COSC345
Software Engineering

Requirements

Outline

- User requirements
- System requirements
- Functional requirements
- Non-functional requirements
- Problems
- Suggestions

Requirements

- A description of
  - The services to be provided
  - The constraints on the system
- Express
  - The needs of the customer for a system that helps in some way

- Requirements engineering:
  - Finding out requirements
  - Analyzing requirements
  - Documenting requirements
  - Checking the constraints on the system

Definition

- Requirements could be:
  - A high level description of roughly what's needed
  - A detailed formal definition of:
    - What the system will do
    - How it will do it
    - When it will do it
- If a company puts out a bid then:
  - The requirements could be the call
  - A successful contractor will then specify what they will build to match the requirements
- But, the project will change over time and the requirements cannot be "too rigid"

User Requirements

- What the user thinks they want
  - Natural language statements (and diagrams) of what services the user expects and any constraints that are present (like time)
- Should:
  - Include functional ***and*** non-functional requirements
  - Be understandable by the user base
    - Without technical (implementation) knowledge
    - Avoid design characteristics
- Potential Problems:
  - Lack of clarity
  - Lack of functional / non-functional separation
  - Incorrect amalgamation of several requirements into one
  - Requirements should be written to the audience
    - For example: An executive overview should be no longer than 1 page

System Requirements

- Also known as a ***Functional Specification***
- Specify
  - How the user requirements are to be fulfilled
  - The external behavior of the system
  - What is going to be built
    - A precise and detailed description of the system's functions, services and constraints
    - May form part of the contract between builder and buyer
- It may be necessary to build a prototype to formalize the requirements
- Be careful with natural language
  - It is inconsistent and ambiguous

Functional Requirements

- Can be either user or system requirements
- Statements of services provided
  - What the system should do
  - What the system should not do
  - How the system will respond to input
- Problems lead to cost overruns and delays:
  - Imprecise specification
    - Leads to short-cuts
  - Ambiguous specification
    - Leads to easiest implementation
- Functional requirements should be:
  - Complete, precise, and consistent (… but they won’t be)

Non-Functional Requirements

- Constraints on the software
  - Reliability
    - Security
  - Timing constraints (e.g. response times)
  - Resource constraints
    - CPU / Disk usage / Internet bandwidth usage / etc.
  - Development process constraints
    - Including standards
- Apply to the whole system not a system function
- Imagine a 111 response call system
  - What kinds of functional and non-functional requirements do you expect?

Non-Functional Requirements

- Product requirements
  - Performance constraints
  - Memory / CPU usage
- Organizational requirements
  - Standard followed (ISO 9000)
  - Programming Language
  - Design methodologies
- External requirements
  - Interfaces (e.g. Z39.50)
  - APIs, data structures, and byte-ordering conventions
  - Legal / ethical requirements

- Non-functional requirements are difficult to verify

Quantitative Requirements

- If possible, all requirements should be verifiable
  - Functional requirements are already verifiable
    - "The system must [insert operation here]"
  - Non-functional requirements need careful writing
    - The system should be "fast"
      - How can this be verified?
    - The system must have a mean performance of 10 ops/sec
      - This can be verified
- Reword non-functional requirements so they can be verified – but not all can be
  - How can "must be maintainable" be quantified?

Domain Specific Requirements

- Those requirements specific to the domain
  - Language
  - Equations
  - Definitions
  - Protocols
  - Standards
- Acts as a reference to understand functional and non-functional requirements
- E.g. a location-based system needs GPS “stuff”

Structured Languages

- Requirements can be less ambiguous if a controlled vocabulary is used
- And / or a form might be used

Function	Compute Insulin Dose. Safe sugar level
Description	Computed the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Outputs	CompDose – the dose in insulin to be delivered
Description	Main control loop
Action	CompDose is 0 if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing then CompDose is computed by …
…	…

From Sommerville Figure 6.12

Form Based Requirements

- Include a description of:
  - Functionality
  - Inputs and outputs
  - Where the output goes
  - What is required for this component
  - Action to be taken
  - The procedural contract (pre and post conditions)
  - Side effects (if there are any)
  - Etc.

Software Requirements Document

- Official statement of what should be implemented
- Diverse reader base including:
  - Developers – build the system
  - Management – plan and manage the project
  - Test team – verify the software
  - Board members – pay for it
- Includes
  - User requirements
  - System requirements
- International standards
  - US Department of Defense
  - IEEE / ANSI 830-1998

IEEE / ANSI 830-1998

- Introduction
  - Purpose of the requirements document
  - Scope of the product
  - Definitions, acronyms and abbreviations
  - References
  - Overview of the remainder of the document
- General Description
  - Product perspective
  - Product functions
  - User characteristics
  - General constraints
  - Assumptions and dependencies
- Specific requirements
- Appendices
- Index

Problems

- Requirements that are “too rigid”
  - Constrain the project
  - Stifle innovation
  - Requirements without reason will be ignored!
- Requirements documents must be flexible
  - They may be self-contradictory
  - They may be functionally impossible
- Projects change over time
  - Requirements can change so rapidly that the documents are out of date before they are finished
- Over-detailed specification may take a long time to write
  - Longer than the system takes to build!
- Programming languages are formal specification languages

Suggestions

- Settle on a ***standard*** format
  - Omissions are unlikely if they become “habit”
  - Requirements are easier to check
- Include details of ***whose*** requirement it was
  - That individual can verify compliance
  - Consult that individual if changes are needed
- Separate ***mandatory*** from ***desirable*** requirements
  - Use consistent language (shall / should)
- Requirements should be written to the ***audience***
  - E.g. executive overview

References

- I. Sommerville, *Software Engineering*, Chapter 6