# Requirements Engineering

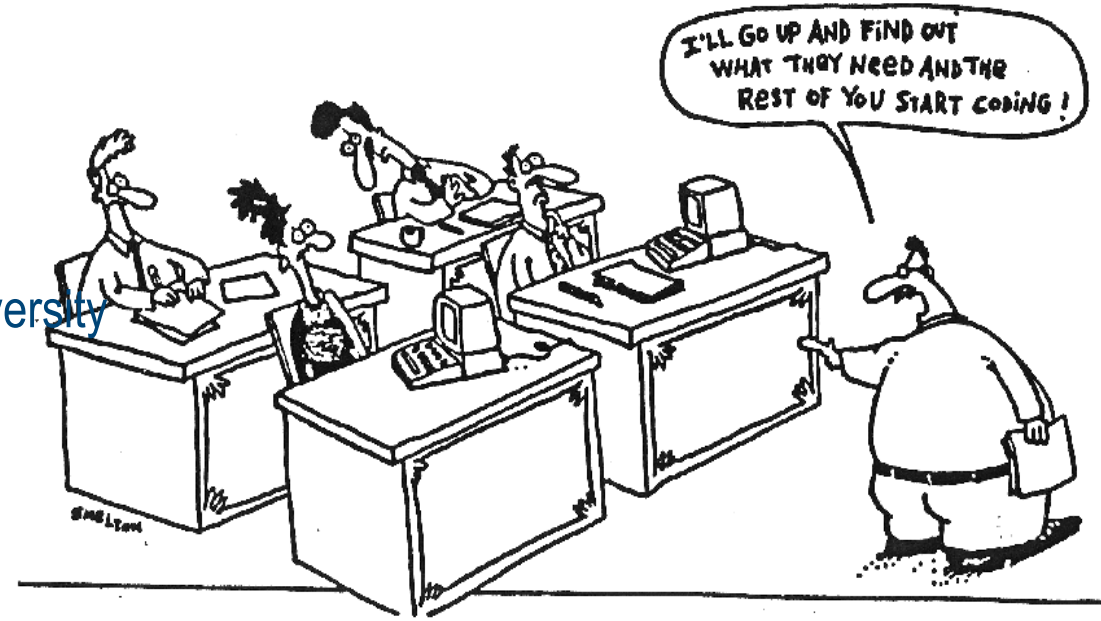
Sommerville, Chapters 6, 7


Instructor: Peter Baumann

email: pbaumann@constructor.university

tel: -3178

office: room 88, Research 1

# Overview

- User vs system requirements
- Functional vs non-functional requirements
- Wrap-up and practical hints

*"If I asked customers what they want they would have said a faster horse."*
-- Henry Ford

# Schematic Requ Eng Procedure

[Pressman]

- Inception — ask questions
  - basic problem understanding; *domain language & implicit knowledge!*
  - Identify stakeholders; *recognize multiple points of view!*
  - Establish trustful communication & collaboration between customer & client; *they will test your competence initially!*
- Elaboration
  - create analysis model that identifies data, function and behavioral
- Negotiation
  - Determine each stakeholder's “win conditions”, negotiate “win-win”
- Requirements management
  - Specification write-up
  - Validation, eg review mechanism: Errors; clarification need; missing information; inconsistencies (major problem!); conflicting / unrealistic requirements
  - Document versioning!

# Requirements Engineering

- Requirements engineering = The process of eliciting
  - the services that the customer requires from a system and
  - the constraints under which it operates and is developed

*functions*

*rules*

- Requirements = descriptions of the system services and constraints that are generated during the requirements engineering process
  - may range from high-level abstract statements to detailed formalized functional specifications

# Types of Requirements

- User requirements
  - Statements in natural language + diagrams of the system's services and its operational constraints
  - *Written for customers, must use their language and mental models*
- System requirements
  - A structured document setting out detailed descriptions of the system's functions, services and operational constraints
  - Defines what should be implemented so may be part of a contract between client and contractor
  - *Must be concise in their technical effects*

