MIS Review Vol. 17, No. 2, March (2012), pp. 31-57
© 2012 Department of Management Information Systems, College of Commerce
National Chengchi University & Airiti Press Inc.

Revisiting Software Requirements Specifications -- What Could We Learn

Björn Johansson[1], Tanja Rolandsson[2]
[1]*Department of Informatics, School of Economics and Management, Lund University*
[2]*The Swedish Armed Forces*

ABSTRACT: *Software requirements specifications (SRSs) are important documentations that reports results of system requirements determination (SRD) when developing software. It forms a base for subsequent activities in a system development process. In order to increase the knowledge of SRS and how such documentation could be structured we present an analysis of nine SRSs. From the analysis of similarities and differences in composition and requirements organization in the SRSs we aim at giving some advice on how a SRS could be improved and thereby supporting development of information systems better. The analysis shows that the overall structure of the SRSs either follows the IEEE (Institute of Electrical and Electronics Engineers) standard 830 with three main sections (introduction -- overview -- list of requirements), or another structure (introduction -- references -- list of requirements). However, how specific requirements then are structured and presented differ from SRS to SRS. The most frequent type of requirements is functional requirements, which is not a big surprise. However, more unpredictable is that non-functional requirements are getting less attention. One conclusion is that even though using standards might not be the only way to formulate SRSs, they are being used and serve their purposes, at least to some extent. However, it can also be concluded that the high focus on functional requirements in standards could be seen as an influential factor explaining why SRSs have such a high focus on functional requirements. The main conclusion is that future SRSs should spend more focus on non-functional requirements since these are both more difficult to describe and will probably play an even more important role when developing information systems in the future.*

KEYWORDS: *Software Requirements Specification, Requirements Engineering, Functional Requirements, Non-functional Requirements, IEEE 830.*

1. Introduction

Information systems development consists of a number of steps. The first one is the system requirements determination (SRD) step. SRD is described by Duggan and Thachenkary (2003) as the overall process of finding, analyzing, and documenting requirements, and includes several activities. It can clearly be stated that the SRD is important in the entire systems development process, since it forms a basis for subsequent activities, affects the design of the system architecture and contributes to the quality of the system (Hull, Jackson & Dick, 2005; Wiktorin, 2003). Hull et al. (2005) describe several reasons for project failure, and amongst those they state that having incomplete requirements is one major reason for failed projects. However, even if projects succeed, insufficient requirements can have several unwanted consequences, such as lower systems quality, that development take longer time, become more expensive than expected as well as unsatisfied users of the developed system (Eriksson, 2007; Jackson, 1995). In addition to the benefit of overcoming unwanted consequences, another benefit of well-formulated requirements is that they can be one of the most important reasons for project success (Hull et al., 2005; Wiktorin, 2003). Apparently, the SRD is of great value to the systems development process, and adequate requirements can be claimed being a necessity for project success.

However, once requirements have been collected, they need to be documented in such a way that they could be used successfully. This documentation usually results in a requirements specification. As part of the SRD, the software requirements specification (SRS) is an important document in the development project (Véras et al., 2010). The SRS can be a channel of communication, conveying the characteristics of the system between developers and users. It can also be part of a contract and as such be used to evaluate performance of systems. Moreover, it can be used to estimate time and cost for the development project (Daniels & Bahill, 2004; Smith, Lai & Khedri, 2007). Even if there are templates available it is not obvious how SRSs should be formulated. For example, the Institute for Electrical and Electronic Engineers (IEEE) has constructed a standard called 830 which lines up a number of attributes SRSs should have in order to be a well-formulated, understandable document (The Institute of Electrical and Electronics Engineers [IEEE], 1998a). This standard is widely mentioned in the requirements engineering literature (Eriksson, 2007; Smith et al., 2007; Wiegers, 1999; Wiktorin, 2003). However, Power (2002) claims that practitioners use a variety of methods, both stylistically and structure-wise, when documenting requirements. In addition, Smith et al. (2007) state that no universally accepted way of documenting requirements exist.

