1. Cmpt 276 - Intro to SW Engineering

Last Mod: 2024-04-29

Instructor: Mr. Russell Tront

Office: ????

Email: tront@sfu.ca

Classes: MWF 1:30 – 2:30m, AQ3003

Wienberg's Second Law:

"IF BUILDERS BUILT BUILDINGS THE WAY PROGRAMMERS WROTE PROGRAMS, THE FIRST WOODPECKER THAT CAME ALONG WOULD DESTROY CIVILIZATION."

The above embarrassingly suggests how the public and business sometimes regards the software industry.

In this course, you'll learn some reasons for this. And learn:

- professional S/W development strategies and techniques, and
- S/W development management strategies

to improve this situation.

Table of Contents

1.1 Calendar Description:

- An overview of various techniques used for:
  - software development and
  - project management.
- Major tasks and phases in modern software development, including:
  - Requirements
  - Analysis
  - Documentation
  - Design
  - Implementation
  - Testing
  - Maintenance

- Project management issues are also introduced.
- Students complete a team project using an iterative development process.

1.2 Course Outline

The focus is preparing students to be effective members of an industrial software development team. The theory and practice of software development is introduced. Students will learn various SW development methodologies. And learn multi-person team and project challenges.

The term-long project will use approximately 4-person teams. Each team will produce 5 major document/code drops, one every two weeks, during the last 10 weeks of the course.

Topics:

- The software lifecycle. And challenges including estimation, maintenance/updates, and revision control. Waterfall/plan-driven vs. agile project management.
- Requirements gathering/elicitation, including use cases, data, states, flows.
- Systems analysis, with some object-oriented data normalization. Modelling with some UML notation. Writing a requirements specification.
- Software and documentation revision control.
- User/external interface design (briefly). Write user documentation as a system external spec, before writing code.

- High level/architectural design principles, UML notation, design patterns (briefly). Writing module interfaces first, that others on your team can call/write to.
- Object-oriented design (briefly), implementation, coding style.
- Quality Assurance: Code reviews, unit and integration testing.
- Project estimation and planning, project metrics.
- Time permitting: Maintenance/update issue tracking systems.

Grading:

- 20% Midterm exam
- 40% Final exam
- 40% Team project. Including documentation quality (e.g. spelling/grammar/version control). Other project marking details to be discussed in the first week of class.

Students must attain an overall passing grade on the weighted average of the exams in order to obtain a clear course pass (C- or better).

Textbook and Required Materials

- No textbook will be required.
- Required: A purchased "course pack" of reading materials (if/when needed) will become downloadable from SFU bookstore.

1.3 Team Project and Marking Details

Working in a team is a considerable challenge in many ways! Here's one:

- Assignment material from each team member should be delivered *several days, or a week, before the due date*!
  - Allows one other team member to review the material.
  - Then errors in the material should be pushed back to author to fix.
  - Perhaps needs another review.
  - Then the work must be integrated by a team member into the whole submission.

- Late deliverers will thus drive the rest of the team nuts!
- And poor quality or completeness too.

To discourage laggards, team members will evaluate each other.

- You will be prevented from giving all your teammates a high mark.
- We just want a relative evaluation; We can judge good teams by the quality of the submitted joint work.

The evaluation may result in a course mark adjustment in extreme cases of up to 1/3rd of a letter grade.

1.4 Lecture Note Sections (TBD)

Previously, my notes were divided into these sections. This may change in semester 24-2.

1. Introduction to Cmpt 275.
2. Motivation and Life Cycles.
3. Requirements Gathering.
4. Requirements Analysis and Specification.
5. Object Oriented Analysis and Design
6. External Design (User Interface Design and User Manual).
7. Architectural Design.
8. Software Interface Design
9. Low Level Design
10. Implementation and Unit Testing
11. Integration and Integration Testing

12. Testing.
13. Managing the Software Development Process
14. Software Quality Assurance.
15. Software Metrics
16. Project Planning (Effort and Schedule Estimation).
17. Maintenance, Configuration Management, and Wrap-up.

1.5 Required Readings

Please read the following right away. Available from canvas.sfu.ca in Cmpt 276 D100 course.

[Adolph96] “Cash Cow in the Tar Pit: Re-engineering a Legacy System”, IEEE Software, May 1996.

- Written by an SFU grad about Vancouver company doing Skytrain software!

1.6 Other References

[Brooks75] Brooks, Frederick P. Jr., “The Mythical Man Month: Essays on Software Engineering”, Addison-Wesley, 1995. In particular:

- Chapter 1 “The Tar Pit”.
- Chapter 2 “The Mythical Man-Month”
- Chapter 3 “The Surgical Team”
- Chapter 5 “The Second System Effect”.

[Gibbs94] Gibbs, W. Wyatt, “Software’s Chronic Crisis”, Scientific American magazine, Vol. 271, No. 3, Scientific American, 1994.

[Joch95] Josh, Dan,“How Software Doesn’t Work”, Byte Magazine, McGraw-Hill, 1995.

[Rose95] Rose, Barbara Wade, “Danger: Software at Work”, The Globe And Mail Report On Business, Vol.11, No.9, Thompson Newspapers Co., 1995.

1.7 Objectives

This course introduces you to two major topics:

- S/W Systems Analysis and Design, and
- S/W Development Process Management

Many think "software engineering" is any part of software development. I and others feel it's the:

- study of,
- comparison, and
- use of

techniques to improve software development quality, productivity, and management.

To:

- understand the full development process and
- many of the management issues involved,

the course is structured around a:

- multi-phase,
- medium-sized team

software development project.

