Personal Finance Management Software

Software Requirements Specification with Use-case

Version 1.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| <08/01/17> | <1.0> | Software requirements specification with usecase | Ngo Van Quyen |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

1. Introduction 3

1.1 Purpose 3

1.2 Scope 3

1.3 Definitions, Acronyms, and Abbreviations 3

1.4 References 3

1.5 Overview 3

2. Overall Description 3

2.1 Use-Case Model Survey 3

2.2 Assumptions and Dependencies 3

3. Specific Requirements 3

3.1 Use-Case Reports 3

3.2 Supplementary Requirements 3

4. Supporting Information 3

Software Requirements Specification

# Introduction

Many software teams are discovering that mixing use-case modeling techniques for requirements expression along with traditional methods of documenting specific requirements within a “software requirements specification” (SRS) document provides an efficient means to record the complete set of detailed requirements for a system or application to be built.

In our experience, we have found that a fully specified set of use cases for a system often does a great job of stating many of the requirements for that system. But just as often there are also a significant number of requirements that do not fit well within this modeling technique. Especially for non-functional requirements (e.g., specifications for usability, reliability, performance, maintainability, supportability), it is usually best to use the good-ol’ tried-and-true traditional method for stating requirements.

## Purpose

The SRS provides an organized way to collect all software requirements surrounding a project (or perhaps at a subsystem or feature level) into one document. In these days of automation and visual modeling, we often find ourselves with several different tools for collecting these requirements. The complete set of requirements may indeed be found in several different artifacts accessible by multiple tools. For example, you might find it useful to collect some (or all) of the functional requirements of your system in use cases and you might find it handy to use a tool appropriate to the needs of defining the use-case model.  On the other hand, you might find it appropriate to collect traditionally stated natural-language requirements (such as non-functional requirements, design constraints, etc.) with a word processing tool in “supplementary specifications”. And a requirements management tool must be able to access all requirements for maintaining requirement attributes and traceability.

## Scope

Arguably, some combination of the use case and traditional specification technique should be used in all projects. To facilitate this, we will collect the requirements for our SRS in a UML “package” construct that may include a single document, multiple documents, use case specifications and even the graphical use case model which describes relationships amongst the use cases.

## Definitions, Acronyms, and Abbreviations

### **Definition**

### [Software requirements specification](http://en.wikipedia.org/wiki/Software_requirements_specification)(SRS) capture system behavior as opposed to non-functional requirements specifications which define attributes as not behavior. This 'functionality' refers to services, tasks or functions the user performs using the system in question.

*In other words, Software requirements specifications describe what the software or web site should do by defining functions and high-level logic. If the user requirements are written for the requestor and not the end-user, the user requirements are combined with the Software requirements specifications.*

### **Abbreviations**

The Institute of Electrical and Electronics Engineers: IEEE

[Software requirements specification](http://en.wikipedia.org/wiki/Software_requirements_specification): SRS

## References

1. Booch, Grady et al., UML User’s Guide, Addison-Wesley-Longman, Reading, MA (1999)
2. IEEE Std 830-1993, Recommended Practice for Software Requirements Specifications, Software Engineering Standards Committee of the IEEE Computer Society, New York (1993)
3. Jacobson, Ivar, et al., Object-oriented Software Engineering—A Use Case Driven Approach, Addison-Wesley (1992).
4. Jacobson, Ivar, Grady Booch, Jim Rumbaugh, The Unified Software Development Process, Addison-Wesley-Longman (1999)
5. Kruchten, Philippe, The Rational Unified Process—An Introduction, Addison-Wesley-Longman, Reading, MA (1999)
6. Rational Unified Process, version 5.5, Rational Software Corporation, Cupertino, CA (1999)

## Overview

In [product development](http://en.wikipedia.org/wiki/New_product_development), it is important to understand the difference between the baseline functionality necessary for any system to compete in that product domain, and features that make the system different from their competitor's products.   
Some strategies have important implications for [software architecture](http://en.wikipedia.org/wiki/Software_architecture). Specifically, it is not just the Software requirements specifications of the initial release that must be supported in the architecture. The Software requirements specifications of initial products need to be explicitly taken into consideration.

