Requirements Specification

Exercise Session

References

- Most SRS snippets shown are taken from
  - SRS for Project Management System
    by I. Yevgeniy, DOSE course 07
  - SRS for Tschau Sepp Logic Subcomponent
    by A. Dima, O. Clerc, A. Garcia, DOSE course 09
- You can find these two documents on the course website (Doc1, Doc3)

Quality Goals

- Justified
- Correct
- Complete
- Consistent
- Unambiguous
- Feasible
- Abstract
- Traceable
- Delimited
- Interfaced
- Readable
- Modifiable
- Testable
- Prioritized
- Endorsed

Recommended document structure

Title

- Revision History

1. Introduction

- 1.1 Purpose
- 1.2 Scope
- 1.3 Definitions, acronyms, and abbreviations
- 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

Section: Title and Revision History

- Title page
  - Title of document
  - Date, Version
  - Group members
- Revision history
  - When
  - Who
  - What

Tschau Sepp
LOGIC Sub-Component

Software Requirements Specification

- **Authors:**
  Alexandru Dima [1]
  Olivier Clerc [2]
  Alejandro García [3]
- **Document number:**
  TS-LOGIC-SRS-001
- **Total number of pages:**
  30
- **Date:**
  Tuesday 3rd November, 2009
- **Location:**
  Zürich, Switzerland

Date	Author	Version	Change Reference
2009-10-17	Alejandro García	0.1	Adding structure
2009-10-18	Olivier Clerc	0.2	Writing first draft
2009-10-18	Alexandru Dima	0.3	Writing first draft
2009-10-26	Olivier Clerc	0.4	Review and Rewriting
2009-10-26	Alexandru Dima	0.5	Adding requirements
2009-10-26	Alejandro García	0.6	Rewriting Overall description
2009-10-27	Alexandru Dima	1.0	Final version
2009-11-03	Alejandro García	1.1	Appendix

Section: Purpose

- Define the purpose of the SRS
- Specify intended audience

1.1 Purpose

This document specifies the Software Requirements Specification (SRS) for the Project Management System (PMS). It describes scope of the system, both functional and non-functional requirements for the software, design constraints and system interfaces.

1.1 Purpose

This document represents the Software Requirements Specification (SRS) for the `LOGIC` sub-component of the *Tschau Sepp Game Component*. It is designed and written for the stake holders, such as the teaching assistants, professors and developers involved in the project. Its purpose is to describe the scope, both the functional and non-functional software requirements, as well as the design constraints of the whole `LOGIC` sub-component. Furthermore, this document shows how the system's interfaces are designed in detail.

Section: Scope

- Identify software product to be produced by name (e.g. Host DBMS, Report Generator, etc.)
- Explain what the product will and will not do
- Describe application of the software: goals and benefits
- Establish relation with higher-level system requirements if any

Scope examples

1.2 Scope

The Project Management System addresses the management of software projects. It provides the framework for organizing and managing resources in such a way that these resources deliver all the work required to complete a software project within defined scope, time and cost constraints.

The system applies only to the management of software projects and is a tool that facilitates decision making; the PMS does not make decisions.

This SRS describes only required functionality of PMS, not the functionality of external systems like data storage, change management or version control systems.

This document does not divide the PMS into subsystems; it describes only requirements for the whole-system functionality which is defined in the use case model.

1.2 Scope

The *Tschau Sepp Game Component* is an implementation of the Swiss card game Tschau Sepp to be used by the overall *Multiplayer Card Games* system. For a better description of the scope of the system, the *Tschau Sepp Game Component Scope Document* should be consulted.

The scope of the `LOGIC` sub-component is to simulate a Tschau Sepp game between multiple players by maintaining the game state and by enforcing the rules of the game. Issues related to how the game is shown on the screen or how the involved computers communicate in detail via network lie outside of the scope of this sub-component.

Section: Definitions, Acronyms, Abbreviations

- Define all terms, acronyms, and abbreviations required to properly interpret the SRS.

