**

Late assignments policy

- 15% penalty if within 12 hours
- 30% penalty after 12 hours
- Not accepted after 48 hours unless approval is obtained from the instructor before the due date

Group work

- All lab activities are performed in groups
  - Each group consists of 4 people (groups will be set up during the first lab session)
  - 5 lab reports for 3% each
    - 3 finished during the lab, 2 take home
  - Final project for 15% (completed SRS document using IEEE 830 standard)

Final exam

- Will be comprehensive (cover the entire term)
- Closed books, no electronic devices
- 40% weight
- You need to score at least 50% on the final exam to pass the course

Contents of final exam

- Three parts
  - Multiple choice (matching table, multiple choice, T/F)
  - Short questions (theory and open-ended)
  - Small hands-on questions (similar to one covered in the class or lab)

Class participation (extra credit)

- For in-class questions and discussion
  - Points will be given whenever you participate
  - Put your name on the participation list after class
  - Not in class? No points
- Extra credit for 3% of the grade
  - Max score will be equal to average of participation of all students

Any questions so far?

Software – what is it?

- **Computer program AND its related artifacts**
  - Requirements, documentation, test plans, tests, protocols, manuals, user interface, usability tests
- **Changes to program or artifacts usually force changes to other components**

Engineering

- The application of scientific principles in the context of practical constraints
- A sequence of well-defined, precisely-stated, sound steps that follow a method or apply a technique based on a combination of:
  - Theoretical results derived from a formal model
  - Empirical adjustments for an unmodeled phenomenon
  - Rules of thumb based on experience

Software engineering according to Barry Boehm

- Software engineering is that form of engineering that applies
  - A systematic, disciplined, quantifiable approach
  - The principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines

  to create, develop, operate, and maintain cost-effective, reliably correct, high quality solutions to software problems

Software engineering according to David Parnas

- Software engineering involves a multi-person construction of multi-version software

Software engineering is not just about coding!

- Consists of
  - Software process
  - Requirements development and verification
  - Design
  - Implementation
  - Testing and verification
  - Maintenance

Software engineering should guarantee

- Production of quality software
- Delivery of the product on time and within budget
- Satisfaction for the users’ needs

Software problems

- Epic failures: 11 infamous software bugs
  - Matt Lake, Computerworld, September 9, 2010
- More at RISKS Digest
  - Forum on ‘Risks to the public in computers and related systems’ by ACM

    http://catless.ncl.ac.uk/Risks

Famous software problems

- Mariner 1 spacecraft explodes (1962, $554M)
  - Poorly written documentation where a math symbol was misinterpreted
- Ariane-5 explodes (1996, ~$500M)
  - Overflow error due to faulty conversion between 64-bit floating-point data and 16-bit signed integers
- Loss of NASA’s Mars Climate Observer (1998, $328M)
  - Conversion error between English and Metric units (thrusters were 4.45 times more powerful than they should have been)

A bigger picture

- Software bugs cost the U.S. economy $1.7 trillion in 2017
  - Approximately 9% of the gross domestic product that year!
  - vs. 0.6% in 2002

Why care about requirements?

- Would you buy a car that you know will malfunction?
- Software is one of the few markets where buyers accept products of which they know it will malfunction
- You will learn that requirements are one of the places where the most costly bugs are introduced