CSC2106S
Requirements Engineering

Prof. Steve Easterbrook
sme@cs.toronto.edu
http://www.cs.toronto.edu/~sme/CSC2106S/

Today's Menu

Start Here

This Week:
Aims of the course
Syllabus
Readings
What are Requirements?

Next Week:
Engineering Context
Systems Thinking
Role of Modeling

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

Not a phase or stage!

Communication is as important as the analysis

Quality means fitness-for-purpose. Cannot say anything about quality unless you understand the purpose

Designers need to know how and where the system will be used

Requirements are partly about what is needed…

…and partly about what is possible

Need to identify all the stakeholders - not just the customer and user

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
- Understand the essential nature of RE
  - Breadth of skills needed for RE, and the many disciplines on which it draws
  - Contextual factors & practicalities
- Gain a basic grounding for research in RE
  - Methodological issues for RE research
  - Current research issues & direction of the field
  - Awareness of the literature

Teaching and Assessment

- 1 x 3 hour seminar per week (13 weeks)
  - Discussion of weekly reading material
  - Student presentations
  - Plus typically up to 1 hour of "lecture" material from me.
- Weekly readings
  - 1 or 2 papers per week (must read *before* the seminar!)
    - Will be available on the course website
  - plus various background reading
- Assessments:
  - 40% "literature survey" on a topic of your choice
  - 40% "practical project", applying 1 or more RE techniques
  - 10% oral presentation on one or other of the above
  - 10% class discussion (lead a discussion on weekly reading)

ARISE Video-conferencing

- ARISE is a collaborative venture
  - IBM Toronto Lab and the Universities of Waterloo, Toronto and York
  - See http://www.softwareresearch.ca/ for details
- Challenges:
  - Interaction
    - Laptops and instant messaging during the class?
  - Community building
    - Name tags, exchange bios, photos, web addresses
- ARISE research questions
  - How to collect, archive and index the classes?
    - Communications from the instructor (syllabus, assignments, lecture notes)
    - Postings to a Wiki (or similar online discussion tool)
    - Audio+Video of the seminars
    - Instant messaging during the seminars
    - Emails
  - What would you give your consent to?

Syllabus

- Introductory stuff
  - What are Requirements?
  - What is Engineering?
  - What is a System?
- Basic RE activities
  - Planning and Eliciting Requirements
  - Modelling and Analysing Requirements
  - Communicating and Agreeing Requirements
  - Realizing and Evolving Requirements
- Advanced Topics
  - Inconsistency and Uncertainty in RE
  - Use of Formal Methods in RE
  - Research methodology for RE

(I) Introductory Stuff

- What are Requirements?
  - Scope (for this course): "Software-intensive Systems"
  - Separating the Problem from the Solution
  - What Requirements Engineers do
- What is Engineering?
  - Engineering as a profession
  - Engineering projects
  - Engineering lifecycles
  - Engineering design
- What is a System?
  - General systems theory
  - Formal foundations of software systems
  - Conceptual foundations of information systems
  - Empirical foundations of human activity systems
  - Observability of systems

(II) Eliciting and Planning

- Elicitation Targets
  - Stakeholders & User Classes
  - System boundaries
  - Goals
  - Scenarios
- Elicitation techniques
  - Interviews, questionnaires, surveys, meetings
  - Prototyping
  - Ethnographic techniques
  - Knowledge elicitation techniques
  - Conversation Analysis
  - Text Analysis
- The Feasibility Study
  - Types of Feasibility
  - Cost/benefit analysis
- Risk Analysis
  - Identifying and managing risk

