Requirements Engineering

Professor Ray Welland
Department of Computing Science
University of Glasgow

E-mail: ray@dcs.gla.ac.uk

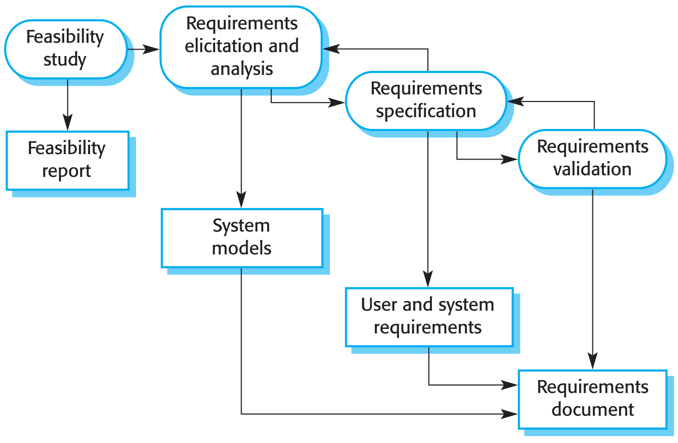
The Importance of Requirements

- Identifying (some) requirements is the starting point for all software development projects regardless of the development method being used.
- In conventional development, the cost of fixing problems after system delivery is high (perhaps 100 times the cost of fixing an implementation error)
- Changes in requirements are inevitable
  - as understanding of the requirements develops
  - as the software development proceeds
  - after the system is delivered (maintenance)

Why Requirements *Engineering*?

- Many projects have failed because of inadequate understanding of requirements
- Need for a **systematic** approach to specifying requirements, stages in the process:
  - **Feasibility Study** – cost/benefit analysis for business and risk analysis prepared for management decision making
  - **Requirements Elicitation and Analysis** (requirements capture or discovery) - finding out what is needed from the customers
  - **Specify Requirements** for the customer (contract) and the system developer
  - **Validate Requirements -** do the specified requirements actually define what the customer wants?

Requirements Engineering - Process



From Sommerville 7 – p143

Desirable Features of a Requirements Specification

- Emphasise **what** is required not how to do it; but there are always constraints (*non-functional requirements*)
- Standardised presentation of documents (e.g. Sommerville 6.5: The Requirements Document), agreed design notations
- Agreed by all customers (need for compromise and trade-offs)
- Realistic: can be realised within cost constraints and using existing technology (or known future technology)
- Consistent: no conflicting requirements
- Complete: identifies all requirements

Requirements Validation

(Sommerville 7.3)

How to ensure that a Requirements Document does define the customers' requirements (check against desirable features)

- **Comprehensibility** - do customers understand the requirements?
- **Verifiability** (testability) - can requirements be realistically tested? {If you cannot test it, it is not a requirement!}
- **Traceability** - source of the requirement? {Important when making changes.}
- **Adaptability** (changeability) - what is the impact of changing a particular requirement?

Requirements Evolution

(Sommerville 7.4 Requirements Management)

- Change is inevitable and must be managed!
- Some requirements are **enduring** (stable) while others are more **volatile** (likely to change). Sommerville (7.4.1) divides volatile requirements into:
  - *Mutable* - caused by environmental changes, changes outside the organisation (e.g. changes in the law)
  - *Emergent* - occur as the customers (and analysts) understanding of the system develops
  - *Consequential* - resulting from the introduction of the new system creating opportunities for further work
  - *Compatibility* - dependent upon other parts of the business, must change if they change

Functional and Non-Functional Requirements

- Requirements can be divided into two main categories; functional and non-functional.
- **Functional** requirements, as the name suggests, define the basic functions of the system and are usually fairly easy to identify and specify.
- **Non-functional** requirements are constraints which restrict the product and the development process, and place external conditions which the product must meet.
- There is a trade-off between functional and non-functional requirements (another compromise)

Non-functional Requirements

- Must be objective and testable
  - (e.g. ‘the system must be user-friendly’ is subjective and there are no criteria for testing the requirement)
- Sommerville (6.1.2) divides non-functional requirements into three major categories:
  - **Product** requirements: usability; efficiency (performance, space); reliability; portability.
  - **Orgnisational** requirements: delivery; implementation; standards.
  - **External** requirements: interoperability; legal (safety, privacy)
- There are standard checklists of non-functional requirements, for example: IEEE Std-830-1993

Non-Functional Requirements - Examples

Compatibility with other systems (internal and external)

Existing Practices - programming languages, methods, tools, platforms

Security of Data and Communications

Performance
throughput (transactions per hour)
response time (maximum acceptable)

Reliability
test load (stress testing)
mean time between failures
availability (up time)

Requirements Elicitation and Analysis

- Sommerville, chapter 7.2
- Elicitation, also called requirement collection, capture or discovery
- Identifying the customers **real** requirements
- Who are the customers?
- The term **stakeholder** is often used rather than customer - anyone who has some direct or indirect influence on the system requirements
- Stakeholders can be managers, end-users of the system, other parts of the organisation, external organisations, ...