The goal of the assignments is more than just to construct some software.

Instead:

- to learn,
- apply and
- “experience”

the proper practices of software engineering!

The project is just the “vehicle” on which you will apply good software development practices.

It’s not great to begin a programming project until:

- you know the program’s requirements.
- have conceptualized the program’s architecture.

So in Cmpt 276 we’ll first do an introduction to requirements analysis and design.

The rest of the course will provide an introduction to software engineering and process management principles.

1.8 Web Materials and Other Administrative Details

Must of the course info:

- Instructor office hours and location.
- TA office hours and location.
- Lecture slides.
- Required reading materials.
- Assignments
- etc.
- will be posted on canvas.sfu.ca under your Cmpt 276 D100 course.

Canvas will likely have an archive of emails I’ve sent the whole class.

- Do not turn off Canvas notifications of any type of postings.

Regarding instructor office hours:

- Feel free to drop by for any concern re the course.
- Or make an appointment (MWF) outside of office hours.
- Or knock any time (MWF).

1.9 More on Team Assignments

- When programmers try to code with each other, all kinds of things can go wrong.
  - Functions get called with the wrong type of parameter, links don’t work, formats don’t match, and **no team member can understand everything about the project**.
- Working in teams allows the project to be *big enough* to encompass a dozen or more modules, and thus allow you to encounter the inter-module and inter-person difficulties that can result.
  - Teamwork is an important part of the Cmpt 276 ‘experience’.

- It’s likely your first time working on a program where you don’t know what and how all the other modules do exactly!
- Group work has a number of interesting problems. You’ll encounter team member:
  - skill and productivity differences,
  - late vs. early deliverers,
  - loners, etc.

I will help you form teams in the second week of the course.

Note: It may not be wise to form teams with your friends!

- Firstly, you may loose them as friends by the end of the course. Teamwork involves some conflict (in design, coding, personalities, etc.).
  - And coordination of parts of a large project is difficult even in the best of circumstances.

- Secondly, it’s important to choose team members with diverse skills. You want:
  - some excellent in English (as much as 10% of some of your assignment marks will be based on spelling, grammar, and clarity of writing),
  - some excellent in user interface design, in Git set-up, and the particular programming language,
  - some with great organizational skills.

  Likely your friends do not compose such a diverse group.

More about the team project:

- Need basic fluency in C++ or Java to understand the lecture code fragments.
- Choose a language which is compatible with your team, likely C++.
- The instructor will help with team creation in a few days' time.
- You'll need to use a code+document revision control system, likely Git.

1.10 Late Penalties

- The late penalty is 5% per day.
- Assignments are due at the START of class!
  - Some might plead that the end of class or midnight might be better. But if not the start of class, then no one comes to class on the due date because they're finishing up the assignment!
  - The deadline is binary.
  - You may hand in the next day up to 11:59 PM with only a 5% per calendar day penalty.

1.11 Scheduling

There are 5 assignments, each due approximately 2 weeks apart. Each assignment is worth the same amount:

- Requirements Specification.
- User Manual.
- Design Specification (including main module, other module header files, and some test case specifications).
- Write, compile, and unit test the various modules (but don't need to link).
- Integrate, link, test, write Integration and Test Report, and demo the software to the Teaching Assistant.

Important:

You can’t do these assignment in the last 3 days before they are due. Generally, you have to:

- Meet in the first few days of the assignment to divide up the work and decide who will be team leader for that assignment.
- Do your main work during the **first** of the two weeks.
- At the start of the second week,
  - meet to exchange documentation and code files.
  - Then some evaluation, testing, and integration takes place.

- Files are then sent back to their author marked up with errors for the author to fix.
  - If authors are not made responsible for fixing their code and documentation mistakes, they will generate shoddy work.
- Have the fixed files ready 3 complete days before the assignment is due.
- The files and document chapters then go through final integration and touch up.

1.12 Configuration Management

- Each document and source file must have a ‘version’ number. (Sometimes called ‘revision’.)
- Each assignment is a ‘release’ of the system.
- A release is composed of a coordinated set of files. e.g. Revision 2 of Requirement Spec. goes with Revision 4 of the User Manual which goes with Revision 9 of the main source file.

If any of your earlier assignment documents are inconsistent with those that you produce for later assignments, you must update the earlier docs and hand them in with the later assignment.

Each assignment will contain a release table specifying the revision number of each document that makes up that release.

1.13 Little Iteration or Maintenance

In the real world, greater than 50% of all effort is spent modifying existing code!

And, maintenance is difficult due to:

- poor or no design documentation,
- poor or no code commenting,
- poor code structure.

This provides a major motivation for spending extra time doing the original software development well in the first place.

- E.g. Would you as a project manager spend \$10,000 now and 1 month delay, in order to save \$20,000 later?

Unfortunately, as a student, you're likely missing these kinds of motivations.

And it's rarely possible to simulate in a university environment.

It's also hard to do iterative software development in a course, because

- you'd need an existing system and documentation to start on,
- you'd dislike working on someone else's code, and
- would get no experience doing initial requirements and design.

Nonetheless, I'll try to raise your consciousness in this regard by:

- illustration,
- industrial horror stories,
- required readings,
- presentation of obviously good software engineering principles, and
- if necessary, waving my arms and jumping up and down a lot!

One of the most interesting experiences/revelations in software development:

- Trying to add features or fix bugs in code you wrote more than 6 months ago. And finding you can’t understand the code!
- I promise this will happen to you in your first year of employment on a huge project, unless you code is well commented.