(III) Modelling & Analysing

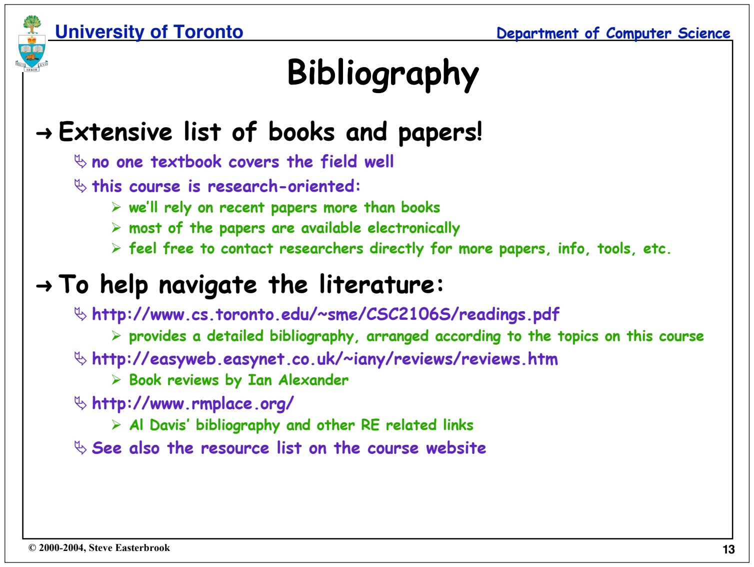
- Basics of modelling
  - Notations and their uses
  - Formality and Expressiveness
  - Abstraction and Decomposition
  - Model management and viewpoints
  - Types of Analysis
- Enterprises
  - Business rules and organisational structures
  - Goals, tasks and responsibilities
  - Soft Systems analysis
- Information Structures
  - Entities and Relationships
  - Classes and Objects
  - Domain Ontologies
- Behaviour
  - Activities and Interactions
  - States and Transitions
  - Concurrency
- Quality Requirements
  - Taxonomies of NFRs
  - Performance
  - Usability
  - Safety
  - Security
  - Reliability
  - Maintainability

(IV) Communicating & Agreeing

- Validation
  - Refutable descriptions
  - Role of contracts and procurement
  - Role of organisational politics
- Documenting Requirements
  - Properties of a good specification
  - Documentation standards
  - Specification languages
  - Making requirements testable
- Prototyping and Walkthroughs
  - Throwaway prototyping
  - Operational prototyping
  - Walkthroughs of operational models
- Reviews and Inspections
  - Effectiveness of Inspection
  - Conducting an Inspection
  - Collaborative Requirements Workshops
- Negotiation and Prioritization
  - Representing argumentation and rationale
  - Computer-supported negotiation
  - Trade-off analysis
  - Release planning

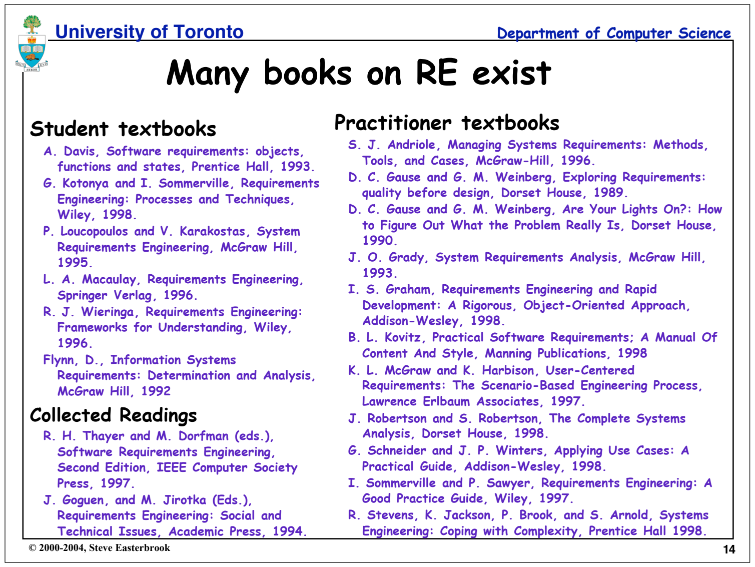
(V) Realizing and Evolving

- Software Evolution
  - Laws of evolution
  - Release planning
  - Product families
  - Requirement Reuse
- Requirements and Architectures
  - Architectural Patterns and Description Languages
  - Mapping requirements to architectures
  - Architectural Robustness
- Managing Change
  - Baselines and change requests
  - Configuration management and version control
  - Impact Analysis
- Traceability and Rationale
  - Pre- and Post- traceability
  - Capturing Design Rationale
  - Traceability techniques
- Managing Inconsistency
  - On the inevitable intertwining of inconsistency and change
  - Learning from inconsistency
  - Feature interaction
  - Living with inconsistency



Bibliography

- Extensive list of books and papers!
  - no one textbook covers the field well
  - this course is research-oriented:
    - we'll rely on recent papers more than books
    - most of the papers are available electronically
    - feel free to contact researchers directly for more papers, info, tools, etc.
- To help navigate the literature:
  - http://www.cs.toronto.edu/~sme/CSC2106S/readings.pdf
    - provides a detailed bibliography, arranged according to the topics on this course
  - http://easyweb.easynet.co.uk/~iany/reviews/reviews.htm
    - Book reviews by Ian Alexander
  - http://www.rmplace.org/
    - Al Davis' bibliography and other RE related links
  - See also the resource list on the course website