Research on requirements determination (Avison & Fitzgerald, 2006; Cysneiros & do Prado Leite, 2004) has shown that functional requirements, as one certain type of requirements, are specified more clearly and to a larger extent than nonfunctional requirements in requirements specifications. The explanation stated is that nonfunctional requirements are more difficult to identify than functional requirements. From the discussion so far the following questions, that are the focus of this article, can be formulated: (1) How are software requirements specifications structured, and how are requirements in them organized, and (2) What is the most common category of requirements in actual software requirements specifications, and (3) What could we learn from an investigation of real SRSs?

These questions lead to the purpose of the research, which is to gain knowledge on how to compose and structure SRSs in order to be able to develop "better" software in the future. This is done by identifying similarities and differences in SRS composition and requirements organization, illustrating what types of requirements are most frequent, and showing how requirements are formulated.

The next section presents a selection of literature on SRSs and requirements in order to create a theoretical background for the analysis. Method is discussed in Section 3. In Section 4 results from each analyzed SRS are presented, followed by an analysis and discussion of the results. Finally, there are some concluding remarks on what we have learnt from revisiting SRSs.

2. About specifications and requirements

Requirements specifications have been researched before. Besides the research by Power (2002), Franko and Hansson (2006) did an examination of impact from organizational rules when formulating a requirements specification and Dahlstrand, Fredborg and Leandersson (2009) suggested a framework for developing a requirements specification when using a particular systems development technique called Co-design. These last two, however, focus on how to create well-formulated SRSs. This research deals with the actual structure and composition of SRSs. In order to do so, the next section presents information necessary to build a theoretical background for the analysis.

2.1 Fundamentals of software requirements specifications

SRSs are important documents for systems development, used by different groups of people for different purposes; by customers, to know what to expect, by the software developers, to know what to build and how, by test groups, to test and evaluate the system and so forth (Hull et al., 2005; McIlroy & Stanton, 2011; Wiegers, 1999). The SRS can act as a channel of communication between developers and customers and help to ensure that the system satisfies customer needs (Adisa et al., 2010). Moreover, it creates the baseline upon which following systems development activities are based (Nicolás & Toval, 2009). It is obvious that every system has requirements, and the SRS exists to make these requirements possible to build (Daniels & Bahill, 2004; Hull et al., 2005).

It is quite clear that an SRS is one part of the overall systems requirements determination process which in its turn is part of the entire systems development process. However, this does not explain what an SRS is. Eriksson (2007) describes an SRS as a document produced when a system is built from scratch, or if there are major changes being made to an existing system. That might be useful information, but it is a very brief description. Wiktorin (2003) writes that “a requirements specification consists of several parts.” That sounds rather unclear and not of help when trying to understand the concept of requirements specifications. However, it does reveal that there needs to be more than one activity when creating the document. Another description, also rather short, is given by Duggan and Thachenkary (2003): “Requirements specification: representing the results [of the previous steps in the SRD process] in a document.” This explains where an SRS comes from, but not what information lies within an SRS.

One explanation of the SRS and its contents is given by Wieringa (1996), who states the following: “A requirements specification consists of a specification of product objectives and a specification of required product behavior.” In other words, an SRS shows the purpose of the system, and how it is supposed to behave -- its functionality, which is described by De Carvalho, Johansson and Parthasarathy (2010) in the following way: an SRS should describe the “what” of a system, not the “how.” Wiegers (1999) states that since the SRS is important for the following activities in systems development , it needs to have a detailed description of system behavior. Smith et al. (2007) use a similar definition as Wieringa; they state that the SRS should describe essential system requirements of the software and its external interfaces, such as functions, performance, constraints and quality attributes. Another similar description of the SRS is given by IEEE in standard 12207: “the systems requirements specification shall describe: functions and capabilities of the system; business, organizational and user requirements; safety, security, human-factors engineering (ergonomics), interface, operations, and maintenance requirements; design constraints and qualification requirements (The Institute of Electrical and Electronics Engineers [IEEE], 1998b).” To sum up, an SRS is a document created when a system is built or rebuilt, containing purpose and behavior of the system as well as descriptions of the system and its desired functions.

