CSC 408F/CSC2105F Lecture Notes

CSC408F/CSC2105F - Software Engineering

Instructor	Dr Yijun Yu Sandford Fleming Building, Room 1101 yijun@cdf.toronto.edu	
Marking	Mid term test	15%
	Final Exam	35%
	Course Project	50%
Text	Hans van Vliet Software Engineering - Principles and Practice, 2nd ed.	
News Group	`ut.cdf.csc408h` **Read the newsgroup on a regular basis!**	
Web Page	`www.cs.toronto.edu/~yijun/cs408h/`	
Slides	`CDF: ~yijun/csc408h/slides/`	

Course Project

- Reengineering a large software system
- Work in teams
- Three Phase Project
  A. Study a large legacy software and specify new requirements
  B. Partial implementation
  C. Swap software and complete implementation
- Select any other teams software at swaps
- Project specification announced soon

Immediate Project Issues

- Team Selection
  - Work in teams of 5, self-selection of teams
  - Form teams *ASAP* for start of project
  - Instructor/tutor deal with orphans/disputes
- Programming Language for Project
  - Discuss/decide in class after project is announced
  - Possible choices XML, C/C++, Java
- **Selecting a hard-working, compatible team is important for success in the course project**

Major Themes

- What is Software Engineering
  - van Vliet Chapter 1
- The Software Process
  - van Vliet Chapter 3
- Software Project Management
  - van Vliet Chapters 2,5,6,7,8
- Software Construction
  - van Vliet Chapters 4, 17, 19
- Software Testing and Validation
  - van Vliet Chapters 13,15
- Software Product Delivery, Software Maintenance
  - van Vliet Chapter 14

Topics Not Covered in CSC408H

- Requirements and Specifications
  - van Vliet Chapter 9
  - CSC340H or CSC2106H
- Software Architecture and Software Design
  - van Vliet Chapters 10, 11, 12
  - CSC407H or CSC2103H
- User Interface Design
  - van Vliet Chapter 16
  - CSC318H or CSC2514H

Reading Assignment

```
van Vliet, Chapter 1

           Sections  3.5,  3.7

           Sections 11.1.1 .. 11.1.3

Usenet news group    comp.risks

READ comp.risks on a regular basis
```


Alternative Major Themes

- How to survive in a large software project
- Getting software right the first time
- Minimizing software production costs
- Maximizing software quality
- Being organized and systematic is infinitely better than not being so
- An engineering approach should be used for the development of software
- Design for Maintainability is really important.

Software Engineering Information Sources

- IEEE Transactions on Software Engineering
- ACM Transactions on Software Engineering and Methodology
- International Conference on Software Engineering (yearly)
- IEEE Software
- ACM SigSoft - Software Engineering Notes (monthly)
- Software - Practice & Experience
- International Conference on Software Maintenance
- Usenet news group - comp.risks, comp.soft-eng
  comp.risks archive: http://catless.ncl.ac.uk/Risks/
- Libraries: CS & Engineering, Gerstein, Metro Reference

What is Software Engineering

- The science and art of building *LARGE* Software Systems
  - On time
  - On budget
  - With Acceptable Performance
  - With Correct Operation
- *LARGE* means:
  - Many people, team not individual effort
  - Many $s spent on design and implementation
  - Over 75,000 lines of source code
  - Lifetime measured in years
  - Continuing modification and maintenance
- Software costs dominate hardware costs

Calibration on LARGE

- 1,000,000 lines of source code
- 13,333 pages at 75 lines/page (laser print)
- A 4’ 5” (1.46m) high stack of paper at 500 pages/2”
- 41,667 screens at 24 lines/screen, or
  20,833 screens at 48 lines/screen
- 22.2 hours to print at 10 pages/minute
- 16.7 hours to compile at 1000 lines/minute (wildly optimistic)

Many real software systems are 3,000,000 to 6,000,000 lines of source code.
Many existing systems are in the 10,000,000 to 20,000,000 range.
Windows 98 is alleged to be more than 50,000,000 lines.