Many books on RE exist

Student textbooks

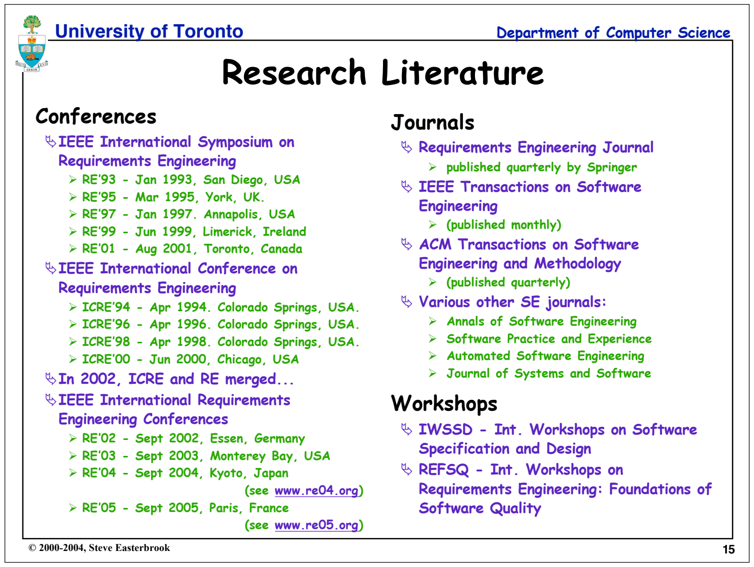
A. Davis, Software requirements: objects, functions and states, Prentice Hall, 1993.
G. Kotonya and I. Sommerville, Requirements Engineering: Processes and Techniques, Wiley, 1998.
P. Loucopoulos and V. Karakostas, System Requirements Engineering, McGraw Hill, 1995.
L. A. Macaulay, Requirements Engineering, Springer Verlag, 1996.
R. J. Wieringa, Requirements Engineering: Frameworks for Understanding, Wiley, 1996.
Flynn, D., Information Systems Requirements: Determination and Analysis, McGraw Hill, 1992

Collected Readings

R. H. Thayer and M. Dorfman (eds.), Software Requirements Engineering, Second Edition, IEEE Computer Society Press, 1997.
J. Goguen, and M. Jirotka (Eds.), Requirements Engineering: Social and Technical Issues, Academic Press, 1994.

Practitioner textbooks

S. J. Andriole, Managing Systems Requirements: Methods, Tools, and Cases, McGraw-Hill, 1996.
D. C. Gause and G. M. Weinberg, Exploring Requirements: quality before design, Dorset House, 1989.
D. C. Gause and G. M. Weinberg, Are Your Lights On?: How to Figure Out What the Problem Really Is, Dorset House, 1990.
J. O. Grady, System Requirements Analysis, McGraw Hill, 1993.
I. S. Graham, Requirements Engineering and Rapid Development: A Rigorous, Object-Oriented Approach, Addison-Wesley, 1998.
B. L. Kovitz, Practical Software Requirements; A Manual Of Content And Style, Manning Publications, 1998
K. L. McGraw and K. Harbison, User-Centered Requirements: The Scenario-Based Engineering Process, Lawrence Erlbaum Associates, 1997.
J. Robertson and S. Robertson, The Complete Systems Analysis, Dorset House, 1998.
G. Schneider and J. P. Winters, Applying Use Cases: A Practical Guide, Addison-Wesley, 1998.
I. Sommerville and P. Sawyer, Requirements Engineering: A Good Practice Guide, Wiley, 1997.
R. Stevens, K. Jackson, P. Brook, and S. Arnold, Systems Engineering: Coping with Complexity, Prentice Hall 1998.



