Requirements Engineering

1 Gathering Requirements

- A software requirement is a functional or non-functional need to be implemented in the system
- Functional means providing particular service to the user
  - For example, in context to an online purchase application the functional requirement will be when customer selects ”Checkout” they must be able to look at the latest items in their basket before paying
- A software requirement can also be non-functional, it can be a performance requirement
  - For example, a non-functional requirement is where a click on a link results in a <0.5s delay to give a response

2 Functional Requirements

- Descriptions of data to be entered into the system
- Descriptions of operations performed on each system
- Descriptions of work flows performed by the system
- Descriptions of system reports or other outputs
- Who can enter data into the system
- How the system meets applicable regulatory requirements
- The user can get some meaningful work done

3 Non-Functional Requirements

- Characteristics of the system which cannot be expressed as functions:
  - Maintainability, portability, usability, security, safety, reliability, performance ..
  - These characteristics can be measurable
  - E.g a system sub-component that needs to have a response time or <1s 95% of the time
  - Constraints are NFR
  - PM issues are NFR (costs, schedule, logistics)

4 RE: Elicitation/discovery

Develop a plan

- What information should be discovered?
- What sources should be used?
- What mechanisms or techniques should be employed

The information to discover:

- A description of the problem domain
- The basic type of application
- The identity of the stakeholders
- The main motivation behind the development
- List of problems requiring solutions (the requirements)
- Any stakeholder imposed constraints upon behaviour or structure of solution system

5 Techniques for discovery

Interviewing:

- Different types of interviews
- Good for getting an overall understanding of what, how they interact with the system and any difficulties
- Not good for understanding specific domain requirements

Scenarios

- Useful for adding detail to an outline requirements description
- Starts with an outline of the interaction and adds detail to create a complete description - must include a description of:
  - What system and users expect when scenario starts
  - Normal flow of events
  - What can go wrong and how it is handled
  - Information on other activities which might be going on at the same time
  - System state when scenario finishes

6 RE: Analysis

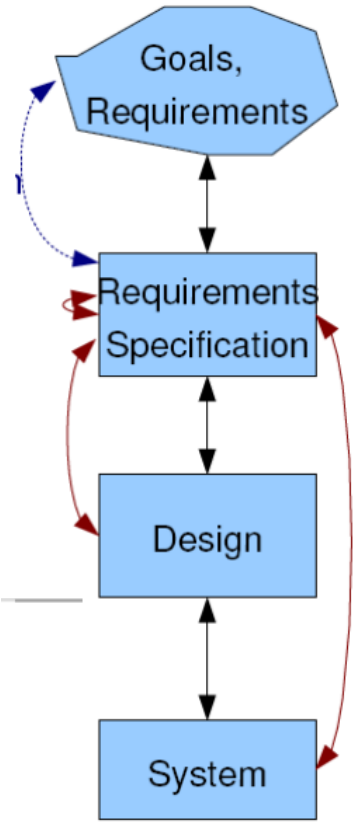
Having determined the basic list of requirements we now analyse them

- Check completeness
- Remove inconsistencies
- Conflict resolution
- Revisit assumptions
- Prioritise

7 Prioritise

- Must have
  - Fundamental requirements, without which the system will not work
  - Define the MVP
- Should have
  - Considered important but the lack of these can be worked around
- Could have
  - Extended functionality but can be left out without undermining the project
- Won't have
  - The waiting list, requirements that can wait until a later development date

8 Validation of requirements



Requirements Validation

- Check that the **right product is being built**
- Ensures that the software being developed (or changed) will satisfy its stakeholders
- Checks the software requirements specification against stakeholders goals and requirements

Requirements verification:

- Check that the **right product is being built right**
- Ensures that each step followed in the process of building the software yields the right products
- Checks consistency of the software requirements specification artefacts and other software development products (design, implementation, ...) against the specification

Requirements error costs are so high so validation is very important

- Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error

Help ensure delivery of what the client wants
Need to be performed at every stage during the (requirements) process

- Elicitation
  - Checking back with the elicitation sources
  - ”So, are you saying that ...?”
- Analysis
  - Checking that the domain description and requirements are correct
- Specification
  - Checking that the defined system requirement will meet the user requirements under the assumptions of the domain/environment
  - Checking conformity to well-formedness rules, standards ..

