CSC340: Information Systems Analysis and Design
Prof Steve Easterbrook
sme@cs.toronto.edu
http://www.cs.toronto.edu/~sme/CSC340F

About the Course

Course website
% www.cs.toronto.edu/~sme/CSC340F/
Textbooks
Lecture Notes
% Available on the course website prior to each lecture
Coursework
% Carried out in teams of 3
% Each team submits one report (per assignment)
% All team members receive the same grade (exceptions can be negotiated)
Deadlines
% Are very strict (use a U of T medical certificate if you are seriously ill)
% Daily penalties apply to late work

Course Objectives

→ Examine the state-of-the-art for research & practice in Requirements Engineering.
  % Role of RE in software and systems engineering
  % Current techniques, notations, methods, processes and tools used in RE
→ Gain practical experience in selected RE techniques
  % Especially goal-oriented and object-oriented modeling techniques
→ Understand the essential nature of RE
  % Breadth of skills needed for RE, and the many disciplines on which it draws
  % Contextual factors & practicalities

A note about terms:
- "Systems Analysis" ⊂ "Requirements Engineering"
- SA typically refers only to information systems
- RE applies to all software-intensive systems
- This course is evolving to cover more of RE

Assessment

→ 3 team assignments:
  1. Conduct an inspection of an existing specification (10%)
     ➢ Report on defects found, overall quality, and inspection stats
  2. Perform a feasibility study for an information systems project (15%)
     ➢ Write a feasibility report
  3. Perform a requirements analysis for the same project (20%)
     ➢ Write a requirements specification
→ 2 tests:
  % Midterm test (20%)
  % Final Exam (35%)
     ➢ Must obtain at least 30% on this exam to pass the course.
Software-Intensive Systems

- **Software (on its own) is useless**
  - Only useful in the context of some human activity that it can support.
  - We sometimes take the human context for granted.
  - Software + Hardware + Human Activities = "Software-Intensive System"

- **A Computer System (on its own) is useless**
  - Only useful in the context of some human activity that it can support.
  - We sometimes take the human context for granted.
  - Software + Hardware + Human Activities = "Computer System"

- **'Software' makes many things possible**
  - It is complex and adaptable.
  - It can be rapidly changed on-the-fly.
  - It turns general-purpose hardware into a huge variety of useful machines.

Quality = Fitness for purpose

- **Software technology is everywhere**
  - Affects nearly all aspects of our lives.
  - But our experience of software technology is often frustrating/disappointing.

- **Software is designed for a purpose**
  - If it doesn’t work well then either:
    - The designer didn’t have an adequate understanding of the purpose.
    - We are using the software for a purpose different from the intended one.

Quality = Fitness for purpose

- **The purpose is found in human activities**
  - E.g., purpose of a banking system comes from the business activities of banks and the needs of their customers.

- **The purpose is often complex**
  - Many different kinds of people and activities.
  - Conflicting interests among them.

Where are the challenges?

**Application Domain**
- Domain properties
- Requirements
**Machine Domain**
- Specification
- Programs
- Computers

Complexity of Purpose

- **People and software are closely-coupled**
  - Complex modes of interaction.
  - Long duration of interaction.
  - Mixed-initiative interaction.
  - Socially-situated interaction.

- **The problems we’d like software to solve are “wicked”**
  - No definitive formulation of the problem.
  - No stopping rule (each solution leads to new insights).
  - Solutions are not right or wrong.
  - No objective test of how good a solution is (subjective judgement needed).
  - Each problem is unique (no other problem is exactly like it).
  - Each problem can be treated as a symptom of another problem.
  - Problems often have strong political, ethical or professional dimensions.
Dealing with problem complexity

→ Abstraction
  % Ignore detail to see the big picture
  % Treat objects as the same by ignoring certain differences
  % (beware: every abstraction involves choice over what is important)
→ Decomposition
  % Partition a problem into independent pieces, to study separately
  % (beware: the parts are rarely independent really)
→ Projection
  % Separate different concerns (views) and describe them separately
  % Different from decomposition as it does not partition the problem space
  % (beware: different views will be inconsistent most of the time)
→ Modularization
  % Choose structures that are stable over time, to localize change
  % (beware: any structure will make some changes easier and others harder)

Which systems are soft?