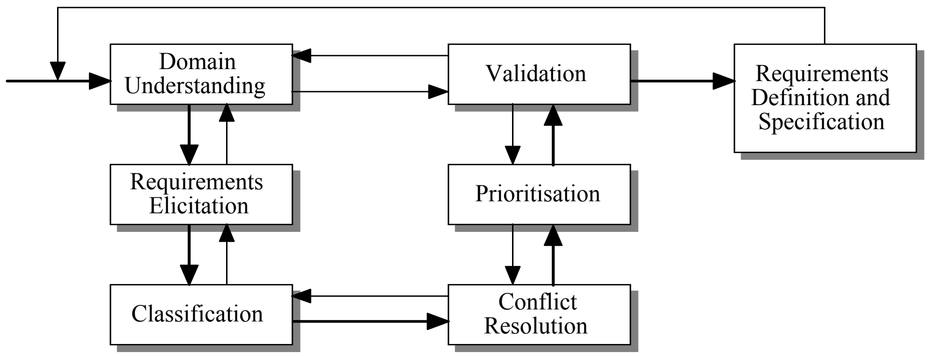
Elicitation and Analysis - **Problems**

- Different types of businesses have their own domain terminology (jargon)
- Organisations have their own terminology and business practices, which are unfamiliar to the system developers
- Stakeholders do not know what they want or make unrealistic demands (e.g. technically infeasible or too expensive)
- Conflict between stakeholder requirements
- Organisational structure and politics (often implicit)
- The organisational environment may change (in the worst case there is a business take over or a major restructuring of the organisation)

Elicitation and Analysis - **Process**

- Domain understanding - the application area and terminology
- Requirements Elicitation - identifying requirements by talking to stakeholders, observing processes, analysing data, etc.
- Classification - organising the information collected, imposing structure (hierarchies, clusters)
- Conflict Resolution - identify and resolve conflicts between different stakeholders requirements
- Prioritisation - identify the most important requirements
- Validation - completeness, consistency and realism

Elicitation and Analysis - **Process (2)**



Requirements Elicitation

Elicit = “to draw forth”

In order to analyse the requirements it is necessary to study the existing system and understand it. This is the field of traditional **systems analysis** and also **usability studies**. A major part of this work requires communication skills rather than technical ability!

Possible techniques:

- **Interviews**
  - planning, recording and analysing; with managers, customers (strategists and those paying for the system!)
    - **Structured**, with pre-planned questions, can be used over telephone
    - **Unstructured**, free format, difficult to analyse
    - **Hybrid**, anywhere between Structured and Unstructured!

Requirements Elicitation (2)

- **Inspection**
  - studying existing records, statistical sampling; useful where there are large volumes of data
- **Observation (Ethnography,** Sommerville 7.2.2**)**
  - observation of activities, processes and workflows
  - participation in activities with end-users
  - video recording of activities, very difficult to analyse
- **Scenarios**
  - talk through different scenarios with users to understand normal and exceptional processes (what if?)
- **Prototypes**
  - Build partial prototypes, quick and dirty, usually focussed on interfaces; try out ideas with users (c.f. active scenario?)
- **Questionnaires**
  - high-volume, low-quality input, difficult to design; suitable for gathering background data (e.g. customer attitudes)

Classification of Requirements

- We need to classify (organise) requirements in order to make it easier to find conflict and redundancy
- There is remarkably little material about classification in general terms!
- Three suggested techniques
  - **Partitioning** - identifying aggregations
  - **Abstraction** - identifying generalisations
  - **Projection** - organisation of knowledge from different viewpoints

Identifying Conflicts and Priorities

- Analysing a draft set of requirements for inconsistencies between requirements
- Using checklists (see next slide for example)
- Systematically checking for conflicts using an interaction matrix
- Negotiating with the stakeholders to resolve conflicts
- Prioritising the agreed requirements, either to meet cost constraints or phased delivery

Checklist for Internal Validation

- Kotonya & Sommerville suggest the following:
  - **Premature design** - early commitment to design or implementation
  - **Complex requirements** - could be split into simpler units
  - **Unnecessary requirements** - “gold-plating”
  - **Use of non-standard platform** - special hardware / software
  - **Conformance with business goals** - consistent with business aims (project mandate)
  - **Ambiguity** - different possible interpretations?
  - **Realism** - given the proposed (available) technology
  - **Testability** - can you write a test for this requirement?

Requirements Engineering - Summary

- Requirements Engineering attempts to provide a systematic approach (framework) to an imprecise problem area
- A major part of requirements engineering concerns conflict resolution between:
  - customer requirements (different viewpoints)
  - functional and non-functional requirements (e.g. level of functionality versus delivery time)
  - non-functional requirements (e.g. performance versus reliability)
- Requirements Engineering process is iterative
- Change is inevitable and must be managed.

Requirements Engineering - Summary (cont.)

- These lectures were intended to identify the *general principles* that apply to requirements engineering, regardless of the methods used
- Background information can be found in:
  Ian Sommerville, Software Engineering (7th Edition), Chapters 6 and 7.
  (Specific references to topics given in the slides.)
  - {Sommerville 6th Edition – chapters 5 and 6 contain similar material.}
- More detailed information in:
  Gerald Kotonya & Ian Sommerville, Requirements Engineering - Processes and Techniques, John Wiley & Sons, 1998

Requirements and Processes

- Requirements Engineering assumes that the bulk of the requirements are identified before development (design, implementation, testing)
- Incremental techniques, such as the Rational Unified Process (using UML) and Extreme Programming (XP), integrate requirements capture within the development cycle. But requirements still have to be identified, conflicts resolved, functions prioritised, …
- Outsourcing of software development puts the emphasis back onto requirement specification before development.