Research Literature

Conferences

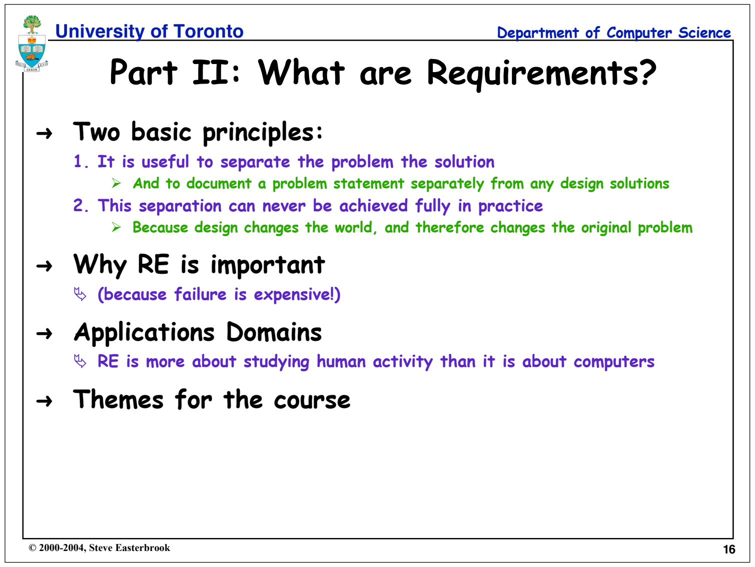
- IEEE International Symposium on Requirements Engineering
  - RE'93 - Jan 1993, San Diego, USA
  - RE'95 - Mar 1995, York, UK.
  - RE'97 - Jan 1997. Annapolis, USA
  - RE'99 - Jun 1999, Limerick, Ireland
  - RE'01 - Aug 2001, Toronto, Canada
- IEEE International Conference on Requirements Engineering
  - ICRE'94 - Apr 1994. Colorado Springs, USA.
  - ICRE'96 - Apr 1996. Colorado Springs, USA.
  - ICRE'98 - Apr 1998. Colorado Springs, USA.
  - ICRE'00 - Jun 2000, Chicago, USA
- In 2002, ICRE and RE merged...
- IEEE International Requirements Engineering Conferences
  - RE'02 - Sept 2002, Essen, Germany
  - RE'03 - Sept 2003, Monterey Bay, USA
  - RE'04 - Sept 2004, Kyoto, Japan (see www.re04.org)
  - RE'05 - Sept 2005, Paris, France (see www.re05.org)

Journals

- Requirements Engineering Journal
  - published quarterly by Springer
- IEEE Transactions on Software Engineering
  - (published monthly)
- ACM Transactions on Software Engineering and Methodology
  - (published quarterly)
- Various other SE journals:
  - Annals of Software Engineering
  - Software Practice and Experience
  - Automated Software Engineering
  - Journal of Systems and Software

Workshops

- IWSSD - Int. Workshops on Software Specification and Design
- REFSQ - Int. Workshops on Requirements Engineering: Foundations of Software Quality

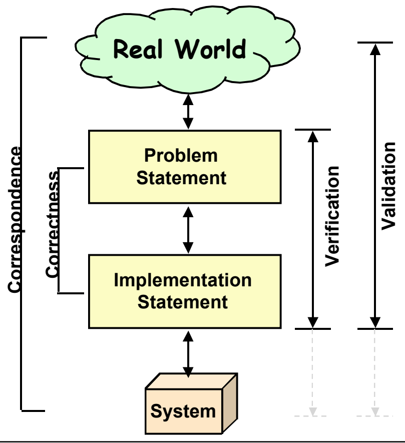


Part II: What are Requirements?

- Two basic principles:
  1. It is useful to separate the problem the solution
     - And to document a problem statement separately from any design solutions
  2. This separation can never be achieved fully in practice
     - Because design changes the world, and therefore changes the original problem
- Why RE is important
  - (because failure is expensive!)
- Applications Domains
  - RE is more about studying human activity than it is about computers
- Themes for the course

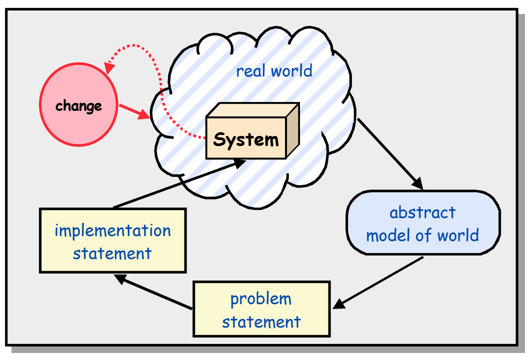
Separate the problem from the solution

- Understand the problem
  - elicitation, requirements acquisition, etc.
- Formally describe the problem
  - specification, modelling, etc.
- Attain agreement on the nature of the problem
  - validation, conflict resolution, negotiation
  - requirements management - maintain the agreement!



