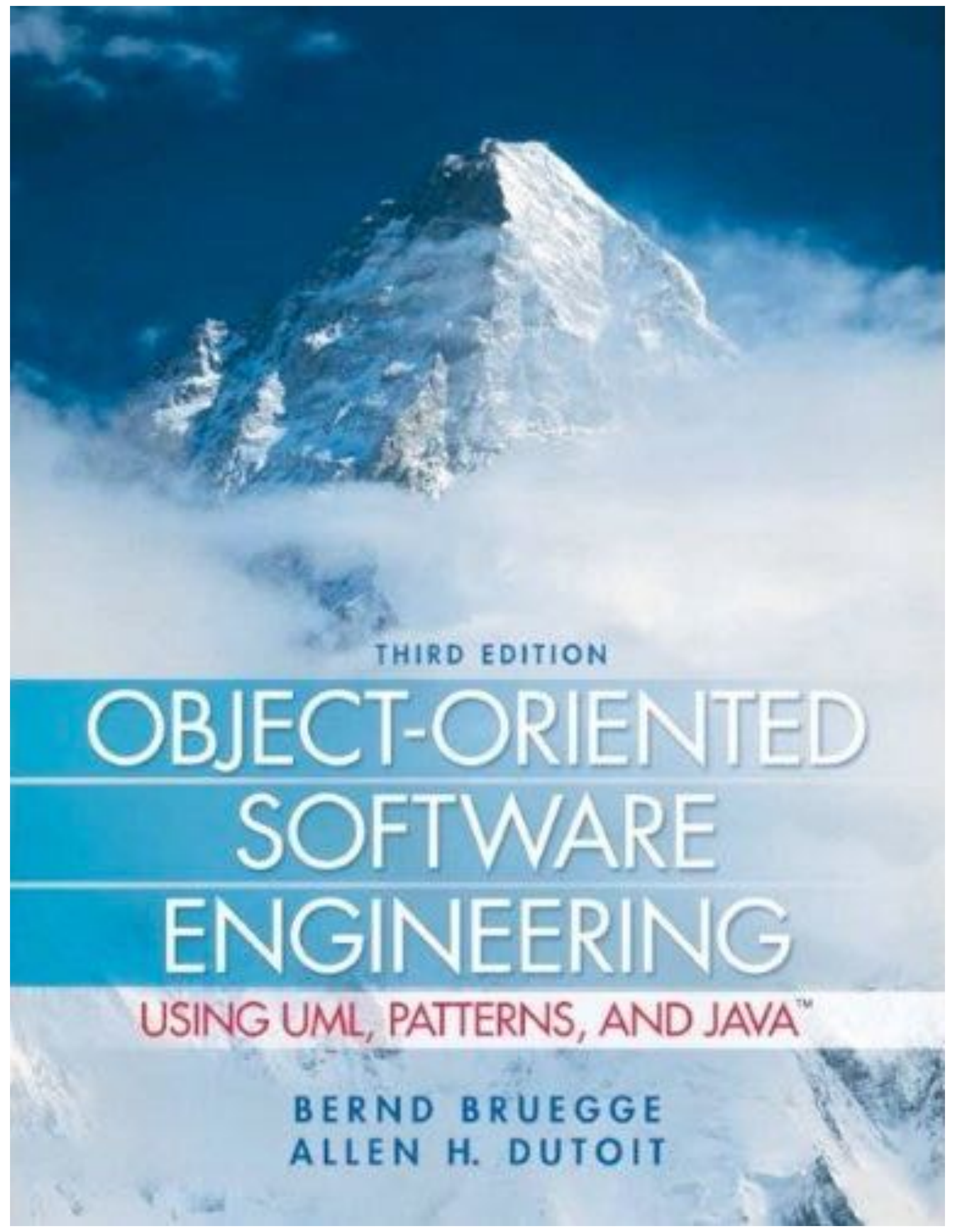
Software Engineering Essentials

Requirements Elicitation

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics

Literature

This unit **requirements elicitation** is based on chapter 4 of the OOSE Book:


THIRD EDITION
OBJECT-ORIENTED SOFTWARE ENGINEERING
USING UML, PATTERNS, AND JAVA™
BERND BRUEGGE
ALLEN H. DUTOIT

Bernd Bruegge, Allen Dutoit: **Object-Oriented Software Engineering Using UML, Patterns, and Java** (3rd edition)

Learning Goals

1) Understand what requirements elicitation is and why its needed

2) Understand the main activities of requirements elicitation

3) Apply the main activities of requirements elicitation

4) Understand the terms: problem statement and requirements

5) Apply heuristics to create scenarios and use cases

Purpose of Requirements Elicitation

Requirements elicitation focuses on describing the **purpose** of the system.

