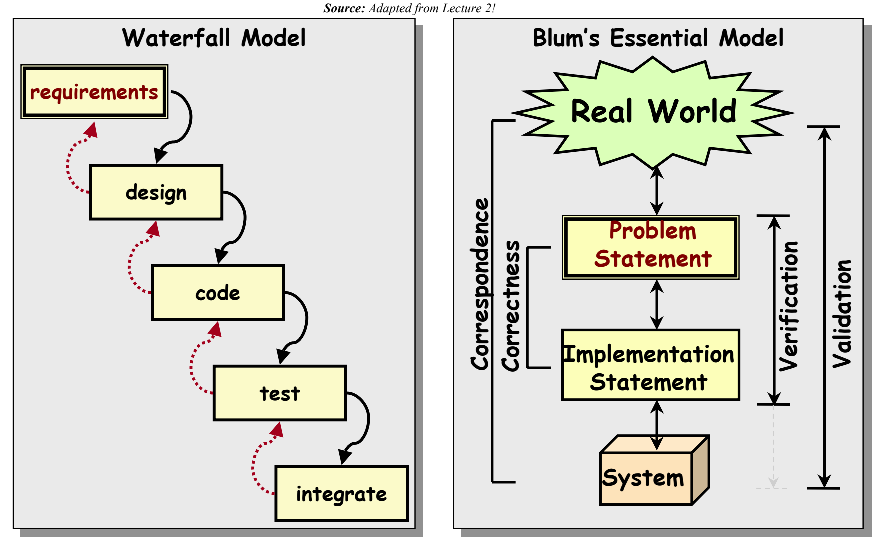
Lecture 14: Requirements Analysis

- → Basic Requirements Process
  - requirements in the software lifecycle
  - the essential requirements process
- → What is a requirement?
  - What vs. How
  - Machine Domain vs. Application Domain
  - Implementation Bias
- → Non-functional Requirements
- → Notations, Techniques and Methods
  - Elicitation techniques
  - Modeling methods

Refresher: Software Lifecycles

Source: *Adapted from Lecture 2!*



Basics of Requirements Engineering

Source: *Adapted from Nuseibeh & Easterbrook, 2000*

- → The 'essential' requirements process:
  - Understand the problem
    - use data gathering techniques to elicit requirements
    - Eg. Interviews, Questionnaires, Focus Groups, Prototyping, Observation,…
  - Model and Analyze the problem
    - use some modeling method(s)
    - Eg. Structured Analysis, Object Oriented Analysis, Formal Analysis,…
  - Attain agreement on the nature of the problem
    - validation
    - conflict resolution, negotiation
  - Communicate the problem
    - specifications, documentation, review meetings,
  - Manage change as the problem evolves
    - Requirements continue to evolve throughout software development
    - (introducing new software changes the problem!!!)
    - requirements management - maintain the agreement!

RE is the weak link in most projects

- → Requirements Engineering is hard *("wicked")*:
  - Analysis problems have ill-defined boundaries (open-ended)
  - Requirements are found in organizational contexts (hence prone to conflict)
  - Solutions to analysis problems are artificial
  - Analysis problems are dynamic
  - Tackling analysis requires interdisciplinary knowledge and skill
- → Requirements Engineering is important:
  - Engineering is about developing solutions to problems
    - A good solution is only possible if the engineer fully understands the problem
  - Errors cost more the longer they go undetected
    - Cost of correcting a requirements error is 100 times greater in the maintenance phase than in the requirements phase
  - Experience from failed software development projects:
    - Failure to understand and manage requirements is the biggest single cause of cost and schedule over-runs
  - Analysis of safety problems
    - Safety-related errors tend to be errors in specifying requirements, while non-safety errors tend to be errors in implementing requirements

What vs. How

Source: Adapted from Jackson, 1995, p207 and van Vliet 1999, p204-210

- ***Requirements should specify 'what' but not 'how'***
  - But this is not so easy to distinguish:
    - What does a car do?
    - What does a web browser do?
  - '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*
- **Requirements *only* exist in the application domain**
  - Distinguishing between the machine and the application domain is essential for good requirements engineering
  - Need to draw a boundary around the application domain
    - I.e. which things are part of the problem you are analyzing and *which are not?*

Implementation Bias

Source: Adapted from Jackson, 1995, p98

- **Implementation bias is the inclusion of requirements that have no basis in the application domain**
  - i.e. mixing some 'how' into the requirements
- **Examples:**
  - The dictionary shall be stored in a hash table
  - The patient records shall be stored in a relational database
- **But sometimes it's not so clear:**
  - The software shall be written in FORTRAN.
  - The software shall respond to all requests within 5 seconds.
  - The software shall be composed of the following 23 modules ....
  - The software shall use the following fifteen menu screens whenever it is communicating with the user.…
- **Instead of 'what' and 'how', ask:**
  - is this requirement only a property of the machine domain?
    - in which case it is implementation bias
  - Or is there some application domain phenomena that justifies it?

