Requirements Engineering: Part I

Software Requirements & Project Management
CITS3220
The Problems of Requirements

What goal(s) are we trying to satisfy?

How do we identify the scope and properties of the solution space?

There are some radically different views in modern SE about the best way to answer these questions. In the next 2 lectures we will explore 2 contrasting approaches.
the **Requirements Specification**

- is the **official statement** of what is required of the system developers
- forms the basis of a system **contract** between client and developer
- is a **reference tool** used by clients, designers, programmers, testers and maintenance engineers.
Attributes of a Good Req. Spec.

- Concise
- Complete
- Unambiguous
- Testable
- Consistent
- Feasible
- Modifiable
- Traceable

- Specifies external system behaviour only
- Specifies constraints on the implementation
- Easy to change
- Ref. Tool for maintainers
- Records forethought about the life cycle
- Characterises acceptable responses to undesired events
Techniques for Writing Good Requirements Specifications

1. Avoid Ambiguity
2. Verifiable Requirements
3. Completeness Measures
4. Standard Document Structures
Avoiding Ambiguity

Problems with NL Specification

- Depends on shared linguistic experience of reader and writer
- Over-flexible
- Difficult to describe the system's functional architecture
- Lack of support for requirements partitioning
- The software system should provide acceptable performance under maximum load conditions.
- The system interface should use a character set available on a standard terminal.
- If the system should fail in operation, there should be minimal loss of data.
- The software development process used should ensure that all of the required reviews have been carried out.
- Structured programming should be used for program development.
- The software must be developed in such a way that it can be used by inexperienced users.
Verifiable Requirements Specs

- Defn. A requirements spec is **verifiable** iff (if and only if) every requirement statement is verifiable iff there is some finite cost-effective way in which a person or machine can check to see if the SW product meets the requirement.

- We can use test cases, analysis or inspection to decide
Some Objective Metrics for Non-functional Requirements

- Performance Speed
  - Number of processed transactions per second
  - User/event response time
  - Screen refresh time

- Size
  - Kilobytes
  - Number of RAM chips
- **Reliability**
  - Mean time to failure
  - Probability of unavailability
  - Rate of failure occurrence
  - Availability Store utilisation
  - Maximum size of system in kilobytes

- **Robustness**
  - Time to re-start after system failure
  - Percentage of events causing failure
  - Probability of data corruption on failure

- **Integrity**
  - Maximum permitted data loss after system failure
Ease of Use
- Training time taken to learn 75% of user facilities
- Average number of errors made by users in a given time period
- Number of help frames

Portability
- Percentage of target-dependent statements
- Number of target systems

Exercise
- Rewrite each of the non-functional requirements on slide 10 so that it can be verified objectively.
Completeness of Requirements

- Have all the requirements needed to specify the requirements of the problem been defined?

- Is the Reqs. Spec. document complete?
Completeness Issues

- Are all inputs covered?
  - Yes, if for every possible input event or value a response has been defined
- Is every object needed by at least one use case?
- Have all boundary cases been considered?
- Fault-tolerance issues?
Document Completeness

- SR Document conforms to a standard document format
- All requirements, figures and tables are numbered
- All requirements are linked to a source or rationale
- A database may be used for managing requirements
Standard Document Structures

- Using a template for the Req.Spec.Doc or for stating individual requirements helps to organise a gathered set of requirements.
- There are many different templates and standards; these will need modification for particular projects or organisations.
Function: Check_card_validity

Description: This operation must ensure that the card input by a user has been issued by a subscribing bank, is in date, contains appropriate account information and includes details of the date and amount of the previous cash withdrawal.

Inputs: Bank-identifier, Account-number, Expiry-date, Last-transaction-date, Last-transaction

Source: Input data is read from the card magnetic stripe

Outputs: Card-status = (OK, invalid)

Destination: Auto-Teller. The card status is passed to another part of the software.

Requires: Bank-list, Account-format, Todays-date

Pre-condition: Card has been input and stripe data read

Post-condition: Bank-identifier is in Bank-list and Account-number matches Account-format and Expiry-date >= Todays-date and Last-transaction-date <= Todays-date and Last-transaction > 0 and Card-status = OK or ( if any of these tests fail ) Card-status = invalid

Side-effects: None

Specification using structured forms
Requirements Validation
Reqs Analysis vs Validation

- **Analysis** works with raw requirements as elicited from the system stakeholders
  - “Have we got the right requirements” is the key question to be answered at this stage

- **Validation** works with a final draft of the requirements document i.e. with negotiated and agreed requirements
  - “Have we got the requirements right” is the key question to be answered at this stage
Validation inputs

- Requirements document
  - Complete, formatted and organised according to organisational standards

- Organisational knowledge
  - Knowledge, often implicit, of the organisation; used to judge the realism of the requirements

- Organisational standards
  - Local standards e.g. for the organisation of the requirements document
Validation outputs

- Problem list
  - List of discovered problems in the requirements document

- Agreed actions
  - List of agreed actions in response to requirements problems. Some problems may have several corrective actions; some problems may have no associated actions
Requirements reviews

- A group of people read and analyse the requirements, look for problems, meet and discuss the problems and agree on actions to address these problems

- We will try some of these in workshops
Review checklists

- Understandability
  - Can readers of the document understand what the requirements mean?

- Redundancy
  - Is information unnecessarily repeated in the requirements document?
Review checklists

- Understandability
  - Can readers of the document understand what the requirements mean?

- Redundancy
  - Is information unnecessarily repeated in the requirements document?
Review checklists

- Consistency
  - Do the descriptions of different requirements include contradictions? Are there contradictions between individual requirements and overall system requirements?

- Organisation
  - Is the document structured in a sensible way? Are the descriptions of requirements organised so that related requirements are grouped?
Review checklists

- Conformance to standards
  - Does the requirements document and individual requirements conform to defined standards? Are departures from the standards, justified?

- Traceability
  - Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included?
Checklist questions

- Is each requirement uniquely identified?
- Are specialised terms defined in the glossary?
- Does a requirement stand on its own or do you have to examine other requirements to understand what it means?
- Do individual requirements use the terms consistently?
- Is the same service requested in different requirements? Are there any contradictions in these requests?
- If a requirement makes reference to some other facilities, are these described elsewhere in the document?
- Are related requirements grouped together? If not, do they refer to each other?
Prototyping

- Prototypes for requirements validation demonstrate the requirements and help stakeholders discover problems.
- Validation prototypes should be complete, reasonably efficient and robust. It should be possible to use them in the same way as the required system.
- User documentation and training should be provided.
User manual development

- Writing a user manual from the requirements forces a detailed requirements analysis and thus can reveal problems with the document.

Information in the user manual:
- Description of the functionality and how it is implemented.
- Which parts of the system have not been implemented.
- How to get out of trouble.
- How to install and get started with the system.
Hard-to-test requirements

- System requirements
  - Requirements which apply to the system as a whole. In general, these are the most difficult requirements to validate irrespective of the method used as they may be influenced by any of the functional requirements. Tests, which are not executed, cannot test for non-functional system-wide characteristics such as usability.
Hard-to-test requirements 2

- **Exclusive requirements**
  - These are requirements which exclude specific behaviour. For example, a requirement may state that system failures must never corrupt the system database. It is not possible to test such a requirement exhaustively.
Hard-to-test requirements 3

- Some non-functional requirements
  - Some non-functional requirements, such as reliability requirements, can only be tested with a large test set. Designing this test set does not help with requirements validation.
Key Points of the Lecture

- **Requirements Specification Document**
  - Basis for client-developer contract

- **Desirable attributes**
  - Concise, complete, consistent, unambiguous, testable, modifiable, traceable, feasible

- **Techniques**
  1. Avoid Ambiguity
  2. Make Requirements Verifiable
  3. Test for Completeness
  4. Use Standard Document Structures and Templates
Key points of the Lecture

- Requirements validation should focus on checking the final draft of the requirements document for conflicts, omissions and deviations from standards.
- Inputs to the validation process are the requirements document, organisational standards and implicit organisational knowledge. The outputs are a list of requirements problems and agreed actions to address these problems.
- Reviews involve a group of people making a detailed analysis of the requirements.
Key points of the Lecture

- Checklists of what to look for may be used to drive a requirements review process.
- Prototyping is effective for requirements validation if a prototype has been developed during the requirements elicitation stage.
- Systems models may be validated by paraphrasing them. This means that they are systematically translated into a natural language description.
- Designing tests for requirements can reveal problems with the requirements. If the requirement is unclear, it may be impossible to define a test for it.
The Problems of Requirements

What goal(s) are we trying to satisfy?

How do we identify the scope and properties of the solution space?

There are some radically different views in modern SE about the best way to answer these questions. In 2 lectures we will explore 2 contrasting approaches.
In Lecture 1 we looked at

The **Requirements Specification**…

- is the **official statement** of what is required of the system developers
- forms the basis of a system **contract** between client and developer
- is a **reference tool** used by clients, designers, programmers, testers and maintenance engineers.
In Lecture 2...

We will examine a contrasting approach.

*Highly Recommended Reading*

W Royce

**Successful Software Management Style: Steering and Balance**

IEEE Software, Sep/Oct 2005

Problem Space

Planning Space

Solution Space

Measurement Space
Requirements

“Required” means non-negotiable
BUT real requirements are

- Bartered,
- Changed, and
- Negotiated, all the time.
Precision vs Accuracy

- We need to accurately forecast estimates, risks, change

BUT

- Unjustified precision is dishonest
How to Steer?

- Scope Management
- Process Rigour
- Progress Honesty
- Quality Control
  - Test demonstrable releases
Scope Management

user specifications

[arrow]

candidate solutions
Process Rigour

- Current SE practice focuses on either
  - Light management (agile process)
  OR
  - Heavy management (process maturity)
- Royce proposes **light EARLY** and **heavy LATER** in project time
- Light requirements management contradicts the methods we discussed yesterday!
Progress Honesty

- Avoid degrading customer-developer relations to mutual distrust
- Customers should be focussed on delivering a usable system (and not on standards and contract terms)
- Developers should be focussed on delivering value in a profitable manner
Quality Control

- Test demonstrable releases
- More on this later in the unit