2.2 Recommended contents of requirements specifications

The IEEE (Institute of Electrical and Electronics Engineers) standard 830 called Recommended Practice for Software Requirements Specification is a standard where an outline for a requirements specification structure is given (IEEE, 1998a; Wiktorin, 2003). IEEE 830 is also mentioned recurrently in other literary works (Eriksson, 2007; Smith et al., 2007; Wiegers, 1999; Wiktorin, 2003). According to this standard, requirements specification should contain three sections; introduction, overview and list of requirements.

Wiegers (1999) presents a modified version of the IEEE standard 830, extending the list of requirements into (a) external interface requirements; (b) system features; (c) other non-functional requirements and finally (d) other requirements. Paragraphs (1), introduction and (2), overview are, with a couple of minor modifications, the same as presented by Wiktorin (2003). There are most likely a large number of other templates available as well, but since IEEE 830 has been frequently mentioned, it acts as the base for the analysis in this research.

Wiktorin (2003) as well as Wiegers (1999) suggests that requirements should be organized in different groups. Wiktorin (2003) also states that since functional requirements usually are numerous, it is necessary to have several sub-categories of functional requirements, making interpretation and understanding of requirements easier. Wiegers (1999) points out that it is necessary to give each requirement a unique identifier. The simplest way is to use a sequential number showing both type of requirement and number. Another way is hierarchical numbering, which is claimed by Wiegers (1999) to be the most common way to label requirements.

From a language point of view, requirements can be documented in different ways. Both formal methods, where mathematically formal syntax and semantics is used to describe the requirements, and informal methods, where the requirements are described in natural language exists (Smith et al., 2007). One example of a formal language is the Vienna Development Method Specification Language (VDM-SL). Figure 1 gives one example when such a language has been used to formulate requirements.

```
functions

42.0  $wf\text{-}first\text{-}S\text{-}no\text{-}sectimetimedev : RS^* \to \mathbb{B}$
  .1  $wf\text{-}first\text{-}S\text{-}no\text{-}sectimetimedev\ (s)\ \underline{\triangle}$
  .2      $s\,(1).sectime = \mathsf{nil}\ \wedge$
  .3      $s\,(1).timedev = \mathsf{nil}$
  .4  $\mathsf{pre}\ \ \mathsf{len}\ s \geq 1\ ;$
```

Figure 1 Example of VDM-SL in Use (Droschl, 2000)

The natural language approach, is arguably the most common one in contemporary SRSs (Nicolás & Toval, 2009). Wiegers (1999) seems to assume that requirements will be written in natural language, and gives a number of guidelines for how to formulate requirements, such as “state requirements in a consistent fashion” and “avoid comparative words.” Level of formality in SRSs does not only depend on what the analyst wants to write. It also depends on the complexity of the system. According to Daniels and Bahill (2004) highly complex systems require a higher level of formality.

As stated above, by categorizing requirements readability of SRSs can be improved. Requirements can have very varying characteristics, so categorizing them by dividing them into different types would be a logic way. But first, it is necessary to define what a requirement is.

Machado, Ramos and Fernandes (2005) present the following definitions of the term requirement: "(1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents; (3) a documented representation of a condition or capability as in (1) or (2) (Machado et al., 2005). "

The first and second of these definitions are applicable to requirements in SRSs, and are hereafter used in this research. The third definition appears to be what is considered as an SRS, and does therefore not apply to the term requirement.

Requirements are not all the same kind, which results in that there are many ways to classify them. Eriksson (2007) divides requirements into three broad categories, where the second category has a number of sub-categories. These different types of requirement categories are functional, non-functional (with the sub-categories usability, reliability, performance and supportability) and design restrictions. Wiktorin (2003) divides requirements into functional and non-functional ones. Avison and Fitzgerald (2006) describe a quite similar division: on one hand there are functional requirements, on the other hand non-functional requirements with a number of sub-categories. Another categorization is made by Grady (1992), who describes the FURPS+ model. FURPS is an acronym for Functionality, Usability, Reliability, Performance, Supportability (Grady, 1992), which is the same types as used by Eriksson (2007). The "+" sign was added to the model to "extend the acronym to emphasize various specific attributes (Grady, 1992)." Yet another division is presented in the IEEE standard 830 (IEEE, 1998a). This standard suggests the following groups of requirements: external interfaces, functions, performance requirements, logical database requirements, design constraints and software system attributes. The last category has five sub-categories; reliability, availability, security, maintainability and portability (IEEE, 1998a).

