Requirements

Software Engineering
Computer Science 520/620

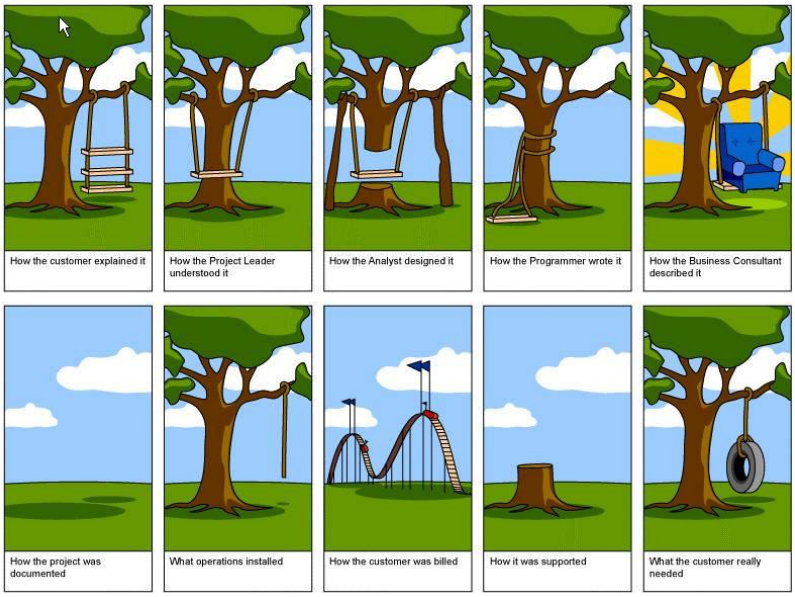
Includes material Adapted from Prof. Leon Osterweil, Prof. Bertrand Meyer and IEEE 830-1998 Standard

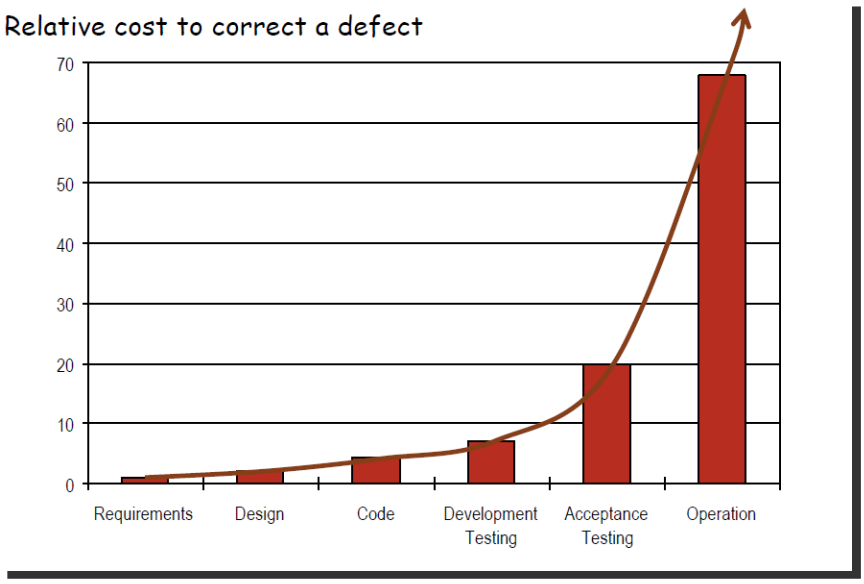
Requirements Specification

"The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later."

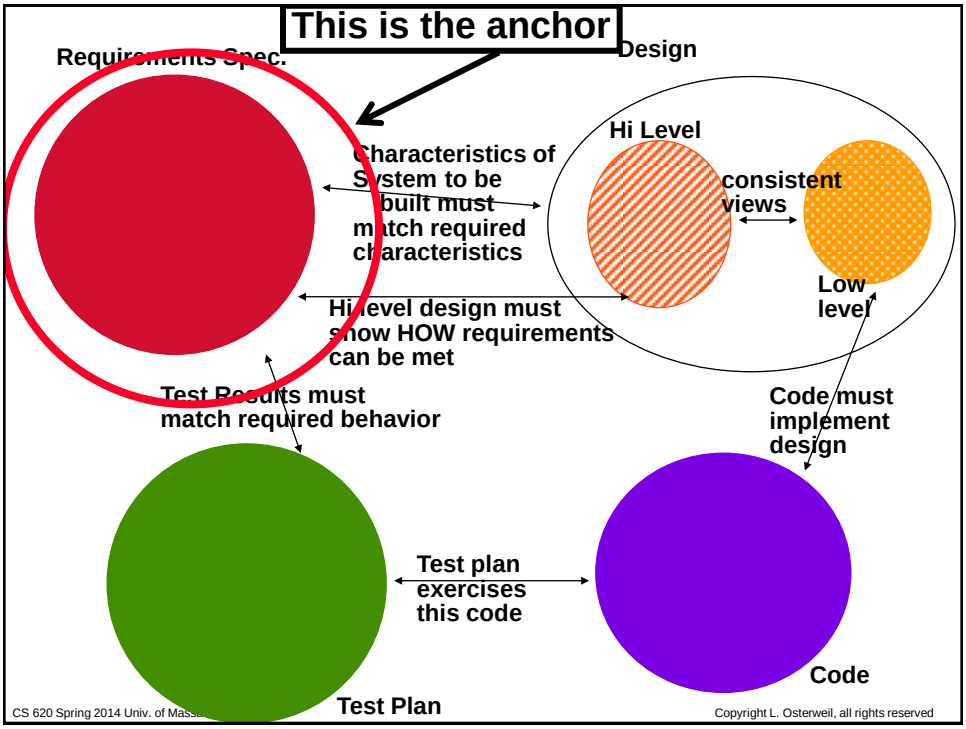
- Fred Brooks: No Silver Bullet - Essence and Accident in Software Engineering, in Computer (IEEE), vol. 20, no. 4, pages 10-19, April 1987.

Recall





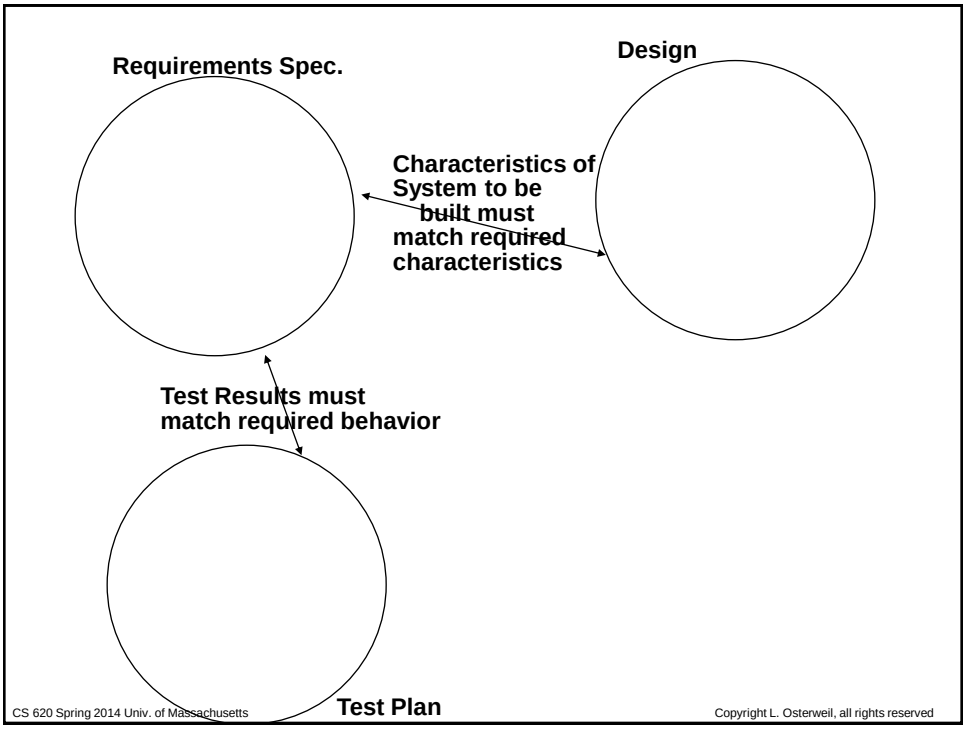
Barry W. Boehm: Software Engineering Economics, Prentice Hall, 1981.

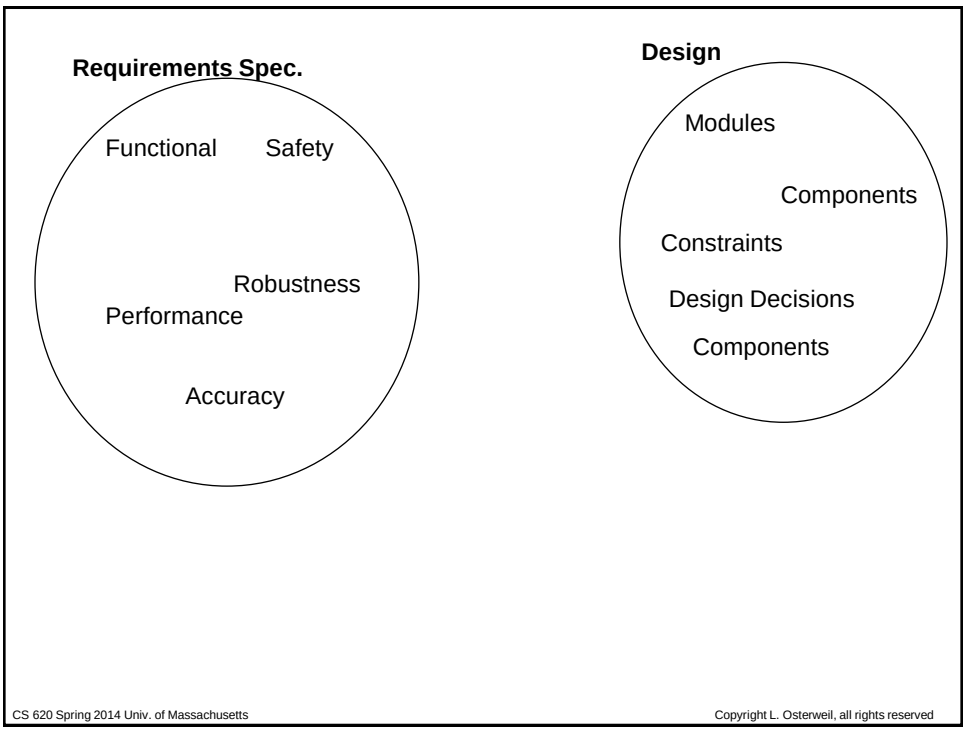


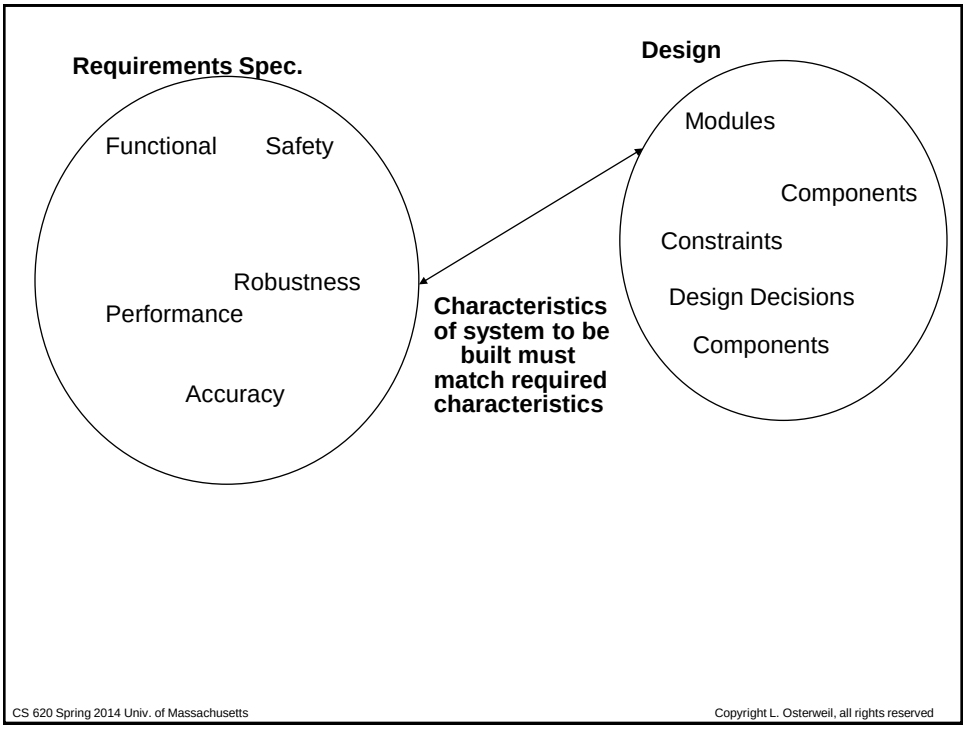
Requirements: Goals and Purposes

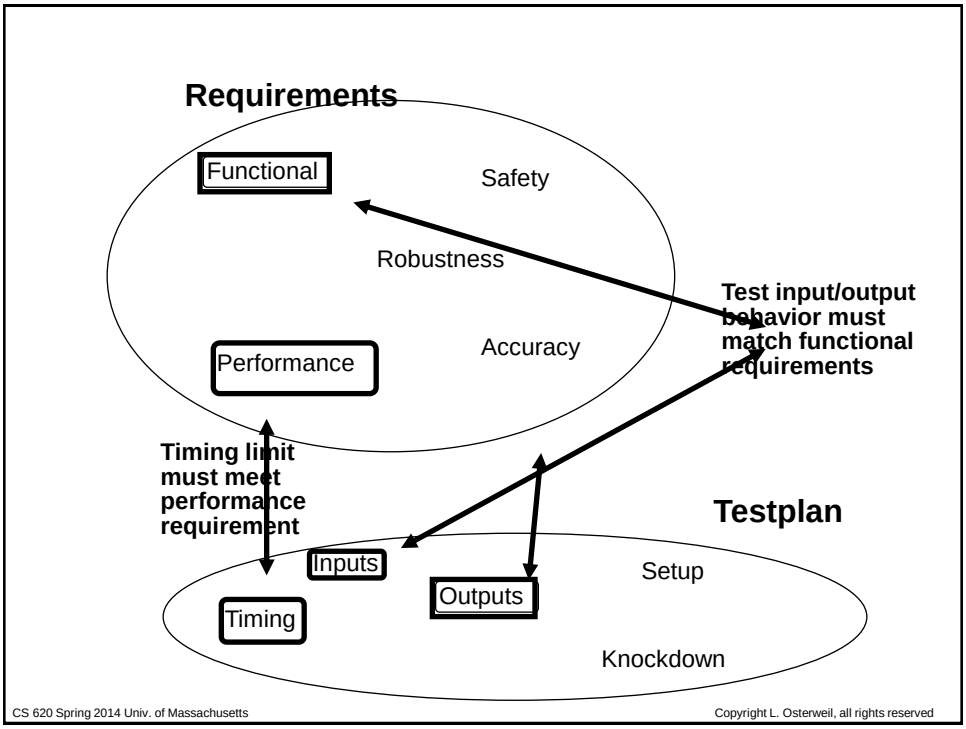
Why "do requirements"?

- Clarify needs before plunging into design
  - Customer "knows" what is wanted
  - But usually doesn't know how to say it
  - Weak sense of what can be achieved
- Clarify acceptance criteria
  - How to know it really delivers what was wanted
  - Decide what the system should not do
- Serve as guide to developers, testers, customers, maintainers
  - "Baselining" requirements









Requirements Specification Driven By Stakeholders and their Questions

- Customers
  - What must it do?
- Developers (eg. designers)
  - What do I have to get it to do?
- Testers
  - What is it supposed to be doing?
  - How would I know it if I saw it?
- Users
  - What is it supposed to do?

- Others???

Requirements Specification Parts

Help stakeholders organize their thoughts about needs by decomposing requirements specification into categories needs and desires.

Some examples:

- Introduction/Background
- Functional
- Environmental
- Performance
- Accuracy
- Robustness
- Security
- Safety

Background/Introduction

Purpose: Give background/context out of which the problem arises, and directions in which it is likely to go

Should contain glossary, references

Should give intuition about problem, domain, existing solutions, components

Probably best written mostly in natural language

Example: UMass has 20,000 students, slow growth next few years
Semester system
Existing system that works, but is not great
Define: FTE, fulltime load, etc.

Functional

Purpose: Indicate the functional transformations that the system will have to compute

Likely to be large and complex, therefore aids to easier and clearer comprehension are needed

Important to state *WHAT* the functions are and not *HOW* they are to be computed

Promising formalisms: dataflow diag., FSM's, UML UC

Usually the chief focus of a requirements specification, and of requirements formalisms--but non-functional requirements are often at least as important

Environmental

Purpose: Indicate the environment in which the software will have to operate

- **On which hardware and software will the software run?**
- **What will be the nature of the user community it will have to support?**
- **With what other manual and automated systems will it have to interface correctly?**

Example: System is to be interactive
Most users to be students
Must run using cellphones, PDAs
Must print reports on existing forms (?)
Must interface successfully with existing student and administrative databases

Performance

Purpose: Specify how much computer and human resource can be allocated to support the execution of the software

How much computer memory can the software use?

How fast must response time be?
--average case
--worst case

How long will users wait for batch runs to terminate?

Example: 2 second response time
overnight printing of all reports
128 Mbytes available on PDAs
500 GBytes of disk available

Accuracy

Purpose: Specify how much tolerance (if any) is acceptable in the results

Most important in numerical computations, but...

Often where "optimality" is defined
eg: what is a "good" game of chess?

Example: Reject scheduling constraints that cause more than 10% of all student requests to be denied

Robustness

Purpose: Specify what sorts of abuse the software will have to resist, and how it will respond

What kinds of "illegal" inputs might be expected, and what should be done about them?

What abnormal environmental conditions might be expected?

Example: System must never corrupt any database
--even after a crash
System must deny illegal requests politely
System must not crash due to
--lack of storage
--user overload

Security

Purpose: Specify which data must be protected, in what ways, from whom, etc.

Usually there are classes of users--what are they?
How to distinguish among the users?

Categories of data too.

Matrix (?) to specify what accesses and permissions different classes and users will have?

Example: Students cannot:
--change course assignments
--cancel courses
--access data on other students
Faculty cannot:
--cancel courses
--change course assignments
Faculty can: access some student data: which?
Administrators can: .... do pretty much anything...

Safety

Purpose: Specify what hazards must be avoided

Specify what the software must NEVER be allowed to do

Has some elements of an inverse or negated set of requirements

Example: System must never divulge credit card data

System must never divulge phone contact data

System must never divulge address or data to unauthorized parties

Not All Go Into All Requirements Specs.

- Some of these may be omitted; some emphasized/deemphasized
- Other sorts of requirements may be added/substituted
  eg: reliability, flexibility, portability.....
- Requirements specification provides information needed to satisfy needs of all stakeholders
- Different stakeholder mixes determine choices of what goes into the requirements spec.

SOME EXAMPLES OF THESE UNDERLYING NEEDS:

- Communication
- Testability
- Precision
- Clarity
- Completeness

Requirement Specification Challenges

- Is it ***Complete***? (to the extent required)
  - Ultimately impossible to be sure about this
- Is it ***Consistent***? (no internal contradictions)
  - Many possible interpretations of this
- Is it un***ambiguous*** ? (possible multiple interpretations)
- Is it sufficently ***precise***?
  - It is possible to be too precise too
- Is it ***Feasible***?
  - If it asks the impossible it would be good to know it
- Is it ***Even?*** (consistent levels of detail)
- Is it ***Understandable***? (what does that mean?)
  - by all stakeholder groups!
- Is there an implementation bias?
- Is there a good basis for proceeding to design?

A Requirement Specification Is Never Perfect in All (Any?) Aspects

- **Imperfections are often understandable, tolerable, unavoidable**
- **Look at real underlying stakeholder needs for the requirements specification (communication, clarity, precision, modifiability....??)**
- **Plan requirements content, structure, relations to meet these needs**
- **Requirements specification medium is crucial in helping assure needs are met**
- **Select requirements specification medium to address needs**

Natural Language Prose Requirements Specification

- **Write requirements in "plain English"**
- **Build upon universal base of understanding of natural language**
- **Possible to augment with defined terms**
- **Use of punctuation for clarification**
- **Text and word processing systems help automate/ maintain/alter**

Examples:

All input data sets will be terminated with an end of file record
System will respond to service requests within 2 seconds
System will have a friendly user interface
System will never go into an infinite loop

Problem: How to reason about a natural language reqts. spec?
How to determine: completeness, unambiguity, etc.?

Disciplined Use of Natural Language

- Natural response to problems of:
  --imprecision
  --ambiguity
  --consistency (especially when due to size)
- Familiar approaches:
  --Restricted use of defined terms
  --Introduction of structuring (paragraph numbering, outline form, templates, etc.)
- Other, earlier examples of disciplined use of natural language:
  --Legal documents
  --Recipes

Data Base Approaches

- Requirement items stored as database entries
- Queries to retrieve information
- Database tools to check for consistency

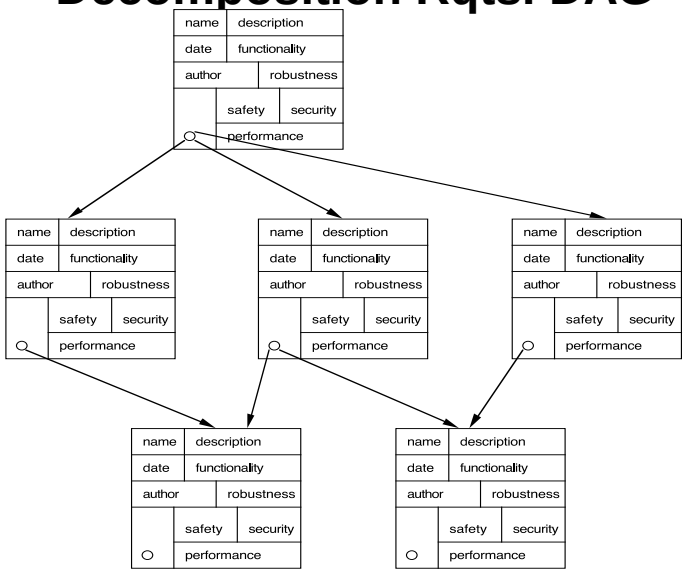
PSL (Relational Database Organization)

```
DESCRIPTION:
     this process performs those actions needed to interpret
     time cards to produce a pay statement for each hourly employee.;
KEYWORDS:      independent;
ATTRIBUTES ARE:
     complexity-level      high;
GENERATES:      pay-statement, error-listing;
RECEIVES:       time-card;
SUBPARTS ARE:      hourly-paycheck-validation, hourly-emp-update,
     h-report-entry-generates, hourly-paycheck-production;
PART OF:  payroll-processing;
DERIVES:  pay-statement;
USING:  time-card, hourly-employee-record;
DERIVES:  hourly-employee-report;
USING:  time-card, hourly-employee-record;
DERIVES:   error-listing;
USING:   time-card, hourly-employee-record;
PROCEDURE:  <<not usually included in a requirements spec.>>
HAPPENS:  number-of-payments TIMES-PER pay-period;
TRIGGERED BY:  hourly-emp-processing-event;
TERMINATION-CAUSES:  new-employee-processing-event;
SECURITY IS:  company-only;
```

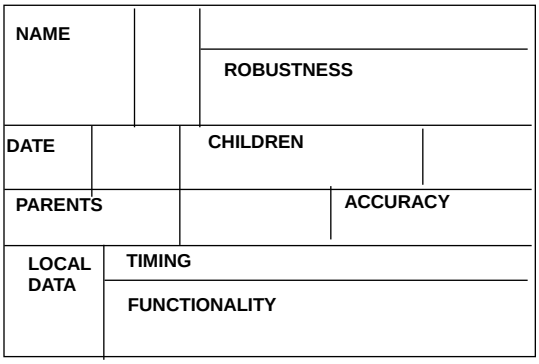

Hierarchical Decomposition Organization

- Requirements Specification as hypertext
- Structure (DAG) of Requirements Elements
  - Child element represents *part-of* relation
- Requirement Element is a record
- Requirement Element fields carry information as:
  - Instances of preset types
  - Instances related to others by relations
    -- express consistency rules
    -- define consistency determination
    -- define inconsistency remediation
  - Relations among
    -- Requirement elements
    -- Requirement elements and parts of other artifacts
    (e.g., testplan elements, other rqts. representations)

Functional
Decomposition Rqts. DAG



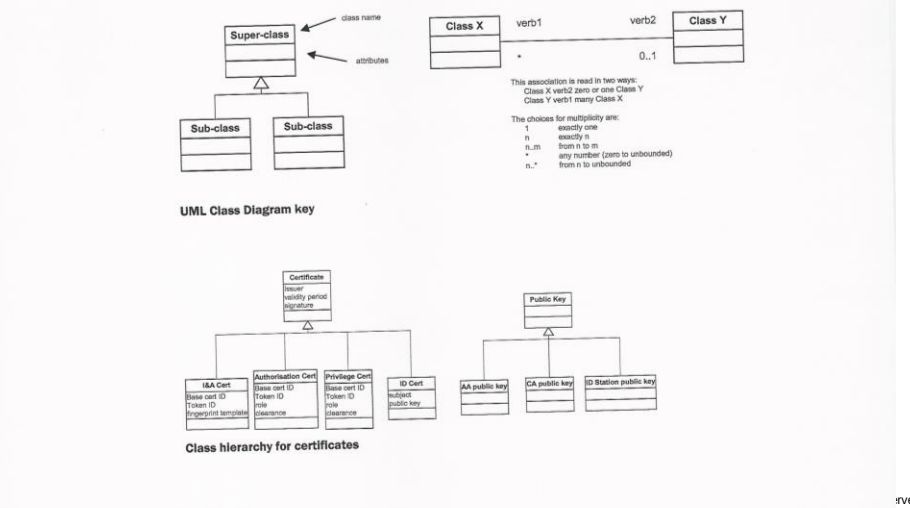
Requirement Element: An Example Structure



Multirepresentation Systems

- Have seen that different representations are of different uses
- One diagram may be useful in different ways to different stakeholders
- But most stakeholders require a variety of diagrams
- Several different diagrams can be expected to be needed to satisfy the different stakeholders
- Problems with different views/diagrams
  - Are they all representing the same software product?
  - How to assure that they are all consistent with each other?
  - If the product changes, then ALL views must change correspondingly

Artifacts are needed in order to specify functional/behavioral requirements



Deliverables of the Requirement Phase

- **Requirement document**
- **Development plan**
- **Test Plan (at least a draft)**

The IEEE 830-1998 Standard

- ”IEEE Recommended Practice for Software Requirements Specifications”
- Approved 25 June 1998 (revision of earlier standard)
- Descriptions of the content and the qualities of a good software requirements specification (SRS).
- Goal: “The SRS should be correct, unambiguous, complete, consistent, ranked for importance and/or stability, verifiable, modifiable, **traceable**.”

Recommended document structure

1. Introduction
1.1 Purpose
1.2 Scope
1.3 Definitions, acronyms, and abbreviations (Glossary!)
1.4 References
1.5 Overview
2. Overall description
2.1 Product perspective
2.2 Product functions
2.3 User characteristics
2.4 Constraints
2.5 Assumptions and dependencies
3. Specific requirements
Appendixes
Index

Scope section

- Identify software product to be produced by name (e.g., Host DBMS, Report Generator, etc.)
- Explain what the product will and will not do
- Describe application of the software: goals and benefits
- Establish relation with higher-level system requirements if any

Product Perspective Section

Describe relation with other products if any.
Examples:

- System interfaces
- User interfaces
- Hardware interfaces
- Software interfaces
- Communications interfaces
- Memory
- Operations
- Site adaptation requirements

Constraints Section

Describe any properties that will limit the developers' options
Examples:

- Regulatory policies
- Hardware limitations (e.g., signal timing requirements)
- Interfaces to other applications
- Parallel operation
- Audit functions
- Control functions
- Higher-order language requirements
- Reliability requirements
- Criticality of the application

Specific Requirements Section

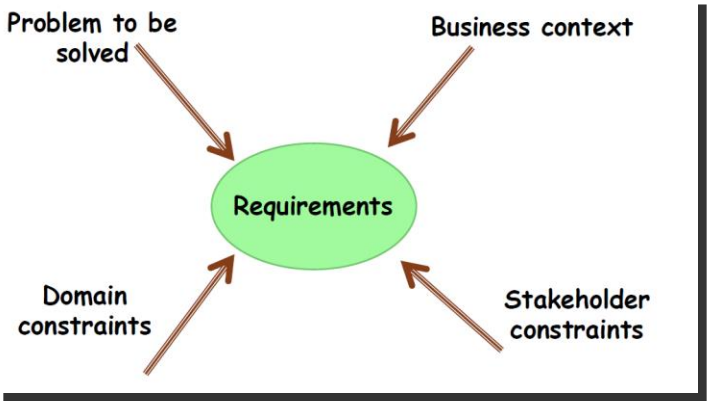
This section brings requirements to a level of detail making them usable by designers and testers.
Examples:

- Details on external interfaces
- Precise specification of each function
- Responses to abnormal situations
- Detailed performance requirements
- Database requirements
- Design constraints
- Specific attributes such as reliability, availability,security, portability

Specific Requirements Section: Example

3. Specific requirements
3.1 External interfaces
    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 Quality requirements
…
3.6 Other requirements

And Remember…



Gerald Kotonya & Ian Sommerville: Requirements Engineering: Processes and Techniques, Wiley, 1998

Key Lessons

It's not programming:

- Programming describes a solution and not a problem
- Programming is constructive

It's not design:

- We do not only describe the software
- We describe the full system (software and environment)
- No separation between software and environment
- We do so in an incremental way
- We want to understand the system

Key Lessons

- Identify & involve all stakeholders
- Requirements determine not just development but tests
- Use cases are good for test planning
- Requirements should be abstract
- **Requirements should be traceable**
- Requirements should be verifiable (otherwise they are wishful thinking)

Object technology helps

- Modularization
- Classifications

Some Papers in Requirements Engineering

- **D. Harel. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming. 8 (1987).**
- **B. Nuseibeh and S. Easterbrook. Requirements Engineering: A Roadmap. Proceedings of 22nd International Conference on Software Engineering (ICSE 2000) June 2000. Limerick, Ireland. ACM Press**
- **B. Nuseibeh, J. Kramer, and A. Finkelstein. A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification. IEEE Transactions on Software Engineering, 20 10 (1994).**
- **C. L. Heitmeyer. Software Cost Reduction. Encyclopedia of Software Engineering. J.J. Marciniak, editor. ISBN: 0-471-02895-9. January 2002.**
- **A. van Lamsweerde. Requirements Engineering in the Year 00: A Research Perspective. Proceedings of 22nd International Conference on Software Engineering (ICSE 2000) June 2000. Limerick, Ireland. ACM Press**
- **A. van Lamsweerde. Goal-Oriented Requirements Engineering: A Guided Tour. 5th IEEE International Symposium on Requirements Engineering. Toronto, Canada, August 2001.**