Why is Software Engineering Important

- Cost of getting software *wrong* is often horrendous
  - Bankruptcy of software producer
  - Injury or loss of human life **Broken software can KILL people**
- Software producer profitability depends on producing software efficiently and minimizing maintenance effort. Software reuse is an economic necessity
- Immense body of old software (legacy code or dusty decks) that must be rebuilt or redesigned to be usable on modern computer systems
- Software maintenance increases the entropy (disorder) in a software system. Without proper care the software can become unmanageable and unusable
- Very, very few contemporary systems work correctly when first installed. We need to do much better
- Over $600,000,000,000 spent each year on producing software

Software Horror Stories[a]

- Bank of America spent $23,000,000 on a 5-year project to develop a new accounting system. Spent over $60,000,000 trying to make new system work, finally abandoned it. Loss of business estimated in excess of $1,000,000,000
- Starting in 1982, Allstate Insurance spent $8,000,000 on an effort to automate its business. The supposed 5-year project continued until 1993 at a cost approaching $100,000,000
- The B1 Bomber required an additional $1,000,000,000 to improve its air defense software, but the software still isn't working to specification
- A U.S. Air Force air defense system was $1,000,000,000 over budget, 4 years behind schedule and only marginally usable
- A regional Blue Cross service lost $60,000,000 in incorrect over payments due to errors in a $200,000,000 computer processing system that wasn't adequately tested before being put into service.

[a] P.Neuman, System Development Woes, CACM, Oct 1993

- Ariane 5, Flight 501
  The loss of a $500,000,000 spacecraft was ultimately attributed to errors in requirements, specifications and inadequate software reuse practices.
- Therac-25
  Hospital patients died or were seriously injured due to errors in programming the user interface of this machine.
- London Ambulance Service
  Major software problems rendered this ambulance dispatching system unusable. Poor project management and poorly developed requirements contributed to the problems. People died and/or received less than optimal medical care due to the problems with this system.

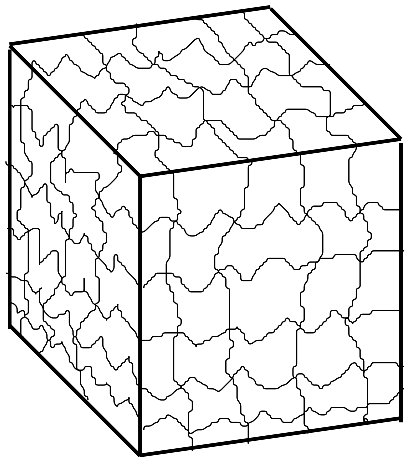
What is Software ?

- Requirements and specification documents
- Design documents
- Source Code
- Test suites and test plans
- Interface to hardware and software operating environment
- Documentation, internal and external

What Makes Large Software Different ?

Scale	Precludes total comprehension
Complexity	Number of functions, modules, paths
Team Effort	Continuingly changing body of programmers
Communication	Distribution of specifications and documentation
Continuing Change	During design & implementation
	During lifetime
Lifetime	Measured in years or decades
Imprecise goals	Conflicting or ambiguous, changing

Real Programs Resemble Large Puzzles



- The dimensionality of the puzzle (connections between pieces) is much higher that can be shown on a 2-dimensional slide.
- Each piece may be written by different (group of) programmers.
- Each piece has a *rigorously specified* **interface** that describes how it interconnects with other pieces.
- For a program to be correct, all of the pieces must fit exactly.
- The shape of the pieces and the interconnections between them change over time as the program is modified and maintained.

Issues in Software Engineering

- Major concern is the construction of *large* programs.
- Central theme is *mastering complexity*.
- Software evolves over its lifetime.
- The efficiency of *software development* is of crucial importance
- Regular cooperation between people is an essential and unavoidable part of large software development.
- Software has to support its users effectively.
- Software Engineering is a field in which members of one culture (designers, programmers) create artifacts on behalf of members of another culture (end users).

The Ideal Goals of Software Engineering

- To produce software that is absolutely correct.
- To produce software with a minimum of effort.
- To produce software at the lowest possible cost.
- To produce software in the least possible time.
- To maximize the profitability of the software production effort.
- To produce software that can be maintained with a minimum of effort.

In practice, none of these ideal goals is ever completely achievable.
The challenge of Software Engineering is to see how close we can get to achieving these goals.

The *art* of software engineering is achieving the best balance among these goals for a particular project.