This research will henceforth use the categorization from Eriksson (2007) to describe different types of requirements. One modification has been done though; the division Eriksson (2007) makes on supportability into maintainability and testability is not used as shown in Figure 2.

To summarize this section, we started out by explaining what an SRS is, stating that it is a documentation of requirements when developing an information system or making major changes to a system. The SRS describes the purpose and functionality of the system. After this, we stated that an SRS needs to be structured in some way to simplify

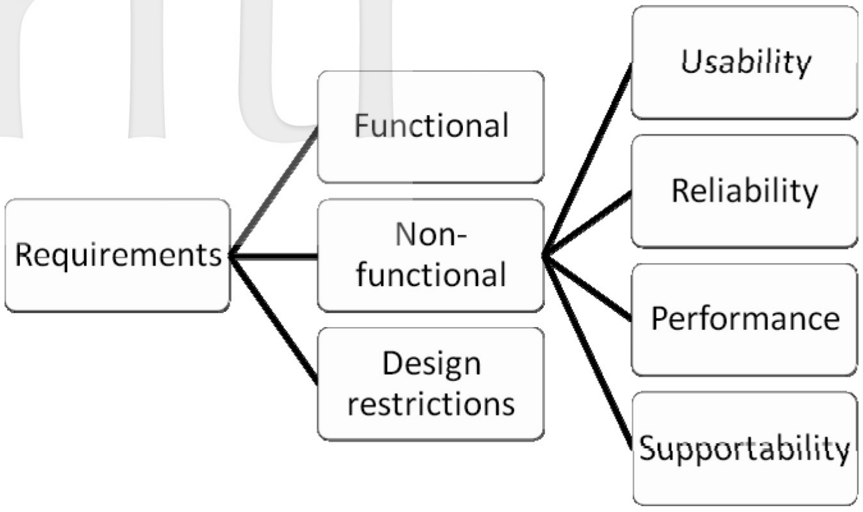

Figure 2 Hierarchy of Requirements Adopted from Eriksson (2007), Sub-categories of Supportability Deleted

the readability, and the IEEE standard 830 was presented as an example, along with an extended version, as presented in Figure 2.

3. Methodological considerations

The data for the investigation of SRSs consists of a collection of real case requirements specifications available online. Selected requirements specifications were found by using a regular Google search; however, the selection followed a rigorous plan as described below. Selected requirements specifications are from software products of different types. They were stated as being the final requirements specification for its specific product. The search was made with the keywords “software requirements specification” and “system requirements specification.” However, these terms are not very specific, resulting therefore in many hits. Instead of limiting the search to get a manageable number of hits, the first 25 pages with hits were skimmed and sites which seemed relevant were opened. This resulted in a list of 16 possible candidates. From this list 10 SRSs were selected. The six not selected turned out to be either school projects, too old, or not requirements specifications for software, following the idea of a purposeful approach (Maxwell, 2005) finding a representative sample for the research, and ensuring that the theoretical background is applicable (Eriksson, 2007; Maxwell, 2005). Choosing data available on Internet can be risky and to get reliable data it is necessary to be critical to the data when collecting it. One of the major principles in source criticism is to understand *who* the author of something is and the origin of the data (Svenning, 2003).

To avoid getting unreliable data, only requirements specifications which measured up to a certain standard were collected. The criteria we used were the following three: the SRS needed an origin, it had to be authentic (i.e., no mock-ups), and it had to be written in the last decade (year 2000 or later).

Each requirements specification needed to have a traceable origin, or at least a possibility to find out who the owner was or who had created it. To further establish the reliability and to ensure that it was okay to use the specification, an e-mail was sent to the owner of the document.

The second criterion was related to authenticity. The requirements specifications had to be from software development projects, no school projects were collected. It was easy to identify school projects; they usually had course name, student name and teacher name stated on the first page. The final criterion, that they needed to be written no earlier than year 2000, was easy to check since almost all SRSs found had a date and version number on them. Undated documents were rejected. The result of this process was that the final data set consisted of nine SRSs.