# Overall Description

## Use-Case Model Survey

The use­case model is an interpretation of the SRS (Spence & Probasco, 2000). For ease of documentation,  the use­case model along with the supplementary specifications document is used  as  the  formal documentation for the project at times. This may seem like an efficient system but   it   cannot   be   substituted   for  a   formal  SRS.   The   need   for  an   SRS   document   is   usually mandated  by  the  management.  Under such circumstances, when an SRS standards document is unavailable, the use­case  model  is  dissected and the use case descriptions cannibalized in an attempt  to  populate  the  SRS.  This  process  tends to be ad­hoc giving rise to inconsistencies in the final  document.  It  also  surfaces  traceability  issues between the use­case model and sections of the  SRS  document.  Changes  in  functional requirements in the specification document need to be reflected in the use­case model and vice­versa. We should also point out that the use­case model is  an  abstraction  of  the  system  model.  It does not capture all the relevant aspects of the system, especially   non­functional   requirements,   which are   required   for   completing   the   product documentation. An unstructured process for using use­cases to populate an SRS is inefficient and  lacks  traceability.  The  SRS  forms  the  basis  for  testing  plans  at a later stage, further boosting its importance in software development process. There   is   an   incentive   to   prepare   the   SRS   in   accordance   to   the   standards.

It   ensures readability  of  the  document  by  other  stakeholders  who  come  on  board  at  a  later  date.  The IEEE Std.830   is   understood   across   organizations   facilitating   communication   between   disparate organizations. It also makes sense from Information systems maintenance or systems testing perspective,   where   convention   is   preferred   over   unique   formats   unless   extra­ordinary circumstances exist. At the same time there is also an incentive to avoid duplication  effort.

## Assumptions and Dependencies

Assumptions and dependencies  come from several places in the use case­based  specification  process.  Some  assumptions are stated in the preconditions of the functional  use  cases,  particularly  when  the  preconditions  refer to things external to the system, whether they are actors or external systems.

# Specific Requirements

Section 3 of the SRS document contains the heart of the specification of exactly what the system  should  do  and how.  Section 3  suggests  that  this  is  the  appropriate  place  for  inclusion  of  a  higher  level  of  detail.  Therefore, essential and business use cases are not appropriate for translation into  Section  3,  only  real system use cases are.

## Use-Case Reports

From the use case description, most of the IEEE 830 items may be extracted, and restated in the following corresponding subsections:

a) Name of item: “Cashier scans barcode on product box.”

b) Description of purpose: “System displays item description and current price on point of sale terminal.” c) Source of input: Cashier (actor name)

d) Valid range, accuracy, and/or tolerance: as stated in preconditions

e) Units of measure: as stated in use case summary, or in the scenario narrative.

f) Timing: shown by sequence of steps in the use case scenario narrative.

g) Relationship to other inputs/outputs: The most relevant related inputs/outputs will be those  that  are  also  involved in the interactions within the same use case. Others may be separately noted.

h) Screen formats/organization: If required, these should be noted as system responses in the  use  case  scenario  narratives where appropriate.

i) Window formats/organization: If required, these should also be noted as system responses  where  appropriate.

j) Data formats: These may be noted either in actor actions or in system responses, depending  on  where  the  requirement applies. For example: “System displays customer’s zip  code  left  justified;  hyphen  to  appear  after first five digits if nine digit zip code on file.”

k) Command formats

l) End messages: These should appear as the last system response in the scenario narrative,  and/or  as  described  in the use case post­conditions.

## Organizing the specific requirements

*IEEE   830   explicitly   concedes   that   there   are   many   possible   ways   of   organizing   the requirements documentation in Section 3 of the SRS. One possible suggestion offered  is  to  organize  the  requirements  by user class, which would translate as actor in the UML use  case  model.  Another  is  to  organize  them  by feature, which would suggest the use of the proposed seven generic use­case  function  types  as a classification scheme, although a domainspecific use  case  classification  might  also  be  helpful  for more complex systems. Yet another suggestion  is  to  organize  requirements  by  functional  hierarchy,  for which a UML system level use case diagram ought to suffice. If all functional  requirements  have  been detailed in use case diagrams and descriptions, non­functional   requirements in supplementary specifications, and the use cases have been properly classified by  the various attributes  describing them,  then  the analyst will have collected and organized  most  of  the  information  necessary  to  create  the  IEEE  830 SRS document without much further manual intervention.*