1.3 Definitions, Acronyms and Abbreviations

The following table explains the terms and abbreviations used in the document.

Term/Abbreviation	Explanation
PMS	Project Management System
CMS	Change Management System (Bug tracking tool)
CVS	Concurrent Versions System
VSS	Microsoft Visual SourceSafe
PERT	Program Evaluation and Review Technique
GUI	Graphical User Interface
LAMP	A server that is running Linux, Apache, My-SQL and PHP
DBMS	Database Management System
DSS	Data Storage System
RBAC	Role Based Access Control

Definitions example

Term	Definition
Player	A person who can interact with the game application that has been started and is not terminated.
User	A potential player of the game.
Server	Refers to the *Multiplayer Card Games* server.
Client	Refers to the whole *Tschau Sepp Game Component* that is connected to the *Multiplayer Card Games* server.
`LOGIC`	A sub-component of the *Tschau Sepp Game Component* that is responsible for maintaining the game's logic.
`GUI`	A sub-component of the *Tschau Sepp Game Component* that is responsible for displaying all the relevant information to the player and receiving his/her actions. For this, graphical icons, text boxes and buttons are used. Furthermore, this sub-component may contain plugins, such as a chat system.
`NET`	A sub-component of the *Tschau Sepp Game Component* that is responsible for sending and receiving messages between the `NET` sub-components that are situated on the other player's computers.
Master	A mode in which the `LOGIC` sub-component can operate. In this mode it is the one who hosts the binding game state and changes it according to the received players' actions. It also informs the other `LOGIC` sub-components about the current game state.
Slave	A mode in which the `LOGIC` sub-component can operate. In this mode it merely forwards the associated player's actions that it re-

Section: product perspective

- Describe relation with other products if any.
- Examples:
  - System interfaces
  - User interfaces
  - Hardware interfaces
  - Software interfaces
  - Communications interfaces
  - Memory
  - Operations
  - Site adaptation requirements

Product perspective example 1

2.2 Product perspective

PMS it a standalone system that provides functionality described in the Product functions section. It includes all subsystems needed to fulfil these software requirements. In addition, the PMS has interfaces to the external systems, such Version Control System, Change Management and Bug Tracking System and Payroll System. These interfaces shall be implemented according to available industry standards and shall be independent from a specific external system.
Any detailed definition of an external system is out of scope of this document.
The figure 1 shows the decomposition of PMS on the functionality areas and the supported external systems.
We have to distinguish a Data Storage System (DSS) from all other external systems in that way, that Data Storage System enables normal functioning of PMS and is therefore essential. PMS stores all its data in the DSS and hence has to maintain the connection to it. PMS shall access the data storage system through standard interface (JDBC, ODBS, ADO etc). See Data storage system section for more information.

Product perspective example 2

2.1 Product perspective

The LOGIC sub-component cannot work on its own but requires both the GUI and NET sub-components. However, the LOGIC sub-component represents the central part of the all the three sub-components that make up the entire *Tschau Sepp Game Component*.

The LOGIC sub-component does not directly have an interface that connects two running LOGIC instances. Instead each LOGIC sub-component is connected to a NET sub-component that is responsible to exchange messages between computers. The LOGIC sub-component, on its own, has two interfaces: one to the GUI sub-component and another one to the NET sub-component.

Any detailed definition of the other sub-components is out of scope of this document.

Figure 1 presents an overall view of the application architecture. With this we want to present the eight different interfaces provided for the four different components that form the *Tschau Sepp Game Component*. This are named starting with the letter I (standing for interface).

There are **no** interfaces between the *Tschau Sepp Game Component* and the *Multiplayer Card Games* server.