Analysis is generally based on finding patterns and a keyword in qualitative analysis is sorting. In order to be able to analyze it, collected data needs to be sorted. This is of great importance when analyzing texts (Halvorsen, 1992; Svenning, 2003). Another critical point Svenning (2003) makes regarding analyzing texts, is the need for a theoretical basis.

In this research two research questions were raised. The first was to examine how specific SRSs were structured and in what way specific requirements are organized. The second was to see if some type of requirements is more common than other types, and is most common. To answer the first question properly, it is necessary to go back to the fundamentals of SRSs. Then the questions are if the SRSs follow the IEEE 830, or not? If not, how does the structure differ from IEEE 830? Is the language formal or informal? The third paragraph in IEEE 830, list of requirements, can be organized in different ways. This is the most interesting part of question number one. Are requirements categorized by type, importance, or some other way? Are they given unique identifiers?

This leads to question number two. According to Avison and Fitzgerald (2006) as well as Cysneiros and do Prado Leite (2004), functional requirements are more common than non-functional requirements. It could be qustioned if this is consistent with reality. To say something about that, the list of requirements in each SRS was analyzed, different types of requirements identified and then analyzed from the statement that certain types of requirements are more common than others. In order to answer these questions, the outline for analysis presented in Table 1 was constructed.

Table 1 Outline for Analysis and Presentation of Results

Label of SRS <Abbreviated Name>	**Categorization of Requirements: <Description of Categorization> Section 5.3**		
	Structure of SRS: <IEEE 830/Other> Section 5.1	Language: <Natural/ Constructed>/ Identification of Requirements: <Type of Identifier> Section 5.2	Number of Different Categories of Requirements: <Functional: Usability: Reliability: Performance: Supportability: Design Restrictions: > Section 5.4

4. Presentation of the results

This section presents the results from respectively SRS one by one, in alphabetical order by the name of the system the SRS is related to. Each system is briefly presented and some examples of different types of requirements are presented. The results of the questions in the outline for analysis (Table 1) of the SRS are shown in Table 2, which is a summary of each SRS. The results organized by each question in the table are then analyzed and discussed in the section that follows.

4.1 APAF

APAF stands for ASPERA-3 Processing and Archiving Facility. ASPERA-3 is an instrument package for a space mission to Mars and the APAF is a system which is designed to process the telemetry collected by ASPERA-3. The software receives data from ASPERA-3, processes it, distributes it, presents it on a web-display and finally submits it for storage. The SRS, which is version 1.0 is from 2007 and to some extent follows the IEEE 830 standard. What is in Section two -- overview in IEEE 830 -- is here included in the first section. The second section is called “requirements specification description” with different requirements for requirements. Section three is the list of requirements, as in IEEE 830. Following is one section called “notes.” Some examples of functional requirements are: “*APAF-FR-02 The APAF system shall process all ASPERA-3 science data into IDFS data sets*,” and “*APAF-FR-07 Web-based displays of the most current ASPERA-3 data shall be provided for public view*,” and an example of supportability requirement is: “*APAF-LR-02SwRI shall provide software support for the APAF system*.”

4.2 CTBTO_WMO_SEA

CTBTO_WMO_SEA (Comprehensive Nuclear Test-Ban Treaty Organization World Meteorological Organization Special Event Analysis) is a software package designed