# User Requirements

- Must be understandable by users who don't have detailed technical knowledge
- defined using natural language, tables and diagrams
- However, with NL there is a danger of:
  - Lack of clarity – *Precision vs difficulty to read*
  - Requirements confusion – *Functional and non-functional requirements tend to be mixed-up*
  - Requirements amalgamation – *different requirements expressed together*
- *UML is your friend!*

# Specification Techniques

- Natural language
  - Ambiguous; over-flexible; lack of modularisation
- Structured language specifications
  - standard templates; limited terminology → *expressiveness* + *aka uniformity*
- Form-based (tabular) specifications
  - Predefined items → *guidance for completeness*
- Description languages / formal specification
  - Concise, but unsuitable for customers; real-life situations generally untractable
- Graphical models

# Structured Presentation Example

**2.6.1 Grid facilities**

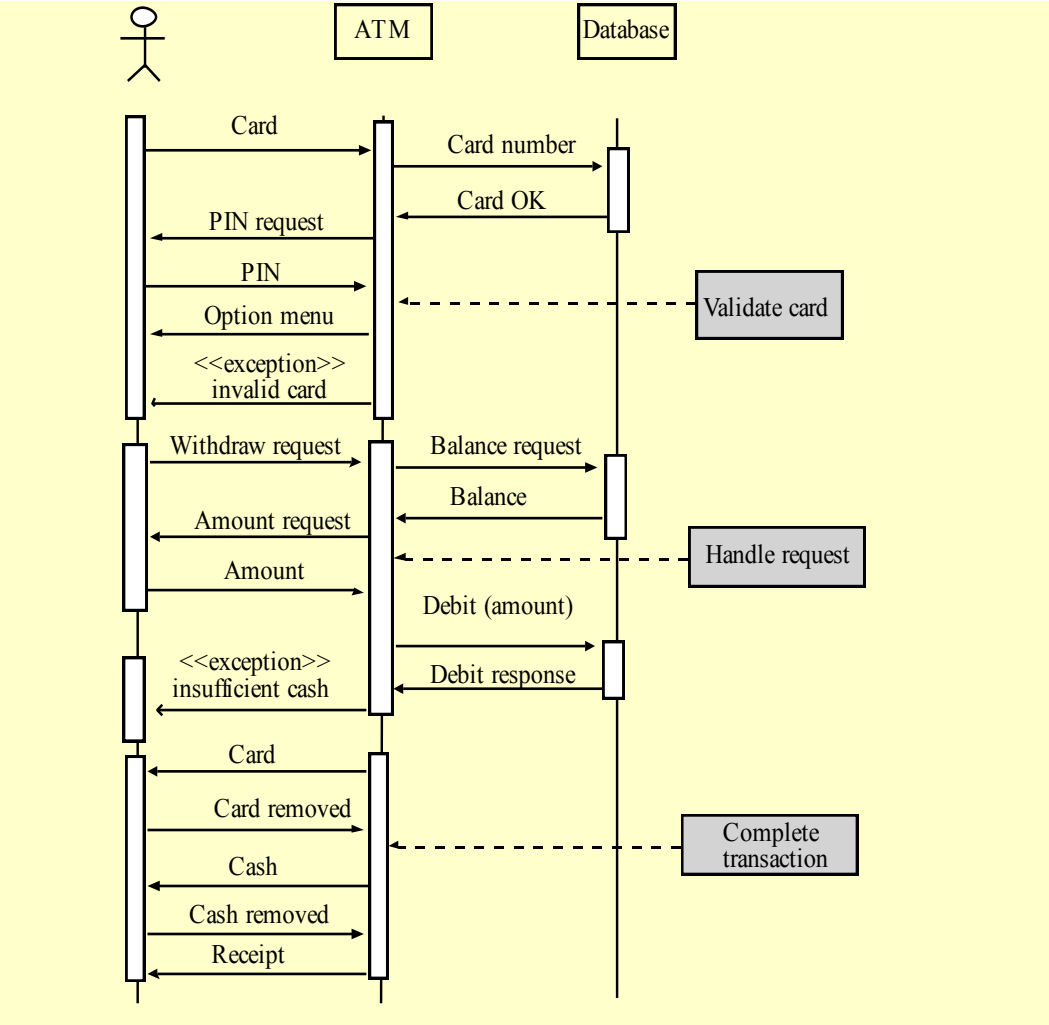
**The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window.** This grid shall be a passive grid where the alignment of entities is the user's responsibility.

