lecture 7: requirements analysis

csc302h
winter 2014

Computer Science
UNIVERSITY OF TORONTO

recap from last time

- managing risk in the context of software projects: assessment + control
- risk exposure:
  - RE = probability × consequences (loss)
- risk reduction leverage:
  - RRL = (RE_{before} – RE_{after}) ÷ cost of mitigating action
  - for ROI calculations and comparison
- risk assessment
  - quantitative (if you can)
  - qualitative (risk exposure matrix)

Computer Science
UNIVERSITY OF TORONTO

recap from last time (2)

- discussed some of the most common software engineering risks & countermeasures
- case studies (& lessons)
  - failed mars missions
  - therac-25 (from risks digest forum)
  - annoying oil tank with a phone!
- don't have iv&v report to the development manager (conflict of interest, that's what the "i" is for)
- principles of risk management

Requirements Analysis

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
  - …or we are using the software for a purpose different from the intended one
- Requirements analysis is about identifying this purpose
- Inadequate understanding of the purpose leads to poor quality software

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

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 organizational context
  - There are multiple stakeholders with different values and goals
  - Software design is part of an ongoing learning process by the organization
  - 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

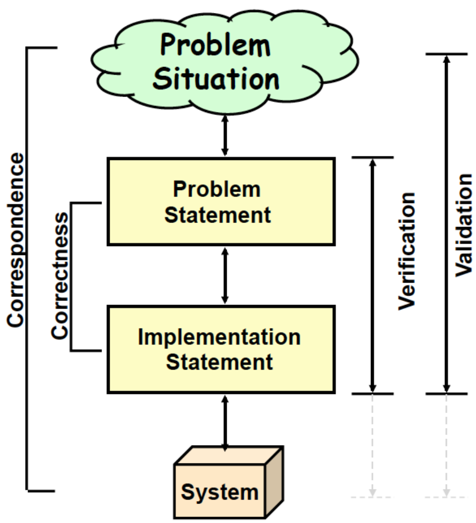
Separate the problem from the solution

A separate problem description is useful:

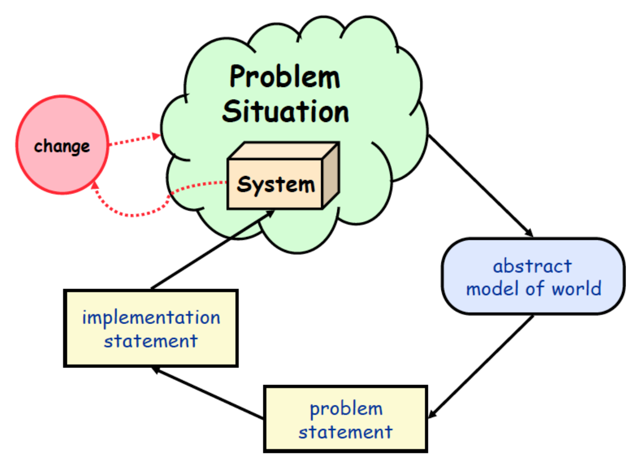
- It can be discussed with stakeholders
- It can be used to evaluate design choices
- It is a good source of test cases
- Note: Most obvious problem might not the right one to solve

Still need to check:

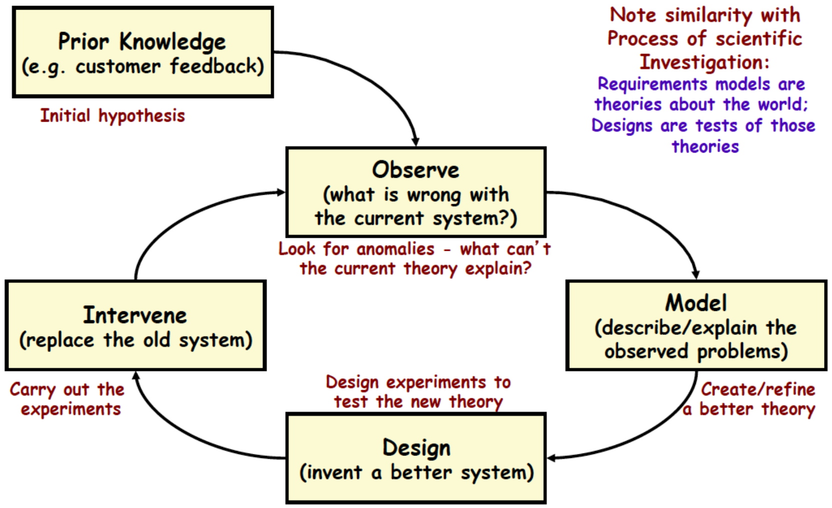
- Solution correctly solves the stated problem (verification)
- Problem statement corresponds to the needs of the stakeholders (validation)