Source: *Adapted from Loucopoulos & Karakostas, 1995, p20 and Blum, 1992*

But design changes the world…

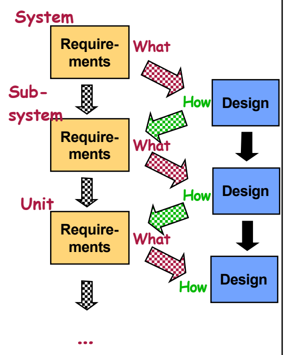


Importance of RE

- Problems
  - Increased reliance on software
    - E.g. cars, dishwashers, cell phones, web services, …
  - Software now the biggest cost element for mission critical systems
    - E.g. Boeing 777
  - Wastage on failed projects
    - E.g. 1997 GAO report: $145 billion over 6 years on software that was never delivered
  - High consequences of failure
    - E.g. Ariane 5: $500 million payload
    - E.g. Intel Pentium bug: $475 million
- Key factors:
  - Certification costs
    - E.g. Boeing 777: >40% of software budget spent on testing
  - Re-work from defect removal
    - E.g. Motorola: 60-80% of software budget (was) spent on re-work
  - Changing Requirements
    - E.g. California DMV system

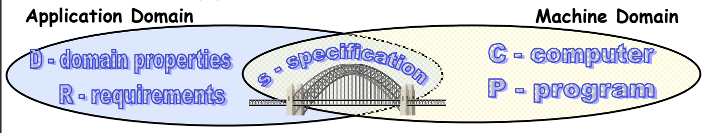
What vs. How

- Traditionally, Requirements should specify 'what' without specifying 'how'
  - But this is not always easy to distinguish:
    - What does a car do?
    - What does a web browser do?
    - What does an operating system do?
  - The 'how' at one level of abstraction forms the 'what' for the next level
- Jackson's work provides a clearer distinction
  - 'What' refers to a system's purpose
    - it is external to the system
    - it is a property of the *application domain*
  - 'How' refers to a system's structure and behavior
    - it is internal to the system
    - it is a property of the *machine domain*



Source: *Adapted from Jackson, 1995, p207*

The Application vs. The Machine



- Some distinctions:
  - Domain Properties are things in the application domain that are true whether or not we ever build the proposed system
  - Requirements are things in the application domain that we wish to be made true by delivering the proposed system
  - A specification is a description of the behaviours the program must have in order to meet the requirements
- Two verification criteria:
  - The Program running on a particular Computer satisfies the Specification
  - The Specification, in the context of the given Domain properties, satisfies the Requirements
- Two validation criteria:
  - Did we discover (and understand) all the important Requirements?
  - Did we discover (and understand) all the relevant Domain properties?

Validation Example

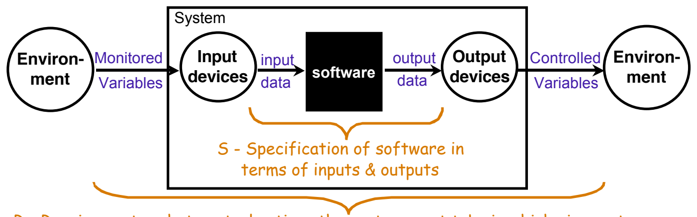
- Requirement R:
  - "Reverse thrust shall only be enabled when the aircraft is moving on the runway"
- Domain Properties D:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- Specification S:
  - Reverse thrust enabled if and only if wheel pulses on
- S + D imply R
  - But what if the domain model is wrong?

Another Example

- Requirement R:
  - "The database shall only be accessible by authorized personnel"
- Domain Properties D:
  - Authorized personnel have passwords
  - Passwords are never shared with non-authorized personnel
- Specification S:
  - Access to the database shall only be granted after the user types an authorized password
- S + D imply R
  - But what if the domain assumptions are wrong?