9 Requirements checking

- Validity checks - Does the system provide the functions which best support the customer’s needs?
- Consistency checks - Are there any requirements conflicts?
- Completeness checks - Are all functions required by the customer included?
- Realism checks - Can the requirements be implemented given available budget and technology?
- Verifiability checks - Can the requirements be checked (tested)?
- Traceable - Where/who did the requirement come from and rationale?

10 How to make the requirements testable

Each requirement carries a **fit** criteria
The fit criteria are a benchmark, or goal, that the testing activity used to determine whether the eventual solution satisfies the requirement

For FR - the function is either completed or not - no scales of measure
For NFR is a quality the product must have, fit criterion is a measure of that

If such criteria cannot be easily expressed or quantified then the requirement is likely to be ambiguous, incomplete or incorrect

Relatively easy of determine fit criteria for quality requirements that are naturally quantitative e.g. performance, size, accuracy

11 The system with assumptions

- Often the requirements for a system-to-be include assumptions about the environment of the system
- The system specification S, then, has the form

$$S = A \Rightarrow G$$

  - where A are the assumptions about the environment and G are the guarantees that the system will meet the requirements as long as A holds
- If these assumptions (A) are implied by the known properties of the domain (D), that is $D \Rightarrow A$, and we can check that the domain properties (D) and the system guarantees (G) imply the requirements (R), that is D and G⇒ R, then the validation condition, D and S ⇒ R is satisfied

12 Formal Verification and Validation

Evaluating the satisfaction of ”D and S⇒ R” is difficult with natural language

- Descriptions are verbose, informal, ambiguous, incomplete ...
- This represents a risk for the development and organization

Verification of this ”validation question” is more effective with formal methods (see below)

- Based on mathematically formal syntax and semantics
- Proving can be tool-supported

Depending on the modelling formalism used, different verification methods and tools may be applied. We call this ”Model-Based V& V”

13 V&V vs Analysis

Both have several activities in common

- Reading requirements, problem analysis, meetings and discussions ...

Analysis works with raw, incomplete requirements as elicited from the system stakeholders

- Develop a software requirements specification document
- Emphasis on ”we have the right requirements”

Requirements V&V works with a software requirements specification and with negotiated and agreed (and presumably complete) domain requirements

- Check that these specifications are accurate
- Emphasis on ”we have the right requirements well done”

14 Requirements V&V Techniques

Various types:

- Simple checks - traceability, well-written requirements
- Prototyping
- Functional test design
- User manual development
- Reviews and inspections
  - Walkthroughs
  - Formal inspections
  - Checklists
- Model-Based V& V
  - First order logic
  - Behavioural models

14.1 Simple Checks

Various checks can be done using traceability techniques

- Given the requirements document, verify that all elicitation notes are covered
- Tracing between different levels of requirements
  - Checking goals against tasks, features, requirements ...

Involves developing a traceability matrix

- Ensures that requirements have been taken into consideration (if not there should be a reason)
- Ensures that everything in the specification is justified

14.2 Prototyping

- The prototyping process can start with those requirements which are poorly understood - helps stakeholder understand what the system will do
- Prototyping can be used to animate requirements (can build on earlier prototype)
- Prototypes for validation must be more complete than for elicitation - must contain sufficient facilities so practical used can be made of it
- Implementation of a number of features helps user identify important features and those not so important
- Useful for questions about user interfaces

14.3 Functional Test design

Functional tests at the system level must be developed sooner or later...

- Can (and should) be derived from the requirements specification
- Each (functional) requirement should have an associated test
- Non-functional (e.g. reliability) or exclusive (e.g. define what should not happen) requirements are harder to validate with testing
- Each requirements test case must be traced to its requirements

- Inventing requirements tests is an effective validation techniques

Designing these tests may reveal errors in the specification (even before designing and building the system)

- Missing or ambiguous information in the requirements description may make it difficult to formulate tests

Some software development processes (e.g.agile methods) begin with tests before programming test-driven development (TDD)

14.4 User Manual Development

Same reasoning as for functional test design:

- Has to be done at some point
- Reveals problems earlier

Forces a detailed look at requirements
Particularly useful if the application is rich in user interfaces/for usability requirements

Typical information in a user manual

- Description of the functionality
- How to get out of trouble
- How to install and get started with the system