*Rationale*: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

*Specification*: ECLIPSE/WS/Tools/DE/FS Section 5.6

*Source*: Ray Wilson, Glasgow Office

# Example: Sequence Diagram of ATM Withdrawal



# User vs. System Requirements: Example

User requirement definition

1. The software must provide a means of representing and accessing external files created by other tools.

derive, and document it!

System requirements specification

1.1 The user should be provided with facilities to define the type of external files.

1.2 Each external file type may have an associated tool which may be applied to the file.

1.3 Each external file type may be represented as a specific icon on the user's display.

1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.

1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

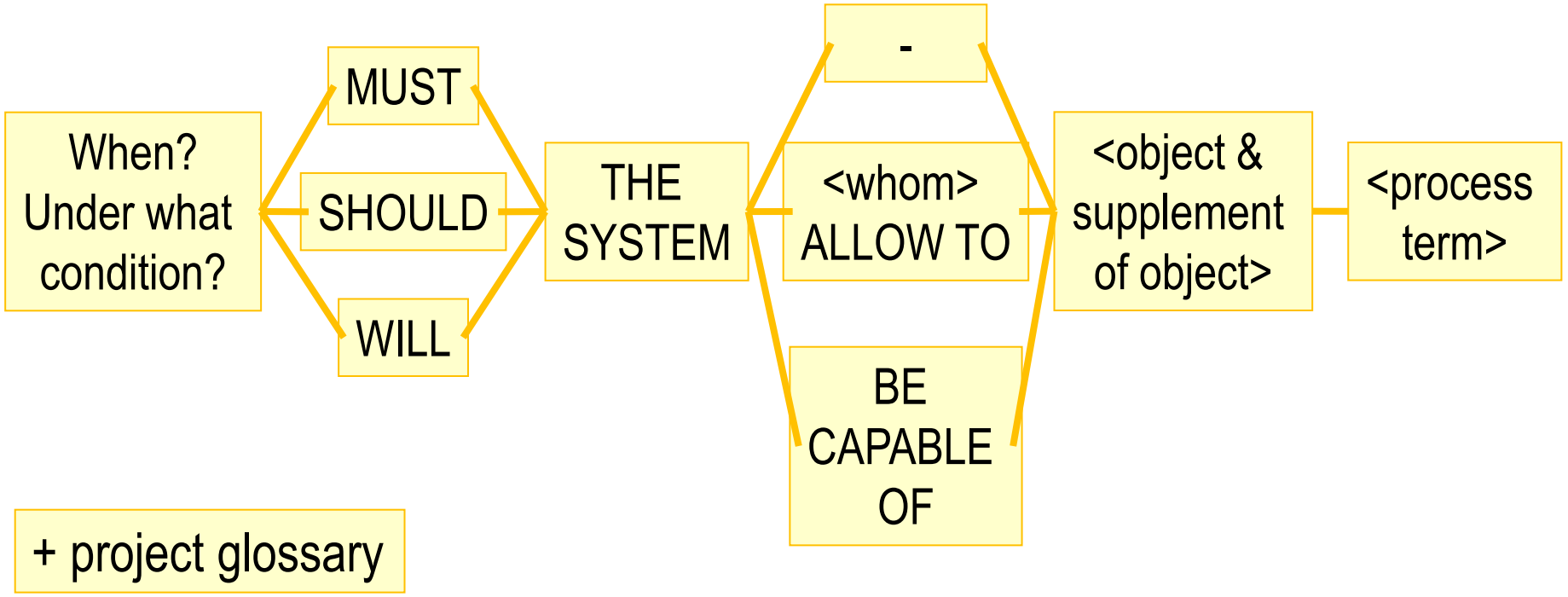
# Bad Language Style

- ECSS Recommendations for the wording of requirements (section 8.3)