But design changes the world…

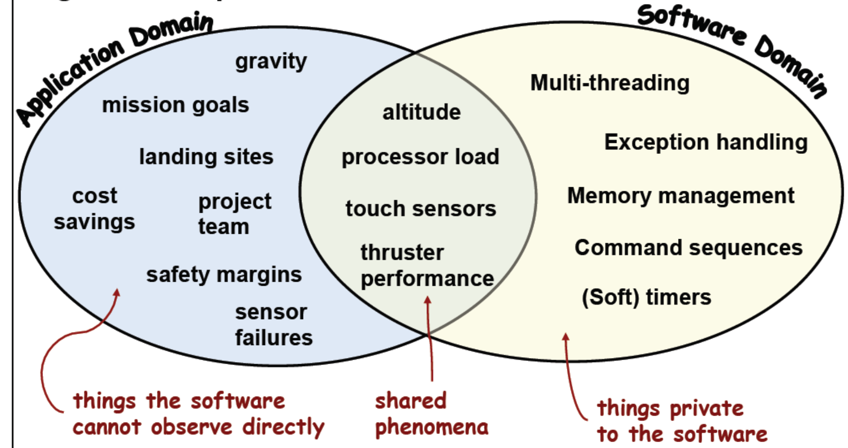


Requirements as Theories

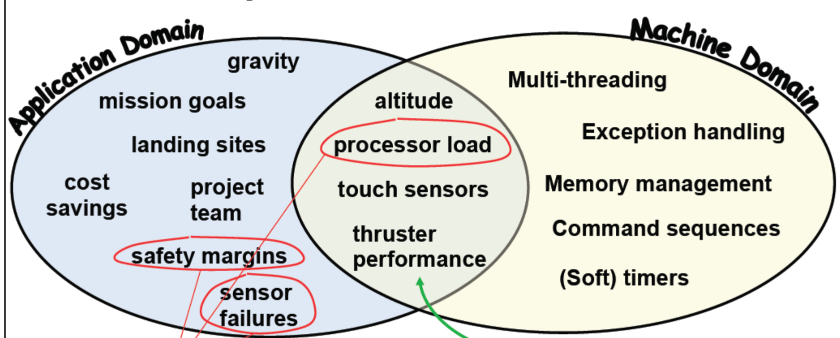


A problem to describe…

E.g. "land a spacecraft on Mars"



A problem to describe…



Don't overload the processors…
Don't use data from failed sensors…
Ignore noise on sensors when legs unfold…

→

Poll multiple sensors continually and compare results to test sensor function.
Start using touchdown sensors at 12m above the surface
(Assumes legs have finished unfolding by then…)

Thinking about Software Requirements

Domain Properties (assumptions):
- things in the application domain that are true, whether or not we ever build the proposed system

(System) Requirements:
- things in the application domain that we wish to be made true, by delivering the proposed system
  - May involve phenomena to which the machine has no access

A (Software) Specification:
- 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!

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 appropriateness (validation) criteria:

- We discovered all the important requirements
- We properly understood 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 $\Rightarrow$ R

Another Example

Requirement R:

- "The database shall only be accessible by authorized personnel"

Domain Properties D:

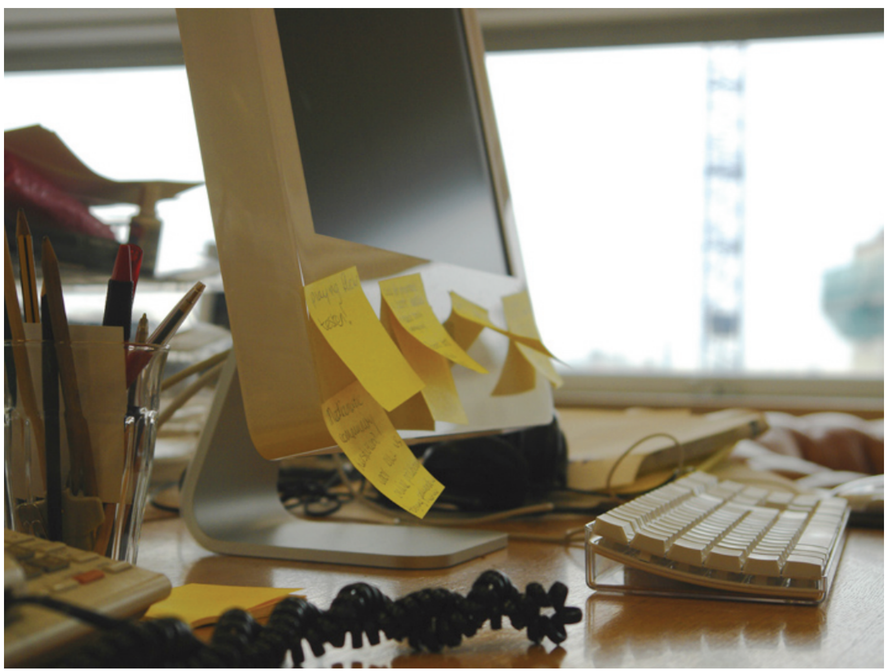
- Authorized personnel have passwords
- Passwords are never shared with non-authorized personnel