14.5 Reviews and inspections

A group of people (mix of stakeholders) read and analyse requirements, look for potential problems, meet to discuss the problems, and agree on a list of action items needed to address these problems

A widely used requirements validation technique

- Reading the document - a person other than the author of the document
- Reading and approval (sign off) - encourages the reader to be more careful (and responsible)
- Walkthroughs (informal)
- Inspections (formal) - very structured and detailed review, defined roles for participants, preparation is needed, exit conditions are defined

Can be expensive, requires careful planning and preparation

- Need appropriate checklists (must be developed if necessary and maintained)

Advantages

- Effective (even after considering cost)
- Allow finding sources of errors (not only symptoms)
- Requirements authors are more attentive when they know their work will be closely reviewed
- Familiarise large groups with the requirements (buy-in)
- Diffusion of knowledge

Risks

- Reviews can be dull and draining (need to be limited in time)
- Time consuming and expensive (but usually cheaper than the alternative)
- Personality problems
- Office politics

14.6 Model-based (formal) V& V

Available V&V techniques will vary from one modelling paradigms to another and will also depend on the available tools

The following functions may be provided through tools:

- Completeness checking: only according to certain syntax rules, templates
- Consistency checking: given model M, show that M does not imply a contradiction and does not have any other undesirable general property (e.g. deadlock possibility)
- Refinement checking: given two models M and M', show that the properties of M imply the properties of M'. This can be used for the validation of the system specification S, that is, showing that D and S $\Rightarrow$ R where D are the domain properties and R are the domain requirements (M = D and S; M' = R)
- Model checking: given a model M and some properties P, show that any system implementation satisfying M will have the properties P
- Generation of system designs or prototype implementations (from workflow or state machine models)
- Generation of test cases
- Performance evaluation

15 The requirements document

- It is NOT a design document. As far as possible, it should set of what the system should do rather than how it should do it
- Give full visibility and secure record of the required behaviour
- Ensure correct information is accurately communicated to the developers, facilitates future maintenance
- Provide a baseline against which functionality can be tested
- Provide information for production of user manuals etc.
- To facilitate future development of new systems with similar functionality
- It's an official statement of what the system developers should implement

16 IEEE propose Requirements Documents to be..

- **Unambiguous** - requirements are often written in a natural language where statements can have more than one meaning. Formal requirements languages help reduce ambiguity because formal language processors automatically detect many lexical, syntactic and semantic errors
- **Complete** - The requirements document is complete if it includes all the significant requirements, whether relating to functionality, performance, design constraints, attributes or external interfaces and conforms to the company standard
- **Verifiable** - an example of a non-verifiable requirement 'the product should have a good human interface'. An example of a verifiable requirement 'the system will respond to a users request within 20 seconds of the user pressing the enter key, 80% of the time.
- Consistent - these types of conflict can occur
  - different terms used for the same object: for example, a 'P45' and a 'tax form' might be used to describe the same form
  - characteristics of objects conflict: different parts of the document state different actions for dealing with faults
  - Logical or temporal faults: for example, 'A follows B' and A and B occur simultaneously to another

- **Modifiable** - the requirements document should have a coherent and easy-to-use organisation, with a table of contents, an index and explicit cross referencing. Statements should be non-redundant where possible
- **Correct** - every requirement listed within the document will be met by the software
- **Traceable** - the origin of each requirement should be clear, thus facilitating 'backward traceability' to previous decisions made and 'forward traceability' to all documents 'spawned' from the requirements
- **Usable** - The requirements document should be designed such that it can be referred to and if necessary modified throughout the life of the product. It should be usable even in the operation and maintenance phase
- **Ranked** - The requirements should be ranked by importance (for instance critical, important, desirable). Assessment to be made by developer/ users

17 Guidelines for writing requirements

Invent a standard format and use it for all requirements
Use language in a consistent way. Use shall for mandatory requirements, should be for desirable requirements
Use text highlighting to identify key parts of the requirements
Avoid the use of computer jargon
To ensure requirements are traceable they must have

- A unique number
- An identifier of the type of requirement or constraint
- Reference to all of the business events and use cases that contain the requirement
- References to dependent requirements that might use the same matter, or have some change effect on this one
- Consistent use of terminology