Software Requirements Specification
¡Name of System/Project¿
Version: ¡versionNo¿

¡Organization/group/person auithoring the document¿

February 21, 2014

Contents

1 Introduction

The requirements specification should ultimately contain sufficient information such that the system coudl be largely developed by a third party without further input. To this end the requirements must be precise and testable.

The requirements need not be fully specified up-front. One might start with the vision, scope and architectural requirements, perform an upfront software architecture engineering phase and then iteratively elicit the detailed requirements for a use case, build, test and deploy the use case before adding the detailed requirements for the next use case. Such an approach follows solid engineering phase for the core software infrastructure/architecture with an agile software development approach within which the application functionality is developed iteratively.

2 Vision

A description of the vision of the project. This typically includes the main purpose of the project and what the client aims to achieve with the project.

3 Background

A general discussion of what lead to the project including potentially

- business/research opportunities,
- opportunities to simplify/improve some aspect of life/work or community,
- problems your client is currently facing,
- ...

4 Architecture requirements

4.1 Access channel requirements

Specify the different access channels through which the system's services are to be accessed by humans and by other systems (e.g. Mobile/Android application clients, Restful web services clients, Browser clients, ...).

4.2 Quality requirements

Specify and quantify each of the quality requirements which are relevant to the system. Examples of quality requirements include performance, reliability, scalability, security, flexibility, maintainability, auditability/monitorability, integrability, cost, usability. Each of these quality requirements need to be either quantified or at least be specified in a testable way.

4.3 Integration requirements

This section specifies any integration requirements for any external systems. This may include

- the integration channel to be used,
- the protocols to be used,

- API specifications in the form of UML interfaces and/or technology-specific API specifications (e.g. WSDLs, CORBA IDLs, ...), and
- any quality requirements for the integration itself (performance, scalability, reliability, security, auditability, ...).

4.4 Architecture constraints

This specifies any constraints the client may specify on the system architecture include

- technologies which MUST be used,
- architectural patterns/frameworks which must be used (e.g. layering, Services Oriented Architectures, ...)
- ...

5 Functional requirements

5.1 Introduction

This section discusses the application functionality required by users (and other stakehodlers).

5.2 Scope and Limitations/Exclusions

Use a high-level use case diagram with

- abstract use cases for services/responsibility domains of the system,
- concrete use cases (the leaf use cases in the specialization hierarchy) as the required concrete use cases/user services,
- optionally some include and extend relationships to show the core functional requirements, and
- actors showing the external systems which are not part of the scope of the system, but which the system integrates with.

List the exclusions/limitations, discussing any functionality which could erroneously be assumed within scope but which has been explicitly excluded from the scope of the system.

5.3 Required functionality

Use for each concrete use case a use case diagram with the required functionality in the form of includes and extends relationships to lower level use cases – this mauy be specified across levels of granularity.

5.4 Use case prioritization

Consider a simple three-level prioritization with

Critical: A use case which is absolutely essential (ask whether the project should be canceled if that functionality could not be provided).

Important: The system would still be useful without some of the important use cases, but the client would get quantifiably less value from the system.

Nice-To-Have: Its a requirement but the value to the client/business is insignificant/not quantifiable.

5.5 Use case/Services contracts

For each use case/service specify

Pre-Conditions: the conditions under which the service may be refused (usually there is an exception associated with each pre-condition).

Post-Conditions: the conditions which must hold true after the servcie has been provided.

Request and Results Data Structures: Use class diagrams to specify the data structure requirements for the request and result objects (i.e. the inputs and outputs).

5.6 Process specifications

For some of the use cases there may be requirements around the process which needs to be followed. If so, these requirements are typically specified via activity and/or sequence diagrams or alternatively via state charts.

5.7 Domain Objects

Use UML class diagrams to specify the data structure requirements in a technology neutral way. These can ultimately be mapped onto different technologies like ERD diagrams/relational databases, XML schemas, Python/Java/C++/... objects, paper based or UI forms, ... But those are just different technology mappings and this would not be part of the requirements specification.

6 Open Issues

Discuss in this section

- any aspects of the requirements which still need to be specified,
- around which clarification is still required, as well as
- any discovered inconsistencies in the requirements.

7 Glossary

The requirements is to be read, understood and validated by a range of people from very different backgrounds (the client, domain experts/business analysts, the developers, software architects, users, ...). Use a glossary to explain any terms which some parties may not be familiar with.