Table 2 Presentation of the Analysis of the Nine SRSs

SRS Name	Categorization of Requirements		
	Structure of SRS	Language / Identification of Requirements	Number of Different Categories of Requirements
APAF	(1) Capability/Functional Requirements; (2) External Interface Requirements; (3) Internal Interface Requirements; (4) Internal Data Requirements; (5) Security & Privacy Requirements; (6) Computer Resource Requirements; (7) Logistics-Related Requirements; (8) Delivery Requirements; (9) Other Requirements Considered.		
	(1) Scope; (2) Requirements Specification Descriptions; (3) Requirements; (4) Notes.	Natural/Letter-Number Combinations	Functional: 28 Usability: 0 Reliability: 0 Performance: 0 Supportability: 2 Design Restrictions: 4
CTBTO_WMO_SEA	(1) Functional Requirements; (2) Acceptance Testing Requirements; (3) Documentation Requirements; (4) Security Requirements; (5) Portability Requirements; (6) Performance Requirements.		
	(1) Scope; (2) References; (3) Functional Requirements; (4) Acceptance Testing Requirements; (5) Documentation Requirements; (6) Security Requirements; (7) Portability Requirements; (8) Performance Requirements; (9) Terminology (Glossary, Abbreviations and Appendices).	Natural/Hierarchical Numbers	Functional: 92 Usability: 6 Reliability: 0 Performance: 5 Supportability: 3 Design Restrictions: 2

Table 2 Presentation of the Analysis of the Nine SRSs (continued)

SRS Name	Categorization of Requirements		
EVLA CB	(1) External Interface Requirements; (2) Performance Requirements; (3) Reliability/Availability; (4) Serviceability; (5) Maintainability; (6) Scalability, Security; (7) Installation & Upgrades; (8) Documentation.		
	IEEE 830: (1) Introduction; (2) Overall Description; (3) Specific Requirements.	Natural/Hierarchical Numbers	Functional: 66 Usability: 2 Reliability: 8 Performance: 12 Supportability: 13 Design Restrictions: 0
I-15 RLCS	(1) External Interface Requirements; (2) Functional Requirements; (3) Performance; (4) Logical Database Requirements; (5) Design Constraints; (6) RLCS Application Software Attributes.		
	IEEE 830: (1) Introduction; (2) Overall Description; (3) Specific Requirements.	Natural/Hierarchical Numbers	Functional: 85 Usability: 0 Reliability: 10 Performance: 20 Supportability: 3 Design Restrictions: 6
MDOT VII DUAP	(1) Input Services; (2) Administrative Services; (3) Dynamic Data Services; (4) Computational Services; (5) Persistent Data Services; (6) Output Services; (7) Presentation Services; (8) Design Constraints; (9) Quality Characteristics; (10) External Services.		
	IEEE 830: (1) Introduction; (2) Overall Description; (3) Specific Requirements.	Natural/Letter-Number Combinations & Level of Priority (Low, Medium, High)	Functional: 92 Usability: 0 Reliability: 0 Performance: 0 Supportability: 4 Design Restrictions: 6

Table 2 Presentation of the Analysis of the Nine SRSs (continued)

SRS Name	Categorization of Requirements		
NPOESS DE	(1) Required States and Modes; (2) Capability Requirements; (3) External Interface Requirements; (4) Internal Interface Requirements; (5) Internal Data Requirements; (6) Adaption Requirements; (7) Security and Privacy Requirements; (8) Computer Resource Requirements; (9) Operator-Related Requirements; (10) Other Requirements.		
	(1) Scope; (2) Referenced Documents; (3) Requirements; (4) Requirements Traceability (Appendices).	Natural/Hierarchical Numbers	Functional: 88 Usability: 0 Reliability: 5 Performance: 13 Supportability: 3 Design Restrictions: 4
OSSAFFCM	(1) Required States and Modes; (2) CSCI Capability Requirements; (3) CSCI External Interface Requirements; (4) CSCI Internal Interface Requirements; (5) CSCI Internal Data Requirements; (6) Adaption Requirements; (7) Safety Requirements; (8) Security and Privacy Requirements; (9) CSCI Environment Requirements; (10) Computer Resource Requirements; (11) Software Quality Factors; (12) Design and Implementation Constraints; (13) Personnel Requirements; (14) Training-Related Requirements; (15) Logistics-Related Requirements; (16) Other Requirements; (17) Packaging Requirements; (18) Precedence and Criticality Requirements.		
	(1) Scope; (2) Referenced Documents; (3) Requirements; (4) Qualification Provisions; (5) Requirements Traceability; (6) Notes.	Natural/Hierarchical Numbers	Functional: 19 Usability: 7 Reliability: 1 Performance: 0 Supportability: 5 Design Restrictions: 10

Table 2 Presentation of the Analysis of the Nine SRSs (continued)

