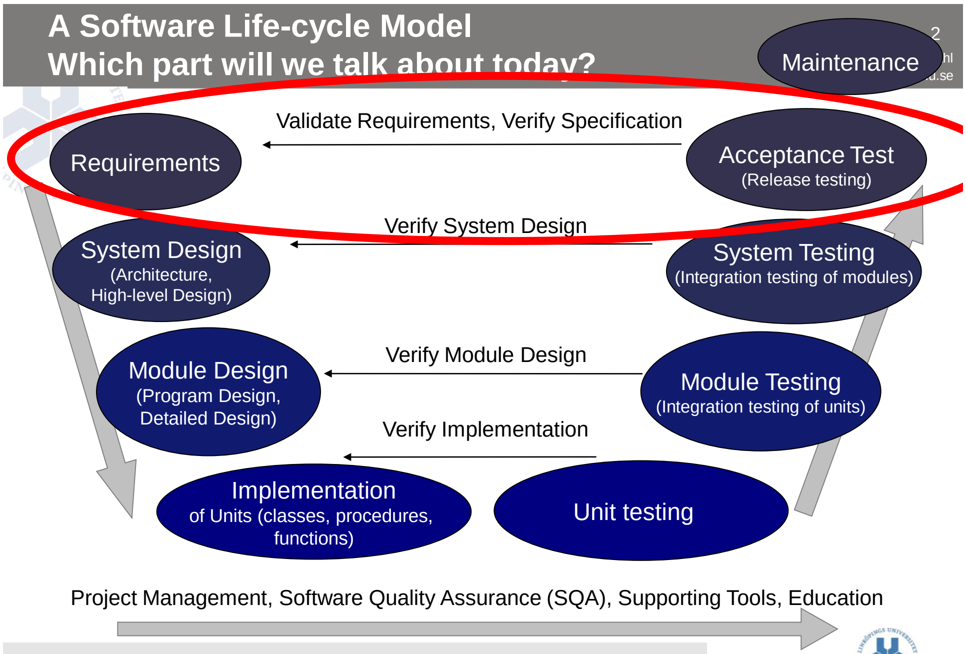
Requirements Engineering

Lecture 7-8

Software Engineering
CUGS
Spring 2011

Kristian Sandahl
Department of Computer and Information Science
Linköping University, Sweden
krisa@ida.liu.se

A Software Life-cycle Model
Which part will we talk about today?



Maintenance

Validate Requirements, Verify Specification

Requirements

Acceptance Test
(Release testing)

Verify System Design

System Design
(Architecture,
High-level Design)

System Testing
(Integration testing of modules)

Verify Module Design

Module Design
(Program Design,
Detailed Design)

Module Testing
(Integration testing of units)

Verify Implementation

Implementation
of Units (classes, procedures,
functions)

Unit testing

Project Management, Software Quality Assurance (SQA), Supporting Tools, Education

Part I
Requirements Elicitation

Part II
Requirements Analysis

Part III
Requirements Specification

What is a software requirement?

- "Software requirements express the needs and constraints placed on a software product that contribute to the solution of some real-world problems."

(Kotonya and Sommerville 2000)

- Example:

When the user enters the degrees in Farenheit, the system shall calculate and write the degrees in Celsius.

Functional and non-functional requirements

- Functional requirements describe the functions that the software is to execute.
- Can be tested by giving input and checking the output.

- Non-functional requirements:
  - Design constraints
  - Quality requirements, possible to measure

Quality factors

- Correctness
- Reliability
- Efficiency
- Usability
- Integrity
- Maintainability
- Flexibility
- Testability
- Security

- Portability
- Reusability
- Interoperability
- Survivability
- Safety
- Manageability
- Supportability
- Replaceability
- Functionality

Price?

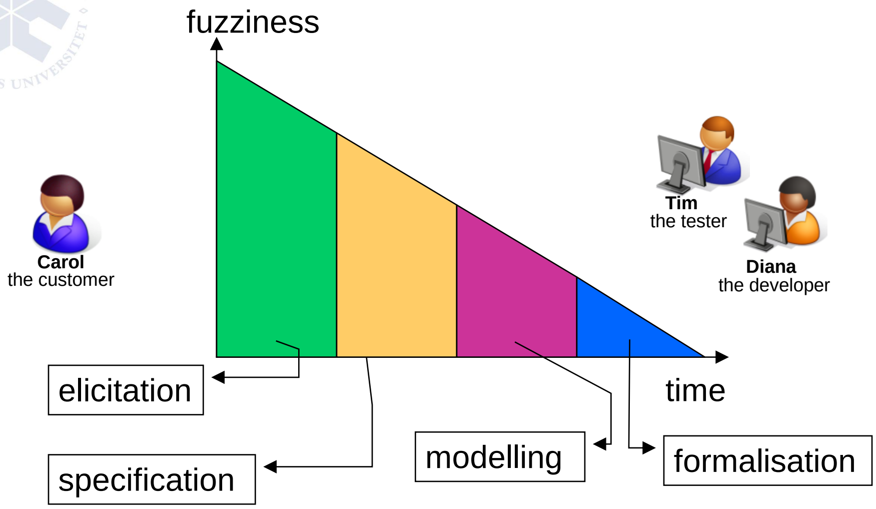
Features

- A distinguishing characteristic of a system item (includes both functional and nonfunctional attributes such as performance and reusability).

(IEEE Std 829)

Higher level stuff:
“The system shall have an SMS-service”

The role of requirements in the life-cycle



fuzziness

Carol
the customer

Tim
the tester

Diana
the developer

elicitation

time

specification

modelling

formalisation

Part I

Requirements Elicitation

Elicitation

need
s
need
s
Carol
the customer
Robert
the requirements engineer

Purpose:

- Understand the **true** needs of the customer
- Trace future implementation to needs

Sources:

- Goals
- Domain knowledge
- Stakeholders
- Environment

Techniques:

- Interviews
- Scenarios
- Prototypes
- Facilitated meetings
- Observation

Interviews

Process:

- Start
- Q & A
- Summary teach-back
- Thank you!
- What's next

Kinds:

- Structured
- Unstructured

Tips

- Be 2 interviewers – shift roles
- Plan the interview
- Don't stick to the plan – use feelings
- Let the customer talk
- Prepare ice-breakers
- Probe thinking
- Look for body language
- Think of human bias
- Why do you get the answers you get?

Part II

Requirements Analysis

Goal

- Detect and resolve conflicts btwn requirements
- Discover bounds of software
- Define interaction with the environment
- Elaborate high-level requirements to derive detailed requirements

Requirements classification

- Functional vs non-functional requirements
- Source
- Product or process requirements
- Priority
- Scope in terms of affected components
- Volatility vs stability

Conceptual Modelling

- Representation in semi-formal notation
- Often diagrammatic representation
- Examples:
  - Object-orientation, use-cases, state-machines
  - Activity diagrams
  - Data flow diagrams
  - Entity-relationship models

Requires a paradigm shift to give full advantage

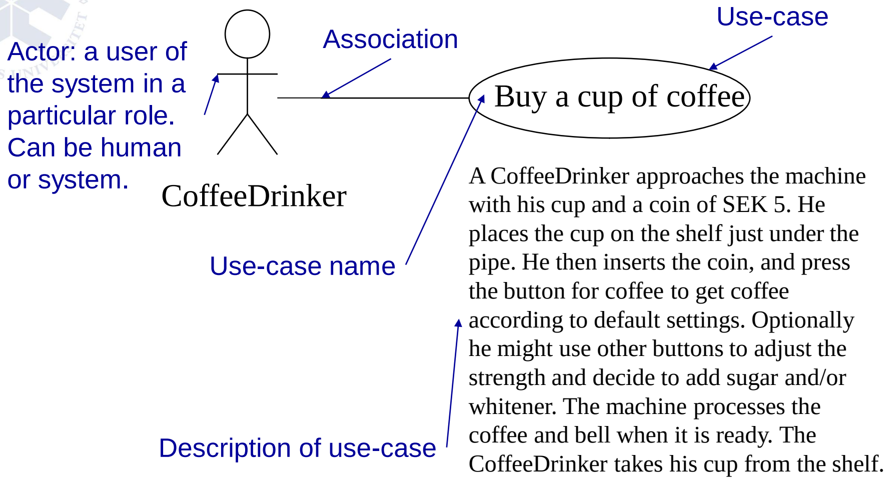
Use-case modelling

A use-case is:

"… a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction of sequence of interrelated events."

Jacobson, m fl 1992: Object-oriented software engineering. Addison-Wesley

Use-case diagram



Actor: a user of the system in a particular role. Can be human or system.

Association

Use-case

Buy a cup of coffee

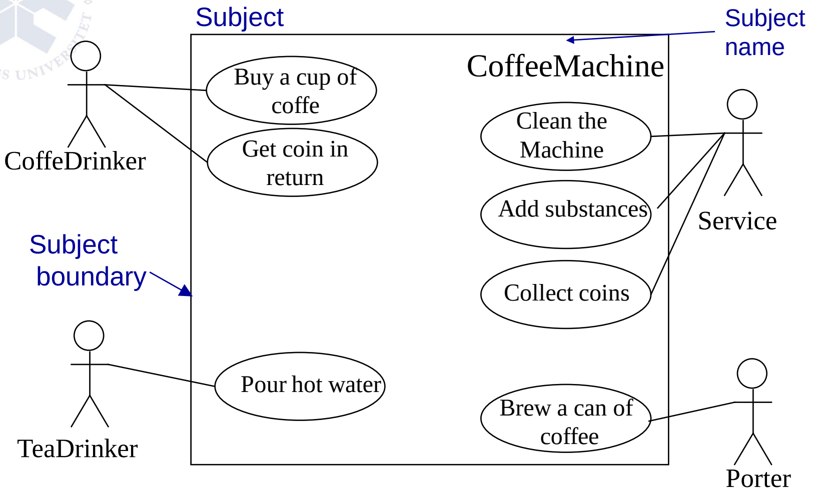
CoffeeDrinker

Use-case name

Description of use-case

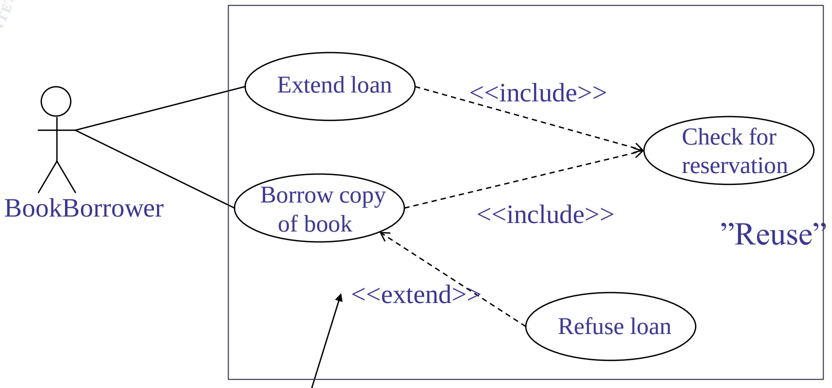
A CoffeeDrinker approaches the machine with his cup and a coin of SEK 5. He places the cup on the shelf just under the pipe. He then inserts the coin, and press the button for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.

Use-case diagram for the coffee-machine



Relations between use-cases

Please, keep as simple as possible.



”Reuse”

Stereotype: extended classification of meaning

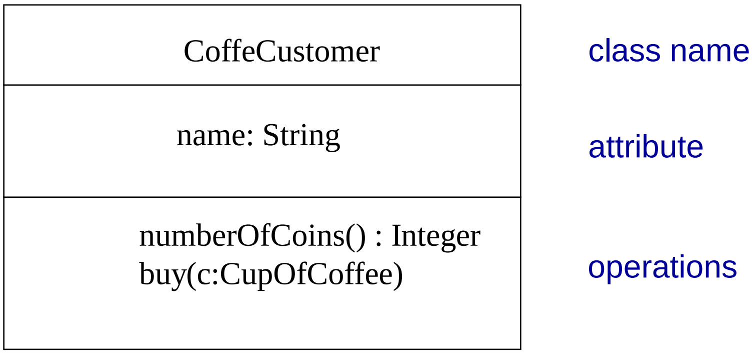
”Separating scenarious”

Identifying classes: noun analysis

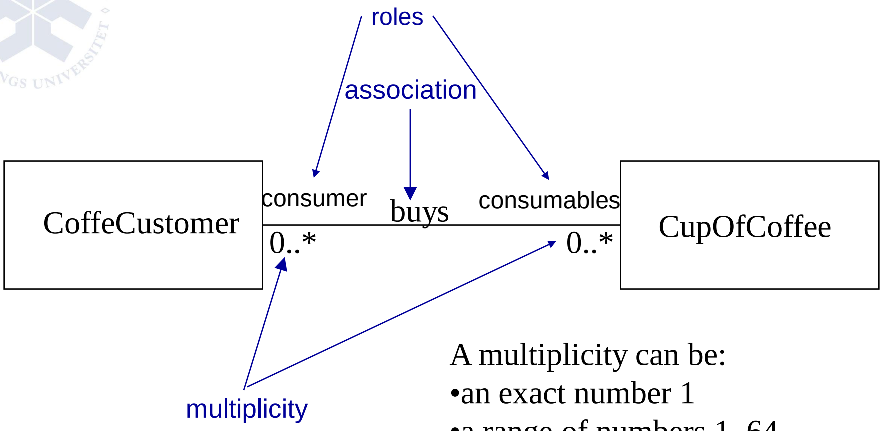
A CoffeeDrinker approaches the machine with his cup and a coin of SEK 5. He places the cup on the shelf just under the pipe. He then inserts the coin, and press the button for coffee to get coffee according to default settings. Optionally he might use other buttons to adjust the strength and decide to add sugar and/or whitener. The machine processes the coffee and bell when it is ready. The CoffeeDrinker takes his cup from the shelf.

- **•machine – real noun handled by the system**
- •cup – unit for beverage
- •coin – detail of user and machine
- •shelf – detail of machine
- •pipe – detail of machine
- **•button– handled by the system**
- •sugar – detail of coffee
- •whitener – detail of coffee
- **•cup of coffee – handled by the system**
- **•indicator – not discovered**

The single class model

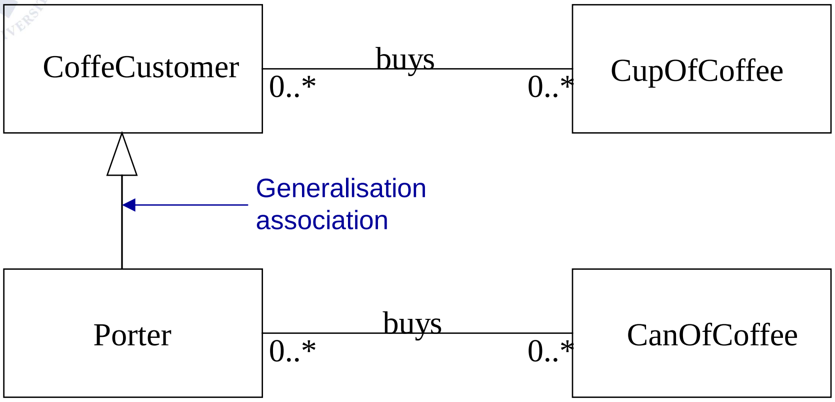


Associations between classes

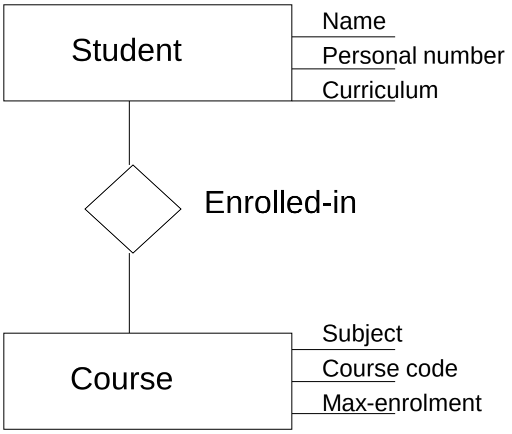


A multiplicity can be:
- an exact number 1
- a range of numbers 1..64
- unspecified number denoted by *

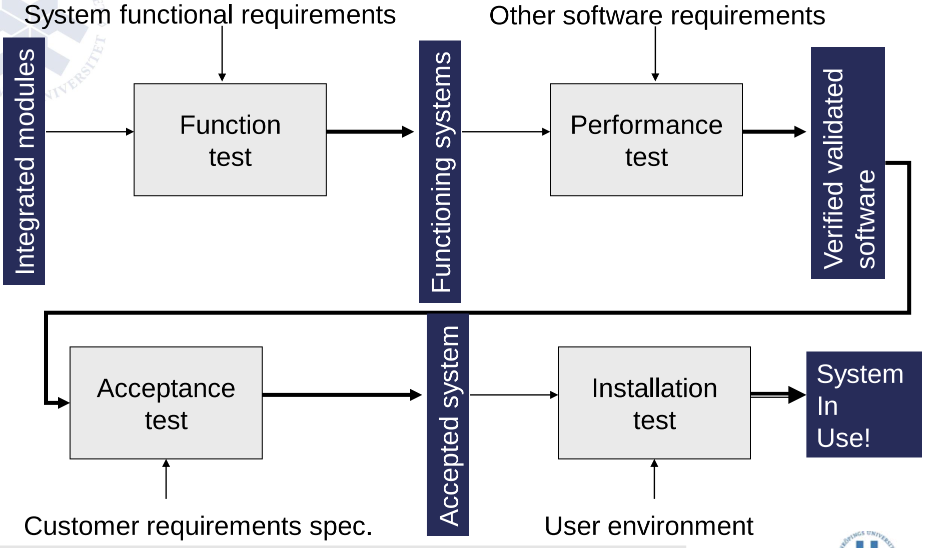
Class model



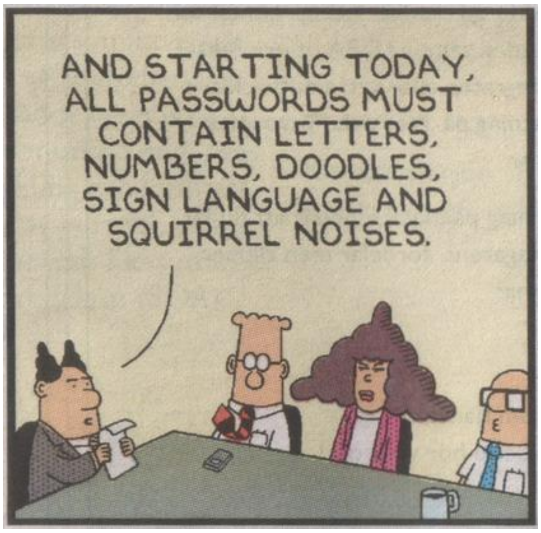
Data model: ER-diagram



Testing non-functional requirements



Usability engineering


AND STARTING TODAY, ALL PASSWORDS MUST CONTAIN LETTERS, NUMBERS, DOODLES, SIGN LANGUAGE AND SQUIRREL NOISES.

Usability

- Relevance
- Efficiency
- Attitude
- Learnability

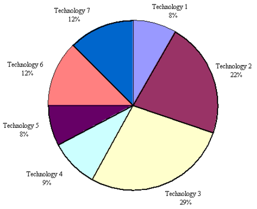
- Usability metrics

Reliability

- The probability that the software executes with no failures during a specified time interval
- Approximation: MTTF/(1+MTTF)
- Example
- Easier to manage: Failure intensity, [failures / hours of execution time]
- Another approximation: λ = (1-R)/t
- Example

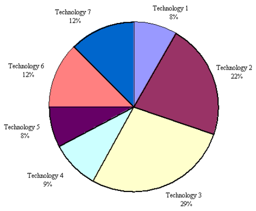
Software reliability engineering

- Define target failure intensity
- Develop operational profile
- Plan tests
- Execute test
- Apply data to decisions



usage

testing



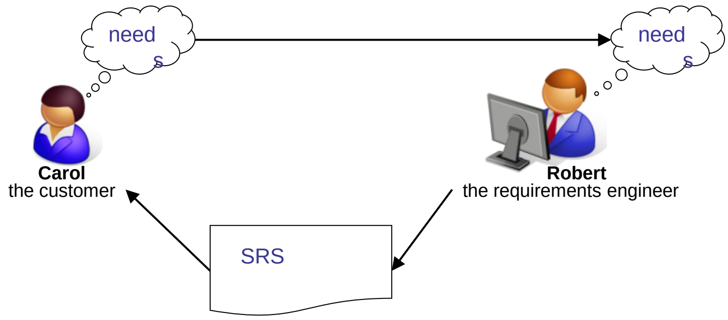
Failure intensity guideline

Impact	Failure intensity	Time btwn failures
Hundreds of deaths, $10^9 cost	10^{-9}	114 000 years
1-2 deaths, $10^6 cost	10^{-6}	114 years
$1000 cost	10^{-3}	6 weeks
$100 cost	10^{-2}	100 h
$10 cost	10^{-1}	10 h
$1 cost	1	1 h

Part III

Requirements specification

Advice towards a good specification



- There is no perfect specification, but you can write a good one
- The RS, or SRS avoids many misunderstandings
- The RS is of special importance in outsourcing programming

SRS contents

1 Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms and abbreviations
- 1.4 References
- 1.5 Overview

2 Overall description
- 2.1 Product perspective
- 2.2 Product functions
- 2.3 User characteristics
- 2.4 General constraints
- 2.5 Assumptions and dependencies
- 2.6 Lower ambition levels

3 Specific requirements

3.1 Interface requirements
- 3.1.1 User interfaces
- 3.1.2 Hardware interfaces
- 3.1.3 Software interfaces
- 3.1.4 Communication interfaces

3.2 Functional requirements

3.3 Performance requirements

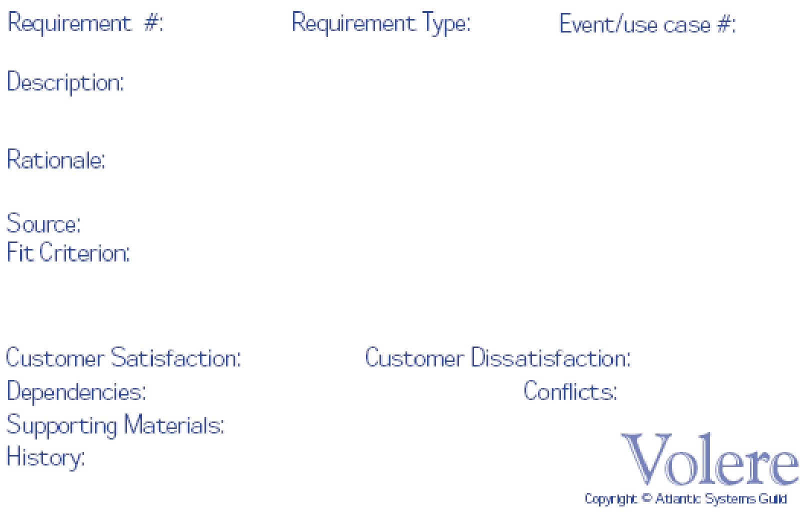
3.4 Design constraints

3.5 Software system attributes

3.6 Other requirements

4 Supporting information
- 4.1 Index
- 4.2 Appendices

Individual requirements



Requirement #: Requirement Type: Event/use case #:

Description:

Rationale:

Source:
Fit Criterion:

Customer Satisfaction: Customer Dissatisfaction:
Dependencies: Conflicts:
Supporting Materials:
History:

Volere
Copyright © Atlantic Systems Guild

Requirements specification

Requirements are:

- Numbered
- Inspected
- Prioritised
- Unambiguous
- Testable
- Complete
- Consistent
- Traceable
- Feasible
- Modifiable
- Useful for:
  - operation
  - maintenance
  - customer
  - developer
  - ….

Define a standard document structure

Benefits:

- Readers can reuse knowledge from previous RSs in understanding
- Writers’ checklist
- Tools can be adapted to generate RSs

Costs:

- Finding the right standard
- Configure variants
- Periodically review standard
- Developers can have a bad attitude against standards

Explain how to use the document

There are many readers of a RS:

- Customers
- Managers
- Software engineers
- Testers
- Maintenance staff
- Technical writers
- Subcontractors

- Part of introduction
- Types of reader
- Technical background needed
- Sections for different readers
- Sections skipped 1st time
- Order of section
- Dependence between section

Takes an hour to write

Include a summary of the requirements

- Better than forward references
- Focus attention on critical and prioritised requirements
- Map to find specific requirements

- Highlight most important requirements in a list
- Table of classification
- Graphic presentation with relations
- Per chapter basis
- Though for large number of requirements

Make a business case for the system

- Helps understanding
- Helps change assessment
- Special document, section or part of introduction
- Requires that top management have an agreement

Define special terms

- Readers and writers might have their own meaning of terms
- Requirements engineer develops a jargon that need to be explained
- Use a glossary, start with a standard one, adapt and maintain
- Highlight terms in the text that can be found in the glossary

Use a data dictionary

- A glossary for variables and terms in diagrams
- Often well-supported in tools
- Often forgotten in student-RSs
- Needs maintenance and adherence
- Can develop into an ontology => massive information exchange, search and checking

- Name of entity
- Aliases
- Type
- Description
- Rationale
- Constraints
  - Units
  - Tolerance
  - Value ranges
  - Error values
- Relations
- Links

Lay out the document for readability

- Many, many readers justify the investment
- Meanwhile, use your standard templates of your word processor and common sense
- It is worthwhile to buy professional training for newly hired personnel

Help readers find information

- Create table of contents
- Create index
- Easy to find support for automatic generation
- Human-made indices are still better

Make documents easy to change

- Requirements will be changed
- Quite easy with tools
- Paper-based specifications needs some thinking:
  - Loose-leaf binders
  - Change bars
  - Short, self-contained chapters
  - Refer to labels, not pages

Requirements database tools

- DOORS
- Focal Point
- Requisite Pro

IBM

- **OSRMT Open Source Requirements Management Tool:**
  - **http://sourceforge.net/projects/osrmt/**
- **Word**
- **Excel**

Formal methods

- Just as models, formal methods is a **complement** to other specification methods.
- Standard is model-based methods, specified mathematically and interpreted with logic.
- Benefits: Non-ambiguous specification, all issues are discovered, proof of properties, simulation, code generation.
- Costs: Time, tools, training and inherent complexity of algorithms.
- High costs ⇒ use only for critical applications

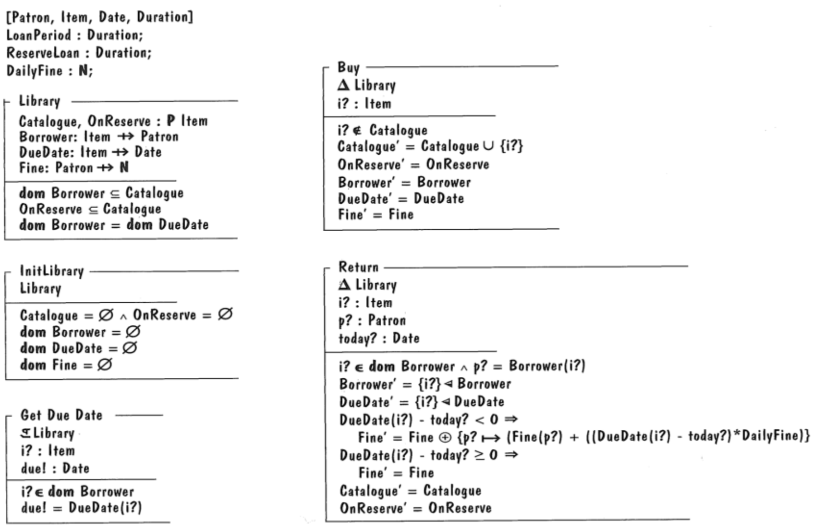
The three Cs - definition

- Consistency – no internal contradictions
- Completeness – everything is there
- Correctness – satisfaction of business goals

Potential problems:

- adding requirements make the specification more complete, but there is a risk of introducing contradiction.
- correctness is vaguely defined,
  formally: consistent + complete?
  pragmatically: satisfaction of customer needs?

Z example



```
[Patron, Item, Date, Duration]
LoanPeriod : Duration;
ReserveLoan : Duration;
DailyFine : ℕ;

┌ Library ─────────────────────────
│ Catalogue, OnReserve : ℙ Item
│ Borrower: Item ⇸ Patron
│ DueDate: Item ⇸ Date
│ Fine: Patron ⇸ ℕ
├──────────────────────
│ dom Borrower ⊆ Catalogue
│ OnReserve ⊆ Catalogue
│ dom Borrower = dom DueDate
└─────────────────────────────────

┌ InitLibrary ─────────────────────
│ Library
├──────────────────────
│ Catalogue = ∅ ∧ OnReserve = ∅
│ dom Borrower = ∅
│ dom DueDate = ∅
│ dom Fine = ∅
└─────────────────────────────────

┌ Get Due Date ──────
│ Ξ Library
│ i? : Item
│ due! : Date
├──────────────────
│ i? ∈ dom Borrower
│ due! = DueDate(i?)
└─────────────────────

┌ Buy ─────────────────────────────
│ Δ Library
│ i? : Item
├──────────────────────
│ i? ∉ Catalogue
│ Catalogue′ = Catalogue ∪ {i?}
│ OnReserve′ = OnReserve
│ Borrower′ = Borrower
│ DueDate′ = DueDate
│ Fine′ = Fine
└─────────────────────────────────

┌ Return ──────────────────────────────────────────
│ Δ Library
│ i? : Item
│ p? : Patron
│ today? : Date
├──────────────────────
│ i? ∈ dom Borrower ∧ p? = Borrower(i?)
│ Borrower′ = {i?} ⩤ Borrower
│ DueDate′ = {i?} ⩤ DueDate
│ DueDate(i?) - today? < 0 ⇒
│    Fine′ = Fine ⊕ {p? ⟼ (Fine(p?) + ((DueDate(i?) - today?)*DailyFine)}
│ DueDate(i?) - today? ≥ 0 ⇒
│    Fine′ = Fine
│ Catalogue′ = Catalogue
│ OnReserve′ = OnReserve
└──────────────────────────────────────────────────
```