→ Generic software components
  % E.g. Core operating system functions, network services, middleware, ...
  % Functionality relatively stable, determined by technical interfaces
  % But note that these systems still affect human activity
    > E.g. concepts of a ‘File’, a ‘URL’, etc.
→ Control Systems
  % E.g. aircraft flight control, industrial process control, ...
  % Most requirements determined by the physical processes to be controlled
  % But note that operator interaction is usually crucial
    > E.g. accidents caused when the system doesn’t behave as the operator expected
→ Information Systems
  % E.g. office automation, groupware, web services, business support, ...
  % These systems cannot be decoupled from the activities they support
  % Design of the software entails design of the human activity
    > The software and the human activities co-evolve

Designing for people

→ What is the real goal of software design?
  % Creating new programs, components, algorithms, user interfaces, ...?
  % Making human activities more effective, efficient, safe, enjoyable, ...?
→ How rational is the design process?
  % Hard systems view:
    > Software problems can be decomposed systematically
    > The requirements can be represented formally in a specification
    > This specification can be validated to ensure it is correct
    > A correct program is one that satisfies such a specification
  % Soft systems view:
    > Software development is embedded in a complex organisational context
    > There are multiple stakeholders with different values and goals
    > Software design is part of an ongoing learning process by the organisation
    > Requirements can never be adequately captured in a specification
    > Participation of users and others throughout development is essential
→ Reconciliation:
  % Hard systems view okay if there is local consensus on the nature of the problem

Definition of RE

Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies.

Need to identify all the stakeholders - not just the customer and user.
Cost of getting it wrong

→ Cost of fixing errors
  % Typical development process:
  requirements analysis → software design → programming → development testing → acceptance testing → operation
  % Errors cost more to fix the longer they are undetected
  > E.g. A requirements error found in testing costs 100 times more than a programming error found in testing

→ Causes of project failure
  % Survey of US software projects by the Standish group:

<table>
<thead>
<tr>
<th></th>
<th>1994</th>
<th>1998</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>16%</td>
<td>26%</td>
</tr>
<tr>
<td>Challenged</td>
<td>53%</td>
<td>46%</td>
</tr>
<tr>
<td>Cancelled</td>
<td>31%</td>
<td>28%</td>
</tr>
</tbody>
</table>

Top 3 success factors:
1. User involvement
2. Executive management support
3. Clear statement of requirements

Top 3 factors leading to failure:
1. Lack of user input
2. Incomplete requirements & specs
3. Changing requirements & specs

What do Requirements Analysts do?

→ Starting point
  % Some notion that there is a “problem” that needs solving
  > e.g. dissatisfaction with the current state of affairs
  > e.g. a new business opportunity
  > e.g. a potential saving of cost, time, resource usage, etc.
  % A Requirements Analyst is an agent of change

→ The requirements analyst must:
  % identify the “problem”/“opportunity”
  > Which problem needs to be solved? (identify problem Boundaries)
  > Where is the problem? (understand the Context/Problem Domain)
  > Whose problem is it? (identify Stakeholders)
  > Why does it need solving? (identify the stakeholders’ Goals)
  > How might a software system help? (collect some Scenarios)
  > When does it need solving? (identify Development Constraints)
  > What might prevent us solving it? (identify Feasibility and Risk)
  % and become an expert in the problem domain
  > although ignorance is important too — “the intelligent ignoramus”

Summary

→ This course covers most of requirements engineering:
  % Analyzing problem situations
  % Studying human activities
  % Formulating requirements so that software solutions can be designed

→ This course is different to most CS courses
  % It is not about how to solve problems using computers
  % It is about how to identify problems worth solving
  % The subject matter is human activity:
    > how to understand it
    > how to support it using software technology

→ Your mileage will vary
  % Comments from students in previous years vary dramatically:
    > “At last – a course that actually taught me something useful”
    > “This course should be scrapped – it’s an embarrassment to CS”