The client, the developers, and the users **identify a problem area** and **define a system** that addresses the problem.

Such a definition is called a **requirements specification** and serves as a contract between the client and the developers.

The **requirements specification** is structured and formalized during **analysis.**

will be covered in unit **Analysis**

Problem Statement

Describes requirements, subsystem decomposition, and communication infrastructure

Describes the current situation, the functionality it should support, and environment in which the system will be deployed

Contains **mutual understanding** of the problem to be addressed by the system

Its neither a *precise* nor *complete* specification but a high level summary of the **customer's** wishes

Typically described by two documents the Requirements Analysis Document (RAD) and the Software Project Management Plan (SPMP)

Requirements

Requirements are textual descriptions of the desired functionality of the system and its properties. They can be divided into two major categories applying the acronym FURPS:

precise, *correct, complete, consistent, unambiguous*

Functional requirements: (F)

A single function of a system or its component

e.g. “A student can see all courses of the current semester in his major and minor subject.”

Non-functional requirements (URPS):

Aspects of the system that are not directly related to its functional behavior

e.g. “interface look, response time, security issues”

Examples of functional requirements

FR1: Search for available courses: A student can see all courses of the current semester in his major and minor subject. He is able to join the course which saves it into his course list. He can also drop a course.

FR2: Check course details: A student can see details about a course such as the course times, the location of the lecture hall on a map and other course attendees including their name and picture.

FR3: Update profile: A student can update his profile settings and his profile picture. He can also change the notification settings.

FR4: Add comments: A student can add comments about a course and thus start a discussion. Others can like the comment and write follow-up comments.

Non-functional requirements

Also called quality requirements and can be described by the acronym URPS

1) **U**sability: human factors, aesthetics, consistency, documentation, responsiveness
2) **R**eliability: availability, failure frequency, robustness
3) **P**erformance: speed, efficiency, resource consumption
4) **S**upportability: maintainability, testability, flexibility

Examples of non-functional requirements

NFR1: The app should be intuitive to use and the user interface should be easy to understand. All interactions should be completed in less than three clicks.

NFR2: Conformance to guidelines: The design of the app should conform to the usability guidelines for the chosen operating system.

NFR3: Target platform: The app has to be developed in Java.

NFR4: Backend system: The customer provides a backend system with a couple of services that have to be used in the app.

Constraints

Also called pseudo requirements:

1) **Implementation:** constraints on the implementation of the system, including the use of specific tools, programming languages, or hardware platforms.
2) **Interface:** constraints imposed by external systems, including legacy systems and interchange formats
3) **Operations:** constraints on the administration and management of the system in the operational setting
4) **Packaging:** constraints on the actual delivery of the system (e.g., constraints on the installation media for setting up the software).
5) **Legal:** concerned with licensing, regulation, and certification issues

Activities during Requirements Elicitation

Identifying actors: Identify the different types of users the future system will support

Identifying scenarios: Develop a set of detailed scenarios for typical functionality provided by the future system

Identifying use cases: Derived from scenarios a set of use cases that completely represent the future system is created

Activities during Requirements Elicitation

Refining use cases: Detailing each use case and describing the behavior of the system in the presence of errors and exceptional conditions

Identifying relationships among use cases: Identify dependencies among use cases found during “identifying use cases”

Identifying nonfunctional requirements: Agree on aspects that are visible to the user, but not directly related to functionality

Summary of requirements elicitation

Requirements elicitation

↓

Identify a problem area

Design a system

↓

Refined problem statement

Requirements analysis document

Analysis

will be covered in unit **Analysis**

↓

Requirements specification

↓

Analysis object model

Dynamic models

Entity, boundary and control objects

Generalization and specialization

Software Engineering Essentials

Requirements Elicitation

Bernd Bruegge, Stephan Krusche, Andreas Seitz, Jan Knobloch
Chair for Applied Software Engineering — Faculty of Informatics