Functional vs. Non-functional

Source: Adapted from van Vliet 1999, p241-2

- **"Functional Requirements"**
  - fundamental functions of the system
    - E.g. mapping of inputs to outputs
    - E.g. control sequencing
    - E.g. timing of functions
    - E.g. handling of exceptional situations
    - E.g. formats of input and output data (and stored data?)
    - E.g. real world entities and relationships modeled by the system
- **"Non-Functional Requirements (NFRs)"**
  - constraints/obligations (non-negotiable)
    - E.g. compatibility with (and reuse of) legacy systems
    - E.g. compliance with interface standards, data formats, communications protocols
  - quality requirements (soft goals)
    - E.g. security, safety, availability, usability, performance, portability,…
    - must be specified

Elicitation Techniques

Source: Adapted from Nuseibeh & Easterbrook, 2000 and van Vliet 1999, section 9.1.1

- **Traditional Approaches**
  - Introspection
  - Interview/survey
  - Group elicitation
- **Observational approaches**
  - Protocol analysis
  - Participant Observation (ethnomethodology)
- **Model-based approaches**
  - Goal-based: hierarchies of stakeholders' goals
  - Scenarios: characterizations of the ways in which the system is used
  - Use Cases: specific instances of interaction with the system
- **Exploratory approaches**
  - Prototyping ("plan to throw one away")

Modeling: Notations vs. Methods

➜ Definitions:

- **Notation**: a systematic way of presenting something
  - may be linguistic (textual) or graphical (diagrams)
- A **Method** provides:
  - a set of notations (e.g. for different viewpoints)
  - techniques for using those notations (esp. analysis techniques)
  - heuristics to provide guidance
- Notation or method?
  - Some notations have been adopted by a number of different methods
  - Some 'methods' are really just notations
- Tools usually support a single method (or a single notation!!)

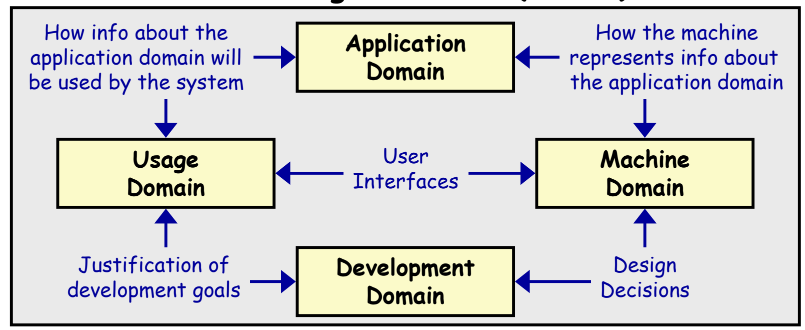
➜ Example Methods

- Structured Analysis
  - SADT
  - SASD
  - Information Engineering
  - JSD
- Entity-Relationship Approach
- Object Oriented Analysis
  - Coad-Yourdon
  - OMT
  - UML (not a method ??)
- Formal Methods
  - SCR
  - RSML

Modeling: Where to start?

Source: Adapted from Loucopoulos & Karakostas, 1995, p73

➜ There are lots of things we could (should) model:



➜ Key questions

- Where do we start?
  - Structured Analysis starts by modeling the *existing* system
  - Object Oriented Analysis starts by identifying candidate objects
- How do we structure our modeling approach?
  - We can *partition* the problem, *abstract* away detail, and create *projections*

Structuring Principle 1: Partitioning

➜ Partitioning

- captures aggregation/part-of relationship

➜ Example:

- goal is to develop a spacecraft
- partition the problem into parts:
  - guidance and navigation;
  - data handling;
  - command and control;
  - environmental control;
  - instrumentation;
  - etc
- Note: this is not a design, it is a problem decomposition
  - actual design might have any number of components, with no relation to these sub-problems
- However, the choice of problem decomposition will probably be reflected in the design

Structuring Principle 2: Abstraction

Source: Adapted from Davis, 1990, p48 and Loucopoulos & Karakostas, 1995, p78

➜ Abstraction

- A way of finding similarities between concepts by ignoring some details
- Focuses on the general/specific relationship between phenomena
  - *Classification* groups entities with a similar role as members of a single *class*
  - *Generalization* expresses similarities between different classes in an 'is_a' association

➜ Example:

- requirement is to handle faults on the spacecraft
- might group different faults into fault classes

- E.g. based on location of fault:
  - instrumentation fault,
  - communication fault,
  - processor fault,
  - etc

OR:

- E.g. based on symptoms of fault:
  - no response from device;
  - incorrect response;
  - self-test failure;
  - etc...

Structuring Principle 3: Projection

Source: Adapted from Davis, 1990, p48-51

➜ Projection:

- separates aspects of the model into multiple viewpoints
  - similar to projections used by architects for buildings

➜ Example:

- Need to model the communication between spacecraft and ground system
- Model separately:
  - sequencing of messages;
  - format of data packets;
  - error correction behavior;
  - etc.

➜ Note:

- Projection and Partitioning are similar:
  - Partitioning defines a 'part of' relationship
  - Projection defines a 'view of' relationship
- Partitioning assumes a the parts are relatively independent

References

van Vliet, H. "Software Engineering: Principles and Practice (2nd Edition)" Wiley, 1999.

Chapter 9 is an excellent introduction to the basics of requirements engineering.

B. A. Nuseibeh and S. M. Easterbrook, "Requirements Engineering: A Roadmap", In A. C. W. Finkelstein (ed) *"The Future of Software Engineering"*. IEEE Computer Society Press, 2000.

Available at http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

Jackson, M. "Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices". Addison-Wesley, 1995.

This is my favourite requirements engineering book. It makes a wonderful and thought provoking read. It consists of a series of short essays (each typically only a couple of pages long) that together really get across the message of what requirements engineering is all about.

Davis, A. M. "Software Requirements: Analysis and Specification". Prentice-Hall, 1990.

This is probably the best textbook around on requirements analysis, although is a little dated now.

Loucopoulos, P. and Karakostas, V. "System Requirements Engineering". McGraw Hill, 1995.

This short book provides a good overview of requirements engineering, especially in a systems context.