SRS Name	Categorization of Requirements		
SRS2XE SAMPLE	(1) Functional Requirements; (2) Acceptance Testing Requirements; (3) Documentation Requirements; (4) Security Requirements; (5) Portability Requirements; (6) Performance Requirements.		
FLAG	(1) Scope; (2) References; (3) Functional Requirements; (4) Acceptance Testing Requirements; (5) Documentation Requirements; (6) Security Requirements; (7) Portability Requirements; (8) Performance Requirements; (9) Terminology (Glossary, Abbreviations and Appendices).	Natural/Hierarchical Numbers	Functional: 49 Usability: 6 Reliability: 0 Performance: 3 Supportability: 4 Design Restrictions: 3
STEWARDS	(1) System Features; (2) External Interface Requirements; (3) Other Non-functional Requirements.		
	IEEE 830: (1) Introduction; (2) Overall Description; (3) Specific Requirements.	Natural/Letter-Number Combinations	Functional: 85 Usability: 1 Reliability: 1 Performance: 2 Supportability: 2 Design Restrictions: 9

to (1) initialize and forward calculations from a larger system and (2) provide a web-based interface for collecting, comparing, source locating and reporting data received from meteorological centers. The SRS with version no. 1.0 is from 2009, and follow an organization-specific template. Section one, introduction, is quite similar to IEEE 830. Section two is not an overview of the system, but a list of references to related documents. Requirements are not collected under one section, instead different types of requirements have their own headings. Examples of functional requirements are: "*1.3.2 The software shall allow for post processing of the virtual RN station measurements*," "*2.2.2 The software shall monitor the automated collection of the data received in response to the request*," and "*2.2.4 The web tool shall be capable to generate WMO Centre comparison statistics: Consolidation of auto-reporting on the web page of the exercise, including plot generation, and integration of the standard display of MMFORs inter-comparison statistics*." One example of a performance requirement is: "*18.4 A full daily update of the reporting web page shall not take longer than 12 hours per 10 x 10 SRS data request examined*." And an example of design restriction is: "*3.1 The CTBTO_WMO_SEA software shall run under UNIX/LINUX while making use of the native C and FORTRAN compiler package gcc running at the PTS*."

4.3 EVLA CB

This SRS describes requirements for a system which is an in-between system and the main component of a data processing pipeline in a system called EVLA (the VLA Expansion Project). The CB stands for Correlator Backend. The system is supposed to receive, assemble, format, process and finally deliver data in a suitable way. Data is sent to EVLA CB from a monitor and control system, and after processing it the EVLA CB delivers it to an end-to-end system. In the SRS it is stated that it follows IEEE 830. The SRS with the version number 2.0 is from 2002. The following examples of requirements is typical examples of functional requirements: "*3.2.1.1 Monitor and Control System -- The BE shall acknowledge receipt of all data received from M&C*," "*3.2.2.7 Data Invalid -- The BE shall replace all invalid data with zero values*," "*3.2.2.31 Reboot network -- The BE shall be able to initiate a reboot of any internal network*," and a typical example of performance requirement is: "*3.3.2.1 Input -- The BE System shall be capable of accepting an aggregate data input stream from the Correlator of a minimum om 1.6 Gbytes/sec. This must be done simultaneously with the output stream, but not necessarily over the same interconnects. This is an initial deployment specification and will be increased over time*," an example of supportability requirement is: "*3.6.1 Software tools -- Software tools and pre-built applications that do not have source code available shall come with a complete diagnostic package and customer support*."

4.4 I-15 RLCS

The Interstate 15 Reversible Lane Control System (I-15 RLCS) is designed to control opening and closing of reversible lanes on the Interstate 15. It was developed since the previous system was getting old and was impossible to extend. I-15 RLCS also provides a graphical user interface, process control and monitoring, sequencing, data processing and security as well as reporting. The SRS is from 2004; however, it does not contain any information about versions. Examples of functional requirements are: "*3.2.2.1 The RCLS shall monitor all field device sensors, and shall process operator requests for changing field device status*," and "*3.2.2.6.1 The RCLS software shall initialize each control unit and device sensor as it is identified*." Examples of performance requirements are: "*3.3.1.1 The external server data store containing RLCS status for use by external systems shall be updated once per minute*," and "*3.3.4.1 At a minimum of every 60 seconds, the system shall check the current date and time against a list of scheduled events for the current mode to determine if any event should be executed*." Examples of design restrictions are: "*3.5.1.1 The data processing and security, and reporting functions of the RLCS application software shall be implemented with commercial off-the-shelf software*," and "*3.5.2.3 The MD5 algorithm shall be used to secure application data and software in the controllers and the application servers*."

