

Object-Oriented Software Engineering

Practical Software Development using UML and Java

Chapter 4: Developing Requirements

4.1 Domain Analysis

The process by which a software engineer learns about the domain to better understand the problem:

The *domain* is the general field of business or technology in which the clients will use the software
A *domain expert* is a person who has a deep knowledge of the domain

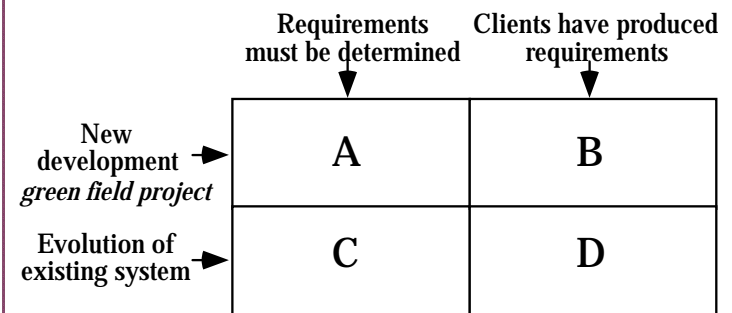
Benefits of performing domain analysis:

- Faster development
- Better system
- Anticipation of extensions

Domain Analysis document

- A. Introduction
- B. Glossary
- C. General knowledge about the domain
- D. Customers and users
- E. The environment
- F. Tasks and procedures currently performed
- G. Competing software
- H. Similarities to other domains

4.2 The Starting Point for Software Projects



4.3 Defining the Problem and the Scope

A problem can be expressed as:

A *difficulty* the users or customers are facing,

Or as an *opportunity* that will result in some benefit such as improved productivity or sales.

The solution to the problem normally will entail developing software

A good problem statement is short and succinct

Defining the Scope

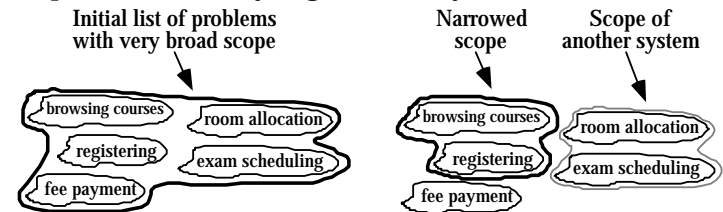
Narrow the *scope* by defining a more precise problem

List all the things you might imagine the system doing

—Exclude some of these things if too broad

—Determine high-level goals if too narrow

Example: A university registration system



4.4 What is a Requirement

Requirement: A statement about the proposed system that all stakeholders agree must be made true in order for the customer's problem to be adequately solved.

Short and concise piece of information

Says something about the system

All the stakeholders have agreed that it is valid

It helps solve the customer's problem

A collection of requirements is a *requirements document*.

4.5 Types of Requirements

Functional requirements

Describe *what* the system should do

Non-functional requirements

Constraints that must be adhered to during development

Functional requirements

What *inputs* the system should accept

What *outputs* the system should produce

What data the system should *store* that other systems might use

What *computations* the system should perform

The *timing and synchronization* of the above

Non-functional requirements

All must be verifiable

Three main types

1. Categories reflecting: usability, efficiency, reliability, maintainability and reusability

—Response time

—Throughput

—Resource usage

—Reliability

—Availability

—Recovery from failure

—Allowances for maintainability and enhancement

—Allowances for reusability

Non-functional requirements

2. Categories constraining the *environment and technology* of the system.

—Platform

—Technology to be used

3. Categories constraining the *project plan and development methods*

—Development process (methodology) to be used

—Cost and delivery date

- Often put in contract or project plan instead

4.6 Some Techniques for Gathering and Analysing Requirements

Observation

Read documents and discuss requirements with users

Shadowing important potential users as they do their work

—ask the user to explain everything he or she is doing

Session videotaping

Interviewing

Conduct a series of interviews

—Ask about specific details

—Ask about the stakeholder's vision for the future

—Ask if they have alternative ideas

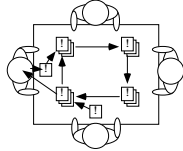
—Ask for other sources of information

—Ask them to draw diagrams

Gathering and Analysing Requirements...