Section: constraints

- Describe any properties that will limit the developers' options
- Examples:
  - Regulatory policies
  - Hardware limitations (e.g., signal timing requirements)
  - Interfaces to other applications
  - Parallel operation
  - Audit functions
  - Control functions
  - Higher-order language requirements
  - Reliability requirements
  - Criticality of the application
  - Safety and security considerations

Specific requirements

- This section brings requirements to a level of detail making them usable by designers and testers.
- Examples:
  - Details on external interfaces
  - Precise specification of each function
  - Responses to abnormal situations
  - Detailed performance requirements
  - Database requirements
  - Design constraints
  - Specific attributes such as reliability, availability, security, portability

Possible structures

3. Specific requirements

- 3.1 External interfaces
  - 3.1.1 User interfaces
  - 3.1.2 Hardware interfaces
  - 3.1.3 Software interfaces
  - 3.1.4 Communication interfaces
- 3.2 Functional requirements
- 3.3 Performance requirements
- 3.4 Design constraints
- 3.5 Quality requirements
- 3.6 Other requirements

3. Specific requirements

- 3.1 Functional requirements
- 3.2 Non-functional requirements

Quality Goals

- Justified
- Correct
- Complete
- Consistent
- Unambiguous
- Feasible
- Abstract
- Traceable
- Delimited
- Interfaced
- Readable
- Modifiable
- Testable
- Prioritized
- Endorsed

Functional Requirements: Bad Example

3 Specific Requirements

3.1 Functionality

3.1.1 Component

3.1.1.1 Component Priority: 3

The system shall known the concept of components which represent the actual content on a page respectively the content–pane of a page or the panes of the template of a page. A component is an instance of one of the existing components in eHTML listed in table 1. Each component refers slightly to one or arbitrary many HTML tags (table 1) however this association is only a guideline. That is the HTML tags may or may not be used to create the components in HTML.

3.1.1.2 Associating components with component-styles Priority: 3

The user shall be able to associate any component with zero or one component-styles of the corresponding component-style-type. The semantics of that association are defined in section 3.1.16.

Functional Requirements: Bad Example

3.1.1.3 Appearance Priority: 3

A component has a appearance on a HTML-site respectively in a HTML browser specified in the appearance requirement of a component. Note that there may be slightly different appearance on various HTML browsers.

Table 1: Components

COMPONENT	element	HTML tag
LINK–COMPONENT	link	...
HEADLINE–COMPONENT	headline	...
SUBHEADLINE–COMPONENT	subheadline	...
TEXTBLOCK–COMPONENT	textblock	...
IMAGE–COMPONENT	image	<img src="...">
ENUMERATION–COMPONENT	enumeration	...
	items	...
LISTING–COMPONENT	listing	...
	items	...
TABLE–COMPONENT	table	<table>...</table>
	row	<tr>...</tr>
	cell	<td>...</td>
SNIPPET–COMPONENT		-
HORIZONTAL–LINE–COMPONENT	horizontal line	<hr>

3 Specific Requirements

3.1 Functionality

3.1.1 Component

3.1.1.1 Component Priority: 3

The system shall known the concept of components, which represent the actual content on a page respectively the content–pane of a page or the panes of the template of a page. A component is an instance of one of the existing components in eHTML listed in table 1. Each component refers slightly to one or arbitrary many HTML tags (Table 1). However, this association is only a guideline. That is, the HTML tags may or may not be used to create the components in HTML.

3.1.1.2 Associating components with component-styles Priority: 3

The user shall be able to associate any component with zero or one component-styles of the corresponding component-style-type. The semantics of that association are defined in section 3.1.16.

3.1.1.3 Appearance Priority: 3

A component has a appearance on a HTML-site respectively in a HTML browser specified in the appearance requirement of a component. Note that there may be slightly different appearance on various HTML browsers.

Table 1: Components

COMPONENT	Element	HTML tag
LINK–COMPONENT	link	...
HEADLINE–COMPONENT	headline	...
SUBHEADLINE–COMPONENT	subheadline	...
TEXTBLOCK–COMPONENT	textblock	...
IMAGE–COMPONENT	image	<img src="...">
ENUMERATION–COMPONENT	enumeration	...
	items	...
LISTING–COMPONENT	listing	...
	items	...
TABLE–COMPONENT	table	<table>...</table>
	row	<tr>...</tr>
	cell	<td>...</td>
SNIPPET–COMPONENT		
HORIZONTAL–LINE–COMPONENT	horizontal line	<hr>

Functional Requirements example 1

3 Specific requirements

In the following, the `LOGIC` sub-component is referred to as the system.

