LECTURE 2: REQUIREMENTS

Software Engineering

Mike Wooldridge

1 Requirements Analysis and Spec

- Involves:
  - *feasibility study*;
  - *requirements analysis*;
  - *requirements definition*;
  - *requirements validation*;
  - *requirements specification*.
- Aim: to establish derive a complete, official statement of what developers are required to do:

  The software requirements document.

1.1 The Requirements Document

- Should specify only *external behaviour*. (Avoid *implementation bias*.)
- Should specify constraints on implementation.
- Should be easy to change.
- Should serve as a reference for system maintainers.
- Should document the expected system lifecycle.
- Should describe desired responses to unexpected inputs.

2 Requirements Analysis

The role of the analyst is:

- To *elicit* requirements.
- To resolve different views.
- To advise on what is feasible.
- To clarify requirements.
- To document requirements.
- To negotiate and gain user's agreement for the spec.

2.1 How to Get Requirements

- Talk to the user:
  - listen to needs;
  - ask for clarification;
  - record the views.
- Clarify views:
  - resolve inconsistencies;
  - generate a consensus.
- Important to involve all the *stakeholders*.

2.2 Problems with Analysis

- Stakeholders don't know what they want.
- Stakeholders may have unrealistic expectations.
- Stakeholders use their own language.
- Different stakeholders have different requirements.
- Political factors affect requirements.
- Economic/business factors create a dynamic environment.

2.3 Requirements Definition

- Requirements definition is:

  High-level, customer-oriented statement of what system is to do.

- Should be accessible to all stakeholders.

- Two types of requirements:

  – *functional*:

    services the system should provide, how it should respond to inputs, how it should behave, what it should not do;

    "The system should then display all the titles of books written by the specified author."

  – *non-functional*:

    constraints the system should operate under;

    "Should be implemented on a Pentium 450 with 64MB of RAM and 2GB hard disk."

- Should be:
  - *complete*:
    document all services to be provided;
  - *consistent*:
    not be contradictory.
  - *structured*:
    not thrown together!
  - *systematic*:
    include evidence of organisation.
  - *free of implementation bias*:
    not mandate a solution.
- Use of *natural language* leads to 3 key problems:
  - lack of clarity;
  - requirements confusion;
  - requirements amalgamation.

2.4 Non-Functional Requirements

- Speed:
  - transactions per second;
  - user/event response time;
  - screen refresh time.
- Size:
  - KBytes;
  - Number of RAM chips.
- Ease of use:
  - required average training time;
  - number of help screens.

- Reliability:
  - mean time to failure;
  - availability.
- Robustness:
  - time to restart after failure;
  - percentage of events causing failure;
  - freedom from data corruption on failure.
- Portability:
  - percentage of target-dependent statements;
  - number of target systems.

2.5 Kinds of Requirements

- Physical environment:
  - where is the equipment to function?
  - is there one location or several?
  - are there any environmental restrictions (temperature, humidity . . .)?
- Interfaces:
  - is the input coming *from* one or more other systems?
  - is the input going *to* one or more other systems?
  - is there a prescribed medium that data comes in/goes out as (e.g., floppy disk, CD ROM)?

- User and human interfaces:
  - who will use the system?
  - will there be several types of user?
  - what is the skill level of each user?
  - what training will be required for users?
  - how easy will it be for users to use/misuse the system?
- Functionality:
  - what will the system do?
  - when will the system do it?
  - are there any constraints on execution speeds, response times, or throughput?

- Documentation:
  - how much documentation is required?
  - to what audience is the document addressed?
  - what help features must be provided?
- Data:
  - what format should input/output data have?
  - how often will it be received or sent?
  - how accurate must it be?
  - to what degree of precision must calculations be carried out to?
  - how much data flows through the system?
  - must any data be retained?

- Security:
  - must access to the system be controlled?
  - how will one user's data be isolated from another's?
  - how often will the system be backed up?
  - must backup copies be stored at a separate location?
  - should precautions be taken against fire & theft?
- Quality assurance:
  - what are the requirements for reliability?
  - what is the mean time between failure?
  - what faults is the system required to catch?

3 Requirements Specification Documents

IEEE Standard 830-1984 specifies three parts:

1. Introduction
2. General Description
3. Specific Requirements

3.1 Part 1: Introduction

1. Introduction
   - 1.1 Purpose
   - 1.2 Scope
   - 1.3 Definitions, acronyms, abbreviations
   - 1.4 References
   - 1.5 Overview

3.2 Part 2: General Description

2. General Description
   - 2.1 Product perspective
   - 2.2 Product functions
   - 2.3 User characteristics
   - 2.4 General constraints
   - 2.5 Assumptions and dependencies

3.3 Part 3: Specific Requirements

3. Specific Requirements
   - 3.1 Functional requirements
     - 3.1.1 Functional requirement 1
       3.1.1.1 Introduction
       3.1.1.2 Inputs
       3.1.1.3 Processing
       3.1.1.4 Outputs
     - 3.1.2 Functional requirement 2
       - * . . .
     - 3.1.n Functional requirement n

- 3.2 External interface requirements
  - 3.2.1 User interfaces
  - 3.2.2 Hardware interfaces
  - 3.2.3 Software interfaces
  - 3.2.4 Communications interfaces
- 3.3 Performance requirements
- 3.4 Design constraints

3.5 Attributes

3.5.1 Security

3.5.2 Maintainability

3.6 Other requirements

4 Problems with Requirements

- *Noise*:

  meaningless or irrelevant information.
- *Silence*:

  missing elements.
- *Overspecification/implementation bias*:

  telling the designer how to do their job.
- *Contradiction*:

  when two descriptions of the same thing differ.
- *Unsatisfiability*:

  specifying something impossible.
- *Ambiguity*:

  not being precise.
- *Wishful thinking*:

  when unrealistic demands are made.

5 Requirements Validation

- The process of showing that requirements define the systems the customer wants.
- Invalid requirements are *very* expensive!
- Need to check that requirements are:
  - *complete*;
  - *correct*.
- *Prototyping* is a valuable validation tool.
  Particularly useful for GUI-based systems.

- Periodic *requirements reviews* are another important technique.
- Requirements reviews checks for:
  - *Verifiability*:
    is the requirement realistically testable?
  - *Comprehensibility*:
    is the requirement understood by procurers and end users?
  - *Traceability*:
    is the origin and rationale of a requirement stated?
  - *Adaptability*:
    is it possible to change a requirement without affecting other requirements?