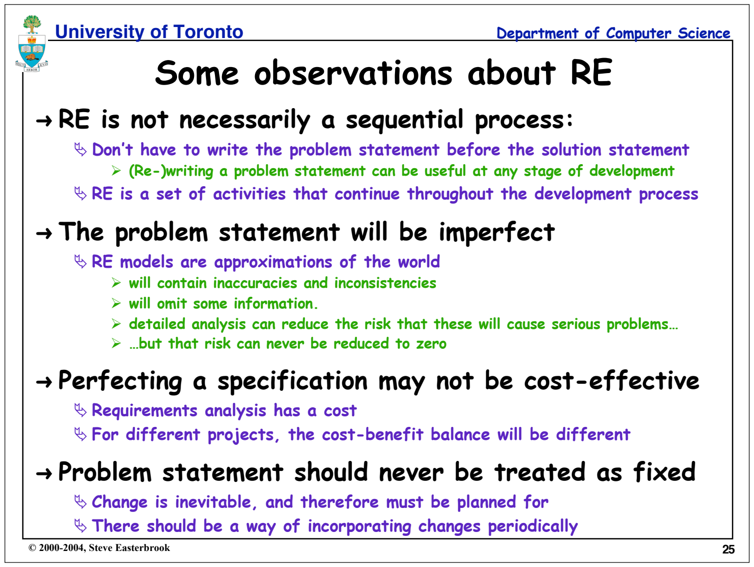
Setting the Boundaries

- How will the software interact with the world?
  - Systems engineer decides what application domain phenomena are shared
- E.g. the four variable model:
  - Decide the boundaries by designing the input/output devices
  - Uses I/O data items as proxies for the monitored and controlled variables



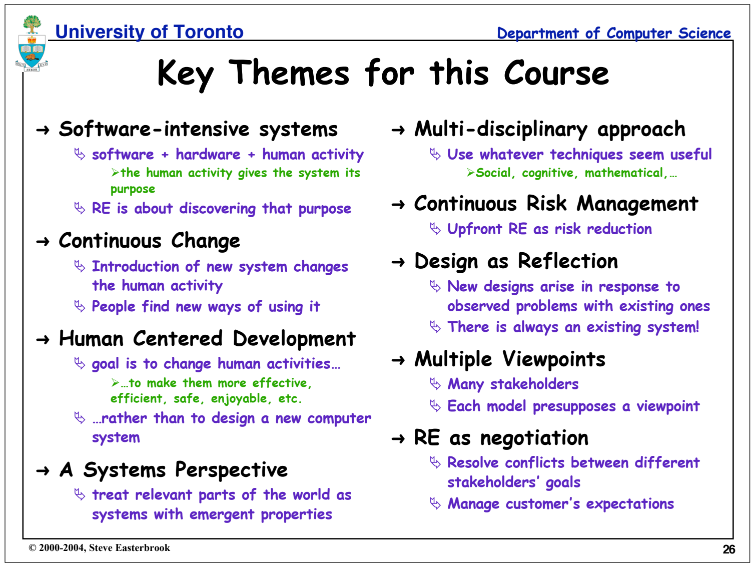
S - Specification of software in terms of inputs & outputs

R - Requirements: what control actions the system must take in which circumstances.
D - Domain Properties that constrain how the environment can behave



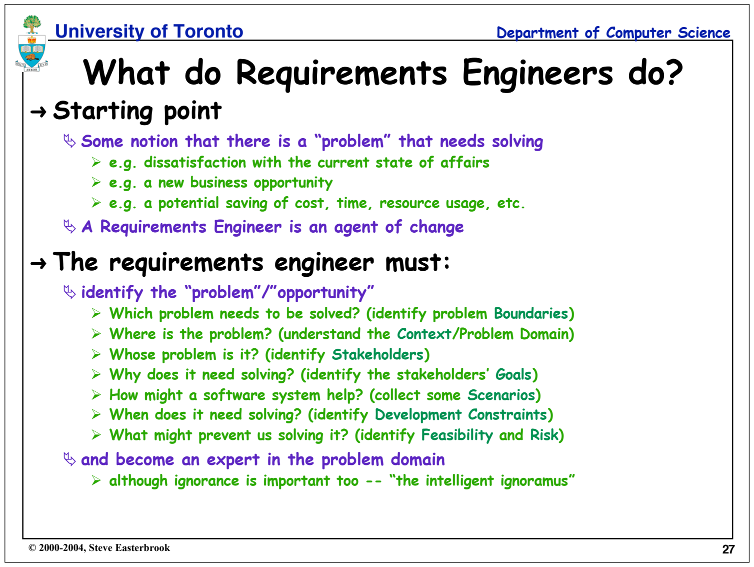
Some observations about RE

- ➜ RE is not necessarily a sequential process:
  - ✎ Don't have to write the problem statement before the solution statement
    - ➢ (Re-)writing a problem statement can be useful at any stage of development
  - ✎ RE is a set of activities that continue throughout the development process
- ➜ The problem statement will be imperfect
  - ✎ RE models are approximations of the world
    - ➢ will contain inaccuracies and inconsistencies
    - ➢ will omit some information.
    - ➢ detailed analysis can reduce the risk that these will cause serious problems…
    - ➢ …but that risk can never be reduced to zero
- ➜ Perfecting a specification may not be cost-effective
  - ✎ Requirements analysis has a cost
  - ✎ For different projects, the cost-benefit balance will be different
- ➜ Problem statement should never be treated as fixed
  - ✎ Change is inevitable, and therefore must be planned for
  - ✎ There should be a way of incorporating changes periodically



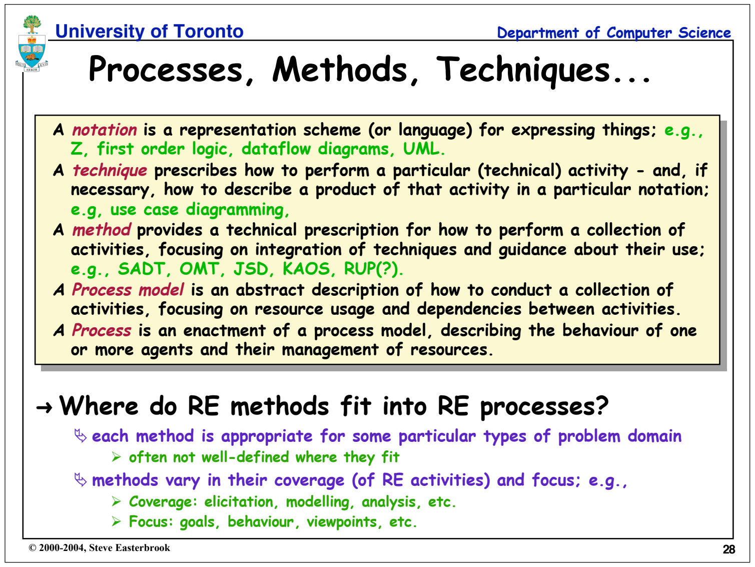
Key Themes for this Course

- ➜ Software-intensive systems
  - ✎ software + hardware + human activity
    - ➢ the human activity gives the system its purpose
  - ✎ RE is about discovering that purpose
- ➜ Continuous Change
  - ✎ Introduction of new system changes the human activity
  - ✎ People find new ways of using it
- ➜ Human Centered Development
  - ✎ goal is to change human activities…
    - ➢ …to make them more effective, efficient, safe, enjoyable, etc.
  - ✎ …rather than to design a new computer system
- ➜ A Systems Perspective
  - ✎ treat relevant parts of the world as systems with emergent properties
- ➜ Multi-disciplinary approach
  - ✎ Use whatever techniques seem useful
    - ➢ Social, cognitive, mathematical,…
- ➜ Continuous Risk Management
  - ✎ Upfront RE as risk reduction
- ➜ Design as Reflection
  - ✎ New designs arise in response to observed problems with existing ones
  - ✎ There is always an existing system!
- ➜ Multiple Viewpoints
  - ✎ Many stakeholders
  - ✎ Each model presupposes a viewpoint
- ➜ RE as negotiation
  - ✎ Resolve conflicts between different stakeholders' goals
  - ✎ Manage customer's expectations



What do Requirements Engineers 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 Engineer is an agent of change
- ➜ The requirements engineer 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"



Processes, Methods, Techniques...

A *notation* is a representation scheme (or language) for expressing things; e.g., Z, first order logic, dataflow diagrams, UML.

A *technique* prescribes how to perform a particular (technical) activity - and, if necessary, how to describe a product of that activity in a particular notation; e.g, use case diagramming,

A *method* provides a technical prescription for how to perform a collection of activities, focusing on integration of techniques and guidance about their use; e.g., SADT, OMT, JSD, KAOS, RUP(?).

A *Process model* is an abstract description of how to conduct a collection of activities, focusing on resource usage and dependencies between activities.

A *Process* is an enactment of a process model, describing the behaviour of one or more agents and their management of resources.

- ➜ Where do RE methods fit into RE processes?
  - ✎ each method is appropriate for some particular types of problem domain
    - ➢ often not well-defined where they fit
  - ✎ methods vary in their coverage (of RE activities) and focus; e.g.,
    - ➢ Coverage: elicitation, modelling, analysis, etc.
    - ➢ Focus: goals, behaviour, viewpoints, etc.