ATTENTION TO COME/ISE 491/492 STUDENT

As a part of your requirement analysis section in your final report, you should write a requirement analysis document (RAD). The details of the document, and a sample of RAD is given below.

Please, do not forget that, in addition to RAD, your final report should include all the components mentioned in the Graduation Project Guidelines Handbook http://www.ce.dogus.edu.tr/forms/COME_ISE_GraduationProjectGuidelines_Spring2010.pdf

4.5.3 Documenting Requirements Elicitation

The results of the requirements elicitation and the analysis activities are documented in the **Requirements Analysis Document (RAD)**. This document completely describes the system in terms of functional and nonfunctional requirements. The audience for the RAD includes the client, the users, the project management, the system analysts (i.e., the developers who participate in the requirements), and the system designers (i.e., the developers who participate in the system design). The first part of the document, including use cases and nonfunctional requirements, is written during requirements elicitation. The formalization of the specification in terms of object models is written during analysis. Figure 4-16 is an example template for a RAD used in this book.

The first section of the RAD is an *Introduction*. Its purpose is to provide a brief overview of the function of the system and the reasons for its development, its scope, and references to the development context (e.g., reference to the problem statement written by the client, references to existing systems, feasibility studies). The introduction also includes the objectives and success criteria of the project.

The second section, *Current system,* describes the current state of affairs. If the new system will replace an existing system, this section describes the functionality and the problems

Requirements Analysis Document

1. Introduction
   1.1 Purpose of the system
   1.2 Scope of the system
   1.3 Objectives and success criteria of the project
   1.4 Definitions, acronyms, and abbreviations
   1.5 References
   1.6 Overview
2. Current system
3. Proposed system
   3.1 Overview
   3.2 Functional requirements
   3.3 Nonfunctional requirements
      3.3.1 Usability
      3.3.2 Reliability
      3.3.3 Performance
      3.3.4 Supportability
      3.3.5 Implementation
      3.3.6 Interface
      3.3.7 Packaging
      3.3.8 Legal
   3.4 System models
      3.4.1 Scenarios
      3.4.2 Use case model
      3.4.3 Object model
      3.4.4 Dynamic model
      3.4.5 User interface—navigational paths and screen mock-ups
4. Glossary

Figure 4-16 Outline of the Requirements Analysis Document (RAD). Sections in *italics* are completed during analysis (see next chapter).

of the current system. Otherwise, this section describes how the tasks supported by the new system are accomplished now. For example, in the case of `SatWatch`, the user currently resets her watch whenever she travels across a time zone. Because of the manual nature of this operation, the user occasionally sets the wrong time and occasionally neglects to reset. In contrast, the `SatWatch` will continually ensure accurate time within its lifetime. In the case of `FRIEND`, the current system is paper based: dispatchers keep track of resource assignments by filling out forms. Communication between dispatchers and field officers is by radio. The current system requires a high documentation and management cost that `FRIEND` aims to reduce.

The third section, *Proposed system,* documents the requirements elicitation and the analysis model of the new system. It is divided into four subsections:

- *Overview* presents a functional overview of the system.

- *Functional requirements* describes the high-level functionality of the system.
- *Nonfunctional requirements* describes user-level requirements that are not directly related to functionality. This includes usability, reliability, performance, supportability, implementation, interface, operational, packaging, and legal requirements.
- *System models* describes the scenarios, use cases, object model, and dynamic models for the system. This section contains the complete functional specification, including mock-ups illustrating the user interface of the system and navigational paths representing the sequence of screens. The subsections *Object model* and *Dynamic model* are written during the Analysis activity, described in the next chapter.

The RAD should be written after the use case model is stable, that is, when the number of modifications to the requirements is minimal. The requirements, however, are updated throughout the development process when specification problems are discovered or when the scope of the system is changed. The RAD, once published, is baselined and put under configuration management.[4] The revision history section of the RAD will provide a history of changes include the author responsible for each change, the date of the change, and a brief description of the change.