4.5 MDOT VII DUAP

MDOT VII DUAP is an abbreviation of Michigan Department of Transportation's Vehicle Infrastructure Integration Data Use Analysis and Processing system. The system is a research program, designed to examine the impact on traffic operations, asset management and transportations planning by new VII data. DUAP is supposed to collect, convert and communicate data to different people to support MDOT. The SRS investigated is version 1.02 from 2007. Examples of functional requirements are: "*IS-010 The System shall collect probe vehicle data. H*," "*IS-070-003 The DUAP data elements shall include roadway event information data fields corresponding to the SAE J2354 structure as enumerated un APPENDIX F -- SAE J2354 Event Information Elements. H*," and "*AS-140 The DUAP System shall be able to organize the sequence of execution of computational modes*." One example of supportability requirement is: "*The DUAP System shall be capable of adding new data sources. L*," and examples of design constraints are: "*DC- 010-010 The DUAP System shall use a Java software foundation. M*," and "*DC-040 The DUAP System shall use Michigan Geographic Framework geo-references. H*." Most of the requirements in this SRS also have a source (another document) and some of the requirements were commented.

4.6 NPOESS DE

The NPOESS (National Polar-Orbiting Operational Environmental Satellite System) Data Exploitation (DE) is a system that aims at distributing data from NPOESS observations to civilian customers and operational and climate communities. It receives data, process and packages and finally delivers the data to the right people. It is also supposed to be part of customer service. The SRS is from 2007 and marked with version 1.0. Sections one and three are similar to IEEE 830. Section two, which in IEEE 830 is called overview, is sort of included in the first section and instead Section two is called referenced documents. After Section three which lists requirements, there is a fourth section called requirements traceability. Examples of functional requirements are: "*3.2.1 The System shall be capable of defining Data Products for Ingest*," and "*3.3.1 The System shall be capable of receiving data and products from IDPS*." One example of performance requirement is: "*3.4.3 The system shall be capable of executing 99 % of its scheduled tasks in any consecutive 30 day period*" and a example of supportability requirement is: "*3.6.2 The System shall be capable of adding additional capacity without redesign of its infrastructure*." Example of design restriction: "*3.8.2.2 The System shall be constructed using COTS and Open Source software where it is possible, practical and approved by the Government*."

4.7 OSSAFFCM

OSSAFFCM is an abbreviation of Open Source Sustainability Assessment Framework Format Converter Module. This software is, as the name reveals, a format converter module in an open source framework. It is designed to calculate sustainability for different products and thus see environmental impact of the product. The purpose of the format converter is to convert data formats from one of four types into another without any loss of data. The SRS has version no. 1.1 and is from 2007. What is included in IEEE 830 Section two, "overview," is included in the first section. The second section contains a list of related documents. Section three lists requirements, similar to IEEE 830. After that follows three sections, called "qualification provisions," "requirements traceability" and "notes." Examples of functional requirements are: "*3.2.1.6 The inventory calculation will be able to cope with loops in the product system*," and "*3.2.4 The converter will allow conversion between important LCI (Life Cycle Inventory) data formats*." Examples of usability requirements are: "*3.12 The software will be documented in English*," and "*3.13.3 The software will be designed to be used by people interested in Life Cycle Assessment and sustainability assessment*." Example of design restriction is: "*3.9 The Framework and the converter will be designed to run in Windows environments (Win 2000, Win XP, Win Vista), Macintosh, and Linux OS. A Java Virtual Machine (JVM) of version 5 or higher needs to be installed. A MySQL database needs to be installed as well. Both JVM and MySQL will be made available on the website*."

4.8 SