7. Requirements for distributed & outsourced projects*

*Part of the material is from our “Requirements Engineering” course
Requirements engineering

Essential in good software projects
Even more essential in oursourcing!
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.
Relative cost to correct a defect

Source: Boehm 81
A definition

“A requirement” is a statement of desired behavior for a system

“The requirements” for a system is the collection of all such individual requirements
Goals of performing requirements

Understand problem or problems that the eventual software system, if any, should solve
Prompt relevant questions about problem & system
Provide basis for answering questions about specific properties of problem & system
Decide what system should do
Decide what system should not do
Ascertained that system will satisfy the needs of its stakeholders
Provide basis for development of system
Provide basis for V & V* of system

*Validation & Verification, including testing
Another view

Calls for tenders, proposal evaluation

Project estimations (size, cost, schedules)

Prototypes

Acceptance tests

QA check lists

Implementation directives

Requirement document

Contract

Project work plan

Follow-up directives

Software architecture

Evolution directives

Documentation, user manuals

After: van Lamsweerde 08

175
Benefits of a good requirements process

- Fewer requirements defects
- Reduced development rework
- Fewer unnecessary features
- Lower enhancement costs
- Faster development
- Fewer miscommunications
- Less scope creep
- Better project organization
- More accurate testing estimates
- Better testing process
- Higher team satisfaction
Do not forget that the requirements also determine the test plan
Possible requirements **stakeholders**

- Clients (bespoke system)
- Customers (product for general sale)
- Clients’ and customers’ customers
- Users
- Domain experts
- Market analysts
- Unions?

- Legal experts
- Purchasing agents
- Software developers
- Software project managers
- Software documenters
- Software testers
- Trainers
- Consultants
15 quality goals for requirements

- Justified
- Correct
- Complete
- Consistent
- Unambiguous
- Feasible
- Abstract
- Traceable

- Delimited
- Interfaced
- Readable
- Modifiable
- Verifiable
- Prioritized*
- Endorsed

Marked attributes are part of IEEE 830, see below
* “Ranked for importance and/or stability”
Difficulties of requirements

Natural language and its imprecision

Formal techniques and their abstraction

Users and their vagueness

Customers and their demands

The rest of the world and its complexity
The two constant pitfalls

Comitting too early to an implementation

Overspecification!

Missing parts of the problem

Underspecification!
The requirements process

Source: Pfleeger & Atlee 05

The process of determining requirements
A spiral model

From: van Lamsweerde 08

Domain understanding & requirements elicitation

Evaluation & negotiation

Consolidated requirements

Agreed requirements

Start

Specification & documentation

Quality assurance

Documented requirements

Alternative proposals
Requirements elicitation: who?

Users/customers?

Software developers?

Requirements engineers (analysts)?
Requirements elicitation: what?

Example questions:
What will the system do?
What must happen if...?
What resources are available for...?
What kind of documentation is required?
What is the maximum response time for...?
What kind of training will be needed?
What precision is requested for...?
What are the security/privacy implications of ...?
Is ... an error?
What should the consequence be for a ... error?
What is a criterion for success of a ... operation?
Requirements elicitation: how?

Contract
User interviews
Requirements workshops
Study of existing non-computer processes
Study of existing computer systems
Study of comparable systems elsewhere
## Stereotypes

### How developers see users
- Don't know what they want
- Can't articulate what they want
- Have too many needs that are politically motivated
- Want everything right now.
- Can't prioritize needs
- Refuse to take responsibility for the system
- Unable to provide a usable statement of needs
- Not committed to system development projects
- Unwilling to compromise
- Can't remain on schedule

### How users see developers
- Don't understand operational needs
- Too much emphasis on technicalities.
- Try to tell us how to do our jobs
- Can't translate clearly stated needs into a successful system
- Say no all the time
- Always over budget
- Always late
- Ask users for time and effort, even to the detriment of users' primary duties
- Set unrealistic standards for requirements definition
- Unable to respond quickly to legitimately changing needs

*Source: Scharer 90*
The two parts of requirements

Purpose: to capture the user needs for a “machine” to be built

Define success as

\[
\text{machine specification} \land \text{domain properties} \implies \text{requirement}
\]

- **Domain properties**: outside constraints (e.g., can only modify account balance as a result of withdrawal or deposit)
- **Requirement**: desired system behavior (e.g., withdrawal of \( n \) francs decreases balance by \( n \))
- **Machine specification**: desired properties of the machine (e.g., request for withdrawal will, if accepted, lead to update of balance)
Components of requirements

1. Domain properties

2. Functional requirements

3. Non-functional requirements (reliability, security, accuracy of results, time and space performance, portability...)

4. Requirements on process and evolution
How to ensure good requirements?

Managerial aspects:
- Involve all stakeholders
- Establish procedures for controlled change
- Establish mechanisms for traceability
- Treat requirements document as one of the major assets of the project; focus on clarity, precision, completeness

Technical aspects: how to be precise?
- Formal methods?
- Design by Contract
"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 practice for Software Requirements Specifications

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
Use cases

One of the UML diagram types

A use case describes how to achieve a single business goal or task through the interactions between external actors and the system

A good use case must:

- Describe a business task
- Not be implementation-specific
- Provide appropriate level of detail
- Be short enough to implement by one developer in one release
Use case example

Place an order:
Browse catalog & select items
Call sales representative
Supply shipping information
Supply payment information
Receive confirmation number from salesperson

May have precondition, postcondition, invariant
Note on use cases

Use cases cannot suffice to define the requirements:
- Not abstract enough
- Too specific
- Describe current processes
- Do not support evolution

Use cases are to requirements what tests are to software specification and design

Major application: for testing
Requirements: key lessons

Requirements are software
- Subject to software engineering tools
- Subject to standards
- Subject to measurement
- Part of quality enforcement

Requirements is both a lifecycle phase and a lifecycle-long activity

Since requirements will change, seamless approach is desirable

Distinguish domain properties from machine properties
- Domain requirements should never refer to machine requirements!
Key lessons (continued)

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
Object technology helps
  - Modularization
  - Classifications
  - Contracts
  - Seamless transition to rest of lifecycle
Requirements in a distributed/outsourced setting

Special points to consider:

- Include both customer and supplier in requirements process
- Use distributed meeting techniques, centered on documents (see discussion of weekly meeting and code review)
- Enforce endorsement by all major stakeholder representatives
- Pay special attention to document structure and configuration management
- Perform a special step of consistency checking between requirements and legal documents (contract, supplier agreement management)
- Pay special attention to glossary, include translation if appropriate
- Pay special attention to the change process
- Consider including a prototype implementation as a validation of requirements