4.6 ARENA Case Study

In this section, we apply the concepts and methods described in this chapter to the ARENA system. We start with the initial problem statement provided by the client, and develop a use case model and an initial analysis object model. In previous sections, we selected examples for their illustrative value. In this section, we focus on a realistic example, describe artifacts as they are created and refined. This enables us to discuss more realistic trade-offs and design decisions and focus on operational details that are typically not visible in illustrative examples. In this discussion, "ARENA" denotes the system in general, whereas "arena" denotes a specific instantiation of the system.

4.6.1 Initial Problem Statement

After an initial meeting with the client, the problem statement is written (Figure 4-17).

Note that this brief text describes the problem and the requirements at a high level. This is not typically the stage at which we commit to a budget or a delivery date. First, we start developing the use case model by identifying actors and scenarios.

4. A **baseline** is a version of a work product that has been reviewed and formally approved. **Configuration management** is the process of tracking and approving changes to the baseline. We discuss configuration management in Chapter 13, *Configuration Management*.

ARENA Problem Statement

1. Problem

The popularity of the Internet and the World Wide Web has enabled the creation of a variety of virtual communities, groups of people sharing common interests, but who have never met each other in person. Such virtual communities can be short lived (e.g., a group of people meeting in a chat room or playing a tournament) or long lived (e.g., subscribers to a mailing list). They can include a small group of people or many thousands.

Many multi-player computer games now include support for the virtual communities that are players of the given game. Players can receive news about game upgrades, new game maps and characters; they can announce and organize matches, compare scores and exchange tips. The game company takes advantage of this infrastructure to generate revenue or to advertise its products.

Currently, however, each game company develops such community support in each individual game. Each company uses a different infrastructure, different concepts, and provides different levels of support. This redundancy and inconsistency results in many disadvantages, including a learning curve for players when joining each new community, for game companies who need to develop the support from scratch, and for advertisers who need to contact each individual community separately. Moreover, this solution does not provide much opportunity for cross-fertilization among different communities.

2. Objectives

The objectives of the ARENA project are to:

- provide an infrastructure for operating an arena, including registering new games and players, organizing tournaments, and keeping track of the players scores.
- provide a framework for league owners to customize the number and sequence of matches and the accumulation of expert rating points.
- provide a framework for game developers for developing new games, or for adapting existing games into the ARENA framework.
- provide an infrastructure for advertisers.

3. Functional requirements

ARENA supports five types of users:

- The *operator* should be able to define new games, define new tournament styles (e.g., knock-out tournaments, championships, best of series), define new expert rating formulas, and manage users.
- *League owners* should be able to define a new league, organize and announce new tournaments within a league, conduct a tournament, and declare a winner.
- *Players* should be able to register in an arena, apply for a league, play the matches that are assigned to the player, or drop out of the tournament.
- *Spectators* should be able to monitor any match in progress and check scores and statistics of past matches and players. Spectators do not need to register in an arena.
- The *advertiser* should be able to upload new advertisements, select an advertisement scheme (e.g., tournament sponsor, league sponsor), check balance due, and cancel advertisements.

Figure 4-17 Initial ARENA problem statement.

4. Nonfunctional requirements

- *Low operating cost*. The operator must be able to install and administer an arena without purchasing additional software components and without the help of a full-time system administrator.
- *Extensibility.* The operator must be able to add new games, new tournament styles, and new expert rating formulas. Such additions may require the system to be temporarily shut down and new modules (e.g., Java classes) to be added to the system. However, no modifications of the existing system should be required.
- *Scalability.* The system must support the kick-off of many parallel tournaments (e.g., 10), each involving up to 64 players and several hundreds of simultaneous spectators.
- *Low-bandwidth network*. Players should be able to play matches via a 56K analog modem or faster.

5. Target environment

- All users should be able to access any arena with a web browser supporting cookies, Javascript, and Java applets. Administration functions (e.g., adding new games, tournament styles, and users) used by the operator should not be available through the web.
- ARENA should run on any Unix operating system (e.g., MacOS X, Linux, Solaris).

Figure 4-17 *Continued.*

Ref: Object-Oriented Software Engineering Using UML, Patterns, and Java , Bruegge and Dutoit, pp 151-155, 3rd edition.