Brainstorming

- Appoint an experienced moderator
- Arrange the attendees around a table
- Decide on a 'trigger question'
- Ask each participant to write an answer and pass the paper to its neighbour



Joint Application Development (JAD) is a technique based on intensive brainstorming sessions

Gathering and Analysing Requirements...

Prototyping

- The simplest kind: *paper prototype*.
 - a set of pictures of the system that are shown to users in sequence to explain what would happen
- The most common: a mock-up of the system's UI
 - Written in a rapid prototyping language
 - Does *not* normally perform any computations, access any databases or interact with any other systems
 - May prototype a particular aspect of the system

Gathering and Analysing Requirements...

Informal use case analysis

- Determine the classes of users that will use the facilities of this system (actors)
- Determine the tasks that each actor will need to do with the system

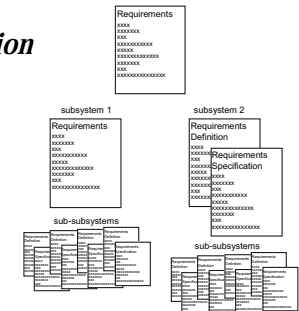
More on use cases in Chapter 7

4.7 Types of Requirements Document

Two extremes:

- An informal outline of the requirements using a few paragraphs or simple diagrams
requirements definition
- A long list of specifications that contain thousands of pages of intricate detail
requirements specification

Requirements documents for large systems are normally arranged in a hierarchy



Level of detail required in a requirements document

How much detail should be provided depends on:

- The size of the system
- The need to interface to other systems
- The readership
- The stage in requirements gathering
- The level of experience with the domain and the technology
- The cost that would be incurred if the requirements were faulty

4.8 Reviewing Requirements

Each individual requirement should

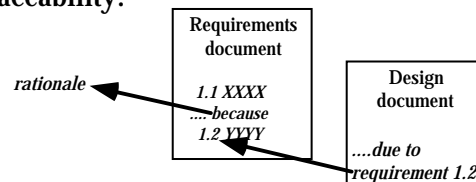
- Have benefits that outweigh the costs of development
- Be important for the solution of the current problem
- Be expressed using a clear and consistent notation
- Be unambiguous
- Be logically consistent
- Lead to a system of sufficient quality
- Be realistic with available resources
- Be verifiable
- Be uniquely identifiable
- Does not over-constrain the design of the system

Requirements documents...

The document should be:

- sufficiently complete
- well organized
- clear
- agreed to by all the stakeholders

Traceability:



Requirements document...

- Problem
- Background information
- Environment and system models
- Functional Requirements
- Non-functional requirements

4.9 Managing Changing Requirements

Requirements change because:

- Business process changes
- Technology changes
- The problem becomes better understood

Requirements analysis never stops

- Continue to interact with the clients and users
- The benefits of changes must outweigh the costs.
 - Certain small changes (e.g. look and feel of the UI) are usually quick and easy to make at relatively little cost.
 - Larger-scale changes have to be carefully assessed
 - Forcing unexpected changes into a partially built system will probably result in a poor design and late delivery

Some changes are enhancements in disguise

- Avoid making the system *bigger*, only make it *better*

4.13 Difficulties and Risks in Domain and Requirements Analysis

Lack of understanding of the domain or the real problem

- Do domain analysis and prototyping*

Requirements change rapidly

- Perform incremental development, build flexibility into the design, do regular reviews*

Attempting to do too much

- Document the problem boundaries at an early stage, carefully estimate the time*

It may be hard to reconcile conflicting sets of requirements

- Brainstorming, JAD sessions, competing prototypes*

It is hard to state requirements precisely

- Break requirements down into simple sentences and review them carefully, look for potential ambiguity, make early prototypes*