

University of Toronto Department of Computer Science

Lecture 2: What are Requirements?

- Two basic principles of requirements engineering:
 - ↳ Separate the problem from the solution
 - ↳ Problems and solutions intertwine with one another
- Describing problems:
 - ↳ Application Domains vs. Machine Domains
 - ↳ Verification vs. Validation
- Systems Engineering
 - ↳ Systems vs. software
- Patterns and Types of Problem
 - ↳ Requirements patterns
 - ↳ Problem Frames

© Easterbrook 2004 1

University of Toronto Department of Computer Science

Separate the problem from the solution

- A separate problem description is useful:
 - ↳ Most obvious problem might not be the right one to solve
 - ↳ Problem statement can be discussed with stakeholders
 - ↳ Problem statement can be used to evaluate design choices
 - ↳ Problem statement is a source of good test cases
- Still need to check:
 - ↳ Solution **correctly** solves the stated problem
 - ↳ Problem statement **corresponds** to the needs of the stakeholders

© Easterbrook 2004 2

University of Toronto Department of Computer Science

But design changes the world...

© Easterbrook 2004 3

University of Toronto Department of Computer Science

Intertwining of problems and solutions

© Easterbrook 2004 4



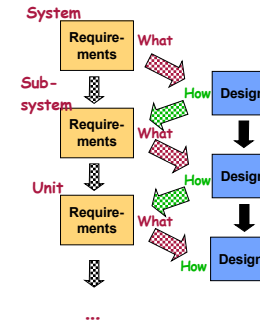
Some observations about RE

- 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 activities 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.
 - analysis should reduce the risk that these will cause serious problems...
- 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



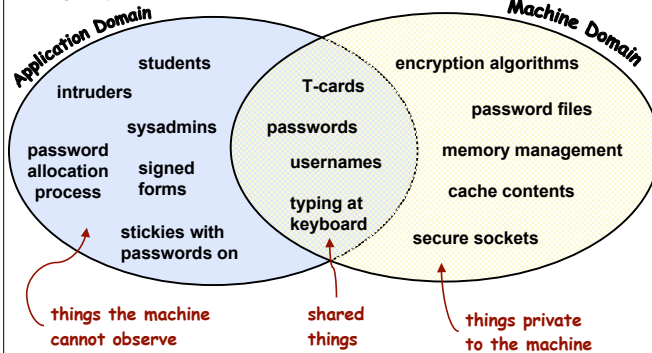
What vs. How

- Traditionally, 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
- Also misses:
 - ↳ 'Why' questions:
 - Why is this system needed?
 - Why should it behave that way?
 - ↳ 'Who' questions:
 - Whose problem is it?
 - ↳ Etc.



A problem to describe...

→ E.g. "prevent unauthorized access to CSG machines"



What are requirements?



- Domain Properties:
 - ↳ things in the application domain that are true whether or not we ever build the proposed system
- Requirements:
 - ↳ things in the application domain that we wish to be made true by delivering the proposed system
 - Many of which will involve phenomena the machine has no access to
- A Specification:
 - ↳ is a description of the behaviours that the program must have in order to meet the requirements
 - Can only be written in terms of shared phenomena!

University of Toronto Department of Computer Science

Fitness for purpose?

→ **Two correctness (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 completeness (validation) criteria:**

- ↳ We discovered all the important requirements
- ↳ We discovered all the relevant domain properties

→ **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
- ↳ **Verification:** $S, D \models R$

© Easterbrook 2004 9

University of Toronto Department of Computer Science

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 entail R**

- ↳ But what if the domain assumptions are wrong?

© Easterbrook 2004 10

University of Toronto Department of Computer Science

But we can also move the boundaries...

→ E.g. Elevator control system:

→ **We can shift things around:**

- ↳ E.g. Add some sensors to detect when people are waiting
- ↳ This changes the nature of the problem to be solved

© Easterbrook 2004 11

University of Toronto Department of Computer Science

Systems vs. Software Engineering

REQ (the requirements - relationships between monitored and controlled variables that the **system** is required to establish or maintain)

NAT (natural relationships between monitored and controlled variables that are part of the domain)

© Easterbrook 2004 12

University of Toronto Department of Computer Science

Example Problem Frames

→ **Required behaviour**

- Problem: build a machine to control part of the world in accordance with a fixed set of control rules
- Likely Solution: an automated control system

→ **Commanded Behaviour**

- Problem: build a machine that allows part of the world to be controlled by an operator by issuing commands
- Likely Solution: a "human-in-the-loop" control system.

→ **Information Display**

- Problem: provide information about the current state of part of the world, in response to information requests
- Likely Solution: an information system.

© Easterbrook 2004 13

University of Toronto Department of Computer Science

More problem frames

→ **Simple workpieces frame**

- Problem: keep track of the edits performed on some workpiece, e.g a text file or a graphical object
- Likely Solution: application software (e.g. a word processor)

→ **Transformation frame**

- Problem: take input data in a certain format, and provide a transformation according to a certain set of rules
- Example Solutions: data processing applications; compilers, etc.

→ **Connection frame**

- Problem: maintain a correspondence between domains that are otherwise not connected
- Example Solutions: data entry system, sensor network, etc.

© Easterbrook 2004 14

University of Toronto Department of Computer Science

Summary

→ **Requirements Engineering is about describing problems**

- It is useful to separate the problem from the solution
 - Even though this cannot be achieved entirely
- Problems evolve continuously:
 - Delivering a solution changes the problem
 - Describing the problem changes the problem

→ **Key distinctions:**

- Application Domains vs. Machine Domains
- Verification vs. Validation
- Systems Engineering vs. Software Engineering

→ **Basic Problem Frames**

- Give us a starting point for understanding the problem
- Tell us what subdomains we need to describe

© Easterbrook 2004 15