1. General format: should be stated in "what-is-necessary" terms, as opposed to telling "how to" perform a task and should be expressed in a positive way, as a complete sentence (with a verb and a noun).
2. Required verbal form:
   "shall" = requirement; "should" = recommendation;
   "may" = permission and "can" = possibility or capability.
3. Format restrictions (List of terms that shall not be used in a TS requirement)
   **"and/or"**, "etc.", "goal", "shall be included but not limited to", "relevant", "necessary", "appropriate", "as far as possible", "optimize", "minimize", "maximize", "typical", "rapid", "user-friendly", "easy", "sufficient", "enough", "suitable", "satisfactory", "adequate", "quick", "first rate", "best possible", "great", "small", "large" and **"state of the art"**.

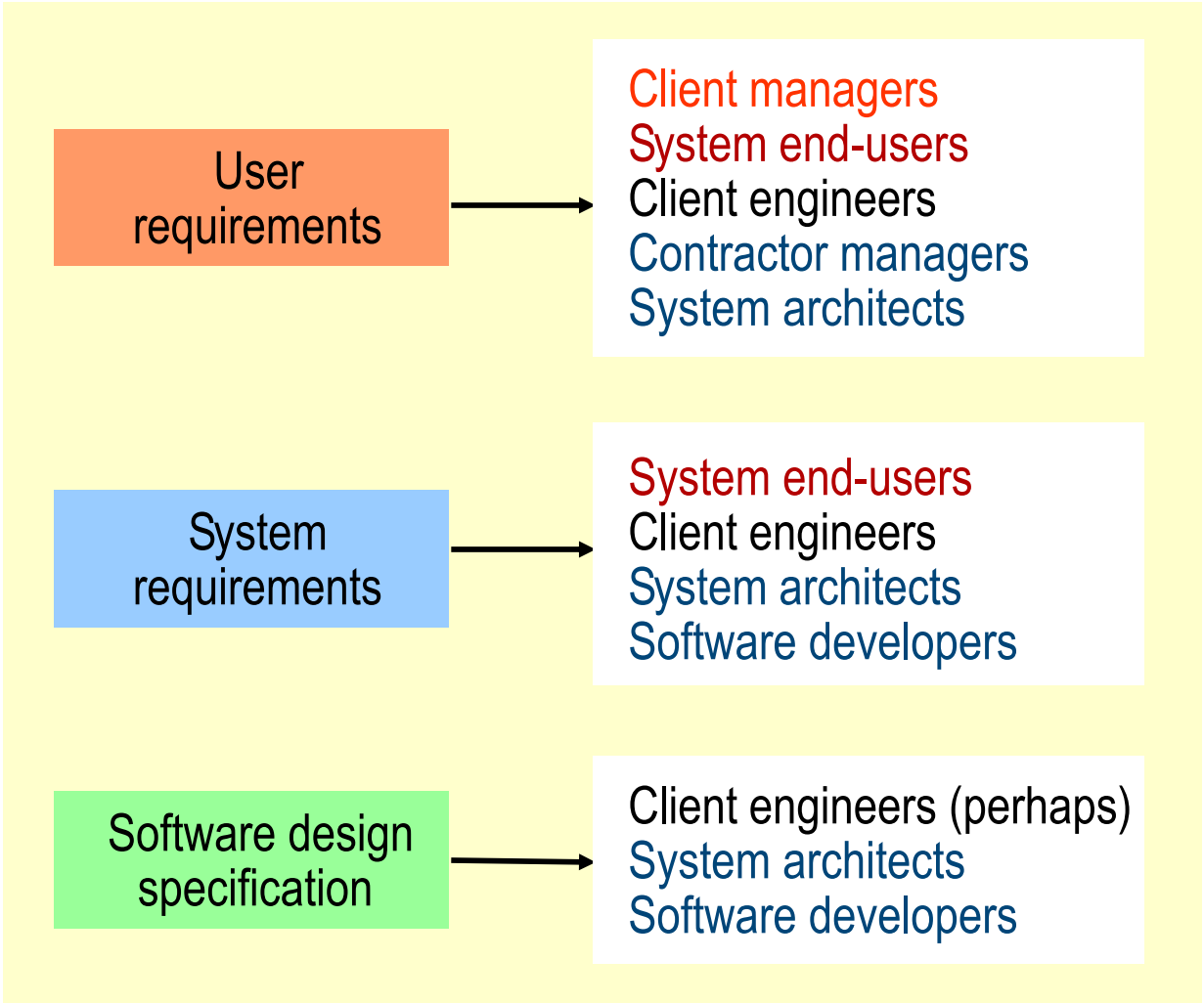
European Cooperation for Space Standardization - ECSS-E-ST-10-06C (6 March 2009) - Technical requirements

# Sample Requirements Sentence Template

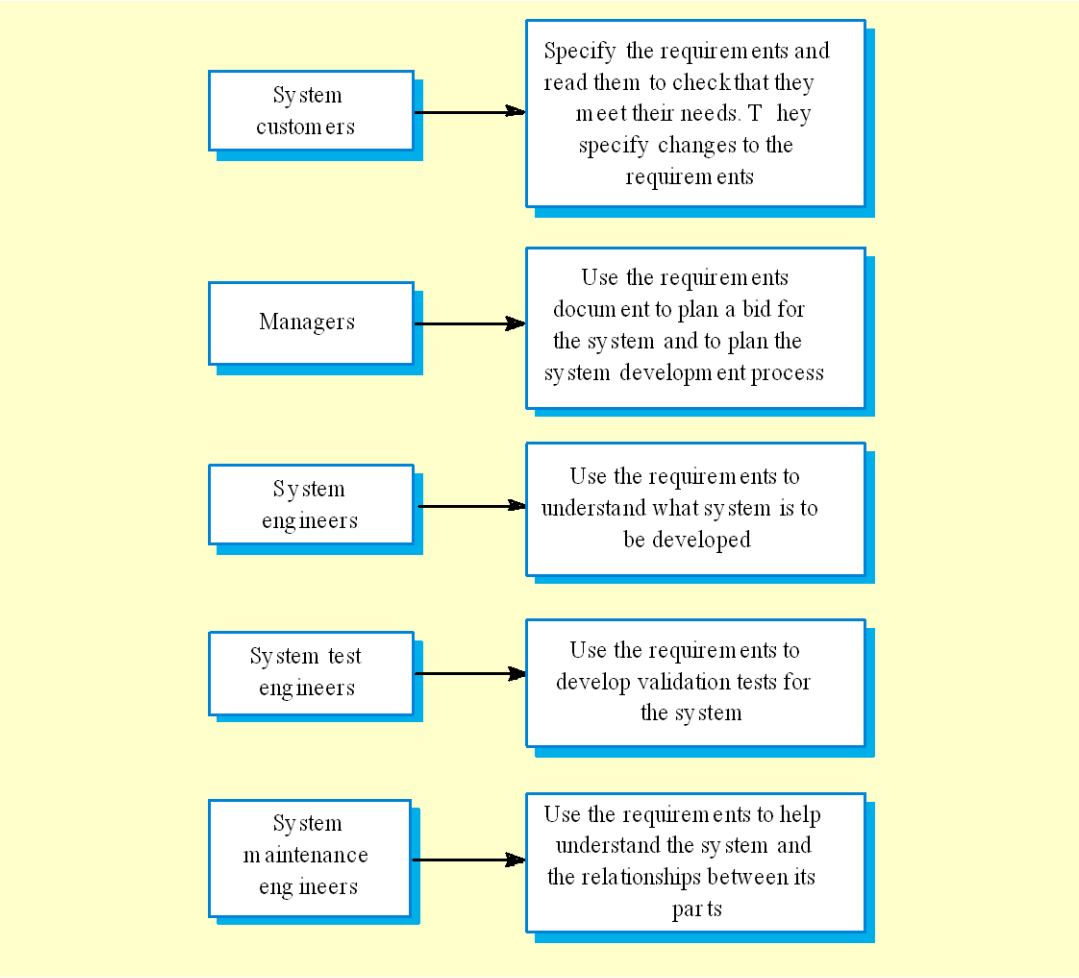



after [Pohl, Rupp 2013]

# Inset: Who Reads Requirements?



# Inset: Who Uses a Requirements Document



# Functional vs Non-Functional Requirements

- Functional requirements
  - Statements of services the system should provide
  - how the system should react to particular inputs
  - how the system should behave in particular situations.
- Non-functional requirements
  - constraints offered by the system such as timing constraints
  - constraints on the development process, standards, etc.
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.
- *Note: this classification is orthogonal to the user/system requirements*

# Examples of Functional Requirements

- LIBSYS system:
  - A library system that provides a single interface to a number of databases of articles in different libraries.
  - Users can search for, download and print these articles for personal study.
- Requirements
  - "*Users shall be able to search either all of the initial set of databases or select a subset from it.*
  - *The system shall provide appropriate viewers for the user to read documents in the document store.*
  - *Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area."*

# Requirements Imprecision

- Problems when requirements not precisely stated
- Ambiguous requirements may be interpreted in different ways by developers and users
  - …and they will !
- ‘appropriate viewers’?

# Non-Functional Requirements

- These define system properties and constraints
  - Properties: *reliability, response time and storage requirements, …*
  - Constraints: *I/O device capability, system representations, …*
  - + domain constraints: *security, legal impacts, domain expert workflow, …*
- Process requirements may also be specified mandating a particular CASE system, programming language, or development method
- Non-functional requirements may be more critical than functional requirements
  - If not met: system is useless

# Non-Functional Requirements Examples

- Product requirement
  - *8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.*
- Organisational requirement
  - *9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95.*
- External requirement
  - *7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.*

# Goals Are Your Friend

- Non-functional requirements may be very difficult to state precisely, imprecise requirements may be difficult to verify
- Goal
  - A general user intention, such as ease of use
- Verifiable non-functional requirement…
  - … = statement using some measure that can be objectively tested
  - Use-case scenarios, augmented with measurable effect!
- Goals helpful to developers: illustrate intentions of users

# Example Goals

- System goal
  - *"The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised."*
- Verifiable non-functional requirement
  - *"Experienced controllers shall be able to use all the system functions after a total of two hours training.*
  - *After this training, the average number of errors made by experienced users shall not exceed two per day, logged over one week."*

# Wrap-Up: The Requirements Document

www.quatic.org

- Requirements document =
  user requirements
  + system requirements
  - Functional + non-functional [+ domain] requirements
  - Link customer ↔ developer
- NOT a design document
  - As far as possible: WHAT system should do rather than HOW
- For user requirements:
  Avoid computer jargon (your language), use customer's language instead
- ~~Invent a standard format~~ Adopt your company's convention
  and use it for all requirements

# Practitioner's Hints

- Talk, talk, talk with all customer stakeholders
  - Everything you miss / don't understand enough will be very expensive later
  - Understand your customer's business better than s/he does...they expect *you* to find right solution
  - desire vs demand
  - but "*remember the Golden Rule: Those who have the gold make the rules.*"
- Fix everything in writing, have it acknowledged (signed!) by customer
  - It may save your / your company's life
  - Your programmers at home must understand what to do
- Be concise, but use your customer's language in the User Req Spec
- Use tools as early as possible, but lay them aside when necessary
  - Document versioning, UML diagramming