Lecture 2: The Requirements/Design Gap

Kenneth M. Anderson

Object-Oriented Analysis and Design

CSCI 6448 - Spring Semester, 2001

Goals for this Lecture

- Review definition of software engineering
- Discuss requirements engineering
- Discuss requirements analysis
- Discuss requirements/design gap

What is "Software Engineering"

- Software
  - Computer programs and their related artifacts
- Engineering
  - The application of scientific principles in the context of practical constraints

A Definition (Daniel M. Berry)

- Software engineering is that form of engineering that applies:
  - a systematic, disciplined, quantifiable approach,
  - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines,
- to creating, developing, operating, and maintaining cost-effective, reliably correct, high-quality solutions to software problems.

Engineers Build Machines

- Software is intangible
  - Descriptions of a desired machine, written according to specific languages and notations
- Computer is tangible
  - General-purpose description executor
  - Behaves as if it were the desired machine
- Software engineers "build" descriptions
  - Organizing, structuring, and making complex assemblages of descriptions
  - Raw materials: languages and notations

Basic Software Engineering Activities

- Create
  - Modeling
- Record
  - Specification
- Analyze
  - Verification & Validation
- Configure
  - Selection, Translation, & Deployment

IEEE definition of requirement

- A condition or capacity needed by a user to solve a problem or achieve an objective
- A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents
- A documented representation of a condition or capability as in 1 or 2

Requirements Engineering

- "The systematic process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained."
  - K. Pohl, 1993

Questions to consider

- Can one be systematic in the face of vaguely understood requirements?
- Can one know whether the requirements are complete in the context of iteration?
- How do you define cooperation among agents?
- What representation formalisms can be used?
- How can a genuine shared understanding be reached?

Two sides to Requirements Engineering

- Requirements Elicitation
  - The process whereby a development agency discovers what is needed and why
  - Uses knowledge elicitation techniques
    - ethnomethodology, human factors, ergonomics, etc.
- Requirements Analysis
  - The process of understanding the requirements
  - Asks questions about completeness and consistency
  - Uses formal methods of systems analysis

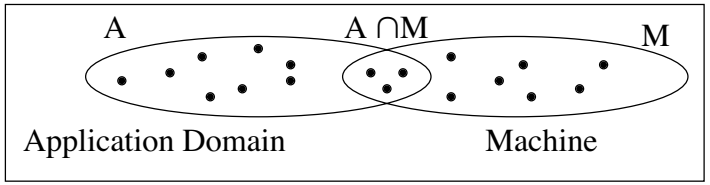
Requirements Analysis

- Understanding the phenomena of the application domain
- Describing the required relationships among the phenomena
- Example: Elevator Controller
  - Phenomena concern the application domain, not the (software) machine that controls it
    - buttons being pressed, buttons lighting up, cars moving in directions, doors opening and closing, people entering and leaving

Design

- Creating a machine that satisfies the requirements
  - Machine ensures satisfaction by sharing phenomena with application domain
    - shared events occur in both domains
    - shared states visible in both domains
- Example: Elevator Controller
  - “Press up button on floor 3” ≈ “Signal on line 3U”
  - “Car at floor 3” ≈ “Floor_Sensor_State[3] = 1”

Application versus Machine Phenomena



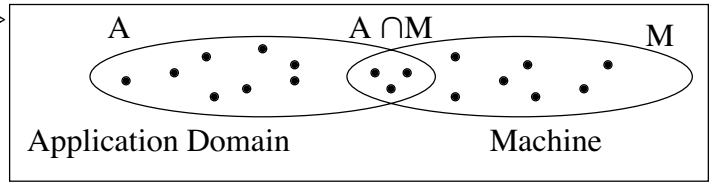
- Not all phenomena are shared
- Creates requirements/design gap
- Example: Elevator Controller
  - Car movement while between sensors
  - Correspondence of person pushing button to person exiting

Does System Satisfy Requirements?

R(equirements) ⇒ A

P(rogram) ⇒ M

S(pec.) ⇒ A ∩ M

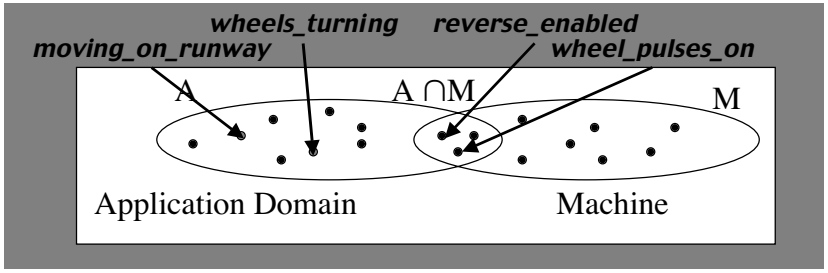


① If computer behaves as P, then S satisfied
- C,P ↦S, where C are the properties of the computer

② If S satisfied, then R must be satisfied
- D,S ↦R, where D are the properties of the application domain

Understanding Domain is Critical

- Example: Automated Thrust Reverser
  - Requirement
    - *reverse_enabled* IFF *moving_on_runway*
  - Domain Properties Assumed by Developers
    - *wheel_pulses_on* IFF *wheels_turning*
    - *wheels_turning* IFF *moving_on_runway*



Domain Misunderstandings ⇛ Errors

- Example: Automated Thrust Reverser
  - Derived Interface Specification
    - *reverse_enabled* IFF *wheel_pulses_on*
  - Domain Properties Assumed by Developers
    - ✓ *wheel_pulses_on* IFF *wheels_turning*
    - ✗ *wheels_turning* IFF *moving_on_runway*
  - Aquaplaning Wheels
    - *moving_on_runway* is TRUE
    - *wheels_turning* is FALSE