Property	Description
Requirement ID	Defines a unique symbolic name for the requirement. It also reflects which functional group it belongs to.
Title	A descriptive title for the requirement.
Priority	Defines the order in which requirements should be implemented. Priorities are designated (highest to lowest) 1, 2, and 3 ... Requirements of priority 1 are mandatory for the *First Implementation*; requirements of priority 2 are mandatory for the *Final Implementation*. A priority greater or equal than 3 represents optional features.
Risk	Specifies the risk of not implementing the requirement. It shows how critical the requirement is to the system as a whole. The following risk levels are defined over the impact of not being implemented correctly. • Critical (C) It will break the main functionality of the system. The system cannot be used if this requirement is not implemented. • High (H) It will impact the main functionality of the system. Some function of the system could be inaccessible, but the system can be generally used.

Functional Requirements example 1

Req. ID	R 3.1.2.004
Title	Validate players actions
Description	If in Master mode, the system shall validate any player action that has been received, in order to enforce the rules of the game.
Priority	2
Risk	H
References	R 3.1.5.001 - R 3.1.5.012

Req. ID	R 3.1.2.005
Title	Update game state
Description	If in Master mode, the system shall change the game state if a received player action has been successfully validated, as to reflect what the action entails.
Priority	1
Risk	C
References	R 3.1.1.002, R 3.1.2.004

Req. ID	R 3.1.2.006
Title	Distribute game state
Description	If in Master mode, when the game state has been changed, the system shall inform all connected systems, which are in Slave mode, about the new game state, and thereby confirm that the action was valid.
Priority	1
Risk	C
References	R 3.1.1.004, R 3.1.3.005

Functional Requirements example 2

3 Specific Requirements

This section contains all software requirements both functional and non-functional. The functional requirements are grouped according use case model.
A requirement has the following properties:

Requirement ID	Uniquely identifies requirement within all PMS documents.
Title	Defines the functional group the requirement belongs to. Gives the requirement a symbolic name.
Description	The definition of the requirement.
Priority	Defines the order in which requirements should be implemented. Priorities are designated (highest to lowest) "1", "2", and "3" … Requirements of priority 1 must be implemented in the first productive system release. The requirements of priority 2 and lower are subject of special release-agreement, which is out of scope of this document.
Source	Refers to the raw requirement(s) from the 2 Raw PMS Requirements document. In a real-time SRS it refers to the source, what the requirement originates from.
Risk	Specifies risk of not implementing the requirement. It shows how the particular requirement is critical to the system. There are following risk's levels and associated impact to the system if the requirement is not implemented or implemented incorrectly: • *Critical (C)* – will break the main functionality of the system. The system can not be used if this requirement is not implemented. • *High (H)* – will impact the main functionality of the system. Some function of the system could be inaccessible, but the system can be generally used. • *Medium (M)* – will impact some system's features, but not the main functionality. System can be used with some limitation.

Functional Requirements example 2

Requirement ID	R1.01.04
Group	Main Functionality\User Roles\Predefined Roles
Description	The default installation of the system shall provide at least the following preconfigured user roles: "*Manager*", "*Team Leader*", "*Team Member*", "*Administrator*". The Table 3 lists the default rights of each role. The system administrator (user with the right to edit user roles) can configure permissions of the roles.
Priority	2
Source	
Risk	M
References	

User Role	**Is allowed to**
Manager	Browse project list, Create/Delete/View/Update/Export/Import project, Assign/Re-assign a resource to the project.
Team Leader	Create/Delete/View/Update task, View/Import project, Assign/Re-assign a resource to the task
Team Member	View task, View project
Administrator	Create/Delete/View/Edit user (manage user), Configure system, Create/Delete/View/Edit user role (manage user role)

Table 3. User Roles