Specification S:

- "Access to the database shall only be granted after the user types an authorized password"

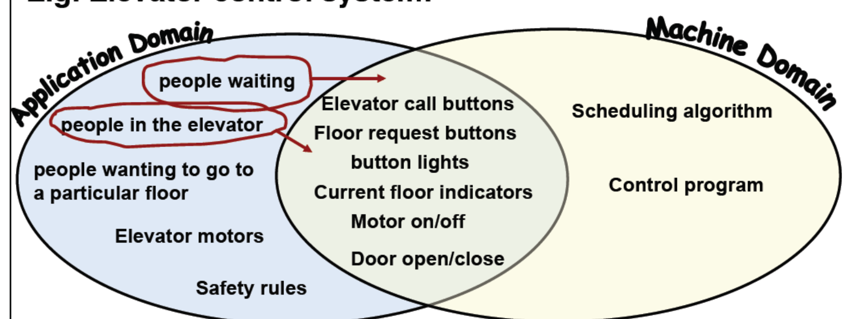
S, D $\Rightarrow$ R

- But what if the domain assumptions are wrong?



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

Observations

Analysis 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.
  - assess 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
- Depends on the consequences of getting it wrong!

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

Stakeholders

Stakeholder analysis:

- Identify all the people who must be consulted during information acquisition

Example stakeholders

- Users
  - concerned with the features and functionality of the new system
- Customers
  - Wants to get best value for money invested!
- Business analysts / marketing team
  - want to make sure "we are doing better than the competition"
- Training and user support staff
  - want to make sure the new system is usable and manageable
- Technical authors
  - will prepare user manuals and other documentation for the new system
- Systems analysts
  - want to "get the requirements right"
- Designers
  - want to build a perfect system, or reuse existing code
- The project manager
  - wants to complete the project on time, within budget, with all objectives met.

Identifying Stakeholders' Goals

Source: Adapted from Anton, 1996.

Approach

- Focus on *why* a system is required
- Express the 'why' as a set of stakeholder goals
- Use goal refinement to arrive at specific requirements
- Goal analysis
  - document, organize and classify goals
- Goal evolution
  - refine, elaborate, and operationalize goals
- Goal hierarchies show refinements and alternatives

Advantages

- Reasonably intuitive
- Explicit declaration of goals provides sound basis for conflict resolution

Disadvantages

- Captures a static picture - what if goals change over time?
- Can regress forever up (or down) the goal hierarchy

Goal Modeling

(Hard) Goals:

- Describe functions that must be carried out. E.g.
  - Satisfaction goals
  - Information goals

Softgoals:

- Cannot really be fully satisfied. E.g.
  - Accuracy
  - Performance
  - Security
  - …

Types of goal:

- Achieve/Cease goals
  - Reach some desired state eventually
- Maintain/Avoid goals
  - Keep some property invariant
- Optimize
  - A criterion for evaluating design choices

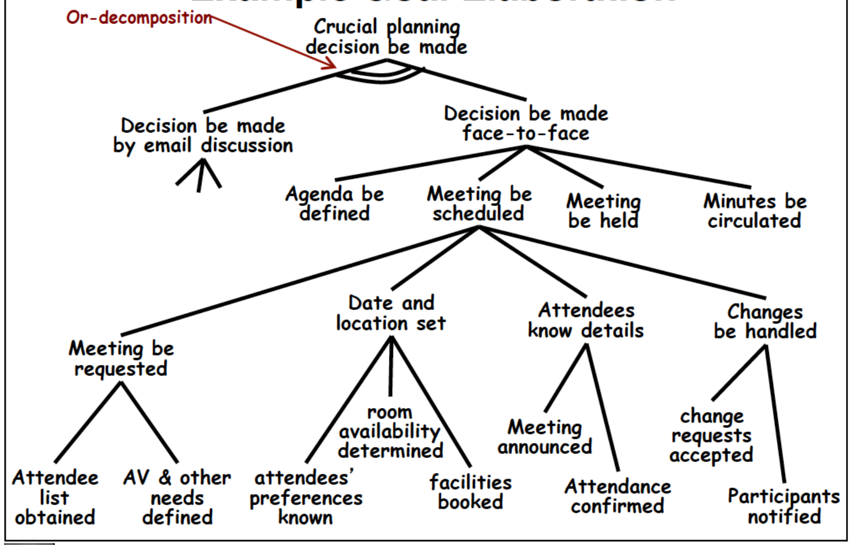
Agents:

- Owners of goals
- Choice of when to ascribe goals to agents:
  - Identify agents first, and then their goals
  - Identify goals first, and then allocate them to agents during operationalization

Modelling Tips:

- Multiple sources yield better goals
- Associate stakeholders with each goal
  - reveals viewpoints and conflict
- Use scenarios to explore how goals can be met
- Explicit consideration of obstacles helps to elicit exceptions

Example Goal Elaboration



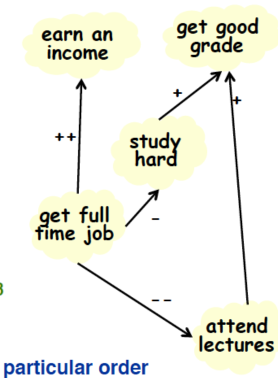
Goal Analysis

Goal Elaboration:

- "Why" questions explore higher goals (context)
- "How" questions explore lower goals (operations)
- "How else" questions explore alternatives

Relationships between goals:

- One goal helps achieve another (+)
- One goal hurts achievement of another (-)
- One goal makes another (++)
  - Achievement of goal A guarantees achievement of goal B
- One goal breaks another (--)
  - Achievement of goal A prevents achievement of goal B
- Precedence ordering – if goals must be achieved in a particular order



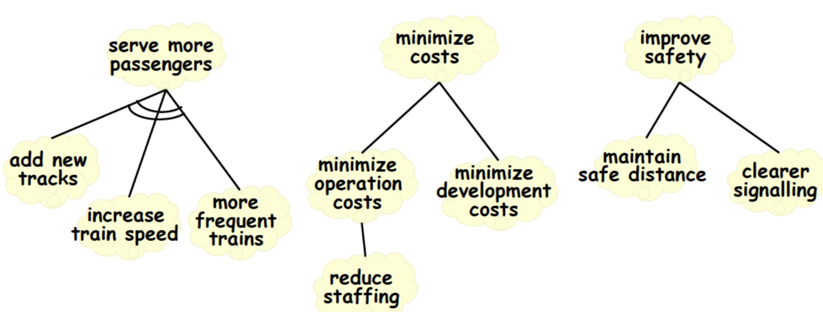
Obstacle Analysis:

- Can this goal be obstructed, if so how?
- What are the consequences of obstructing it?

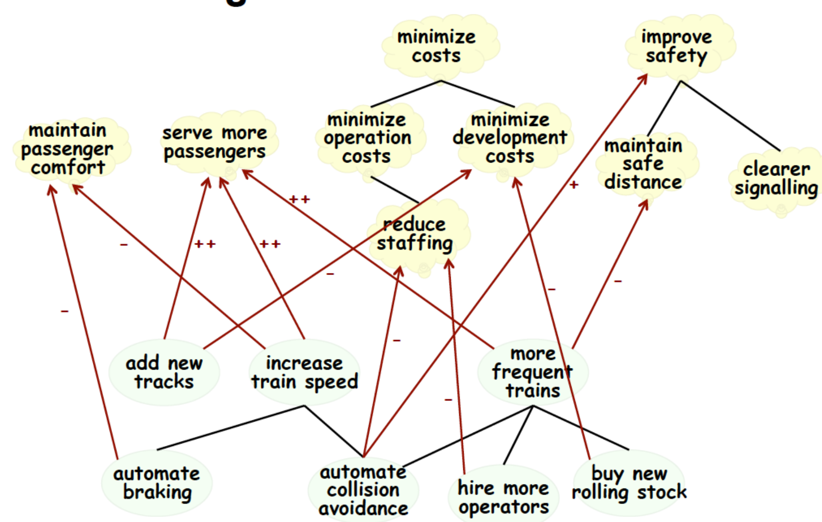
Softgoals

Some goals can never be fully satisfied

- Treat these as softgoals
  - E.g. "system be easy to use"; "access be secure"
  - Also known as 'non-functional requirements'; 'quality requirements'
- Will look for things that contribute to satisficing the softgoals
- E.g. for a train system:



Softgoals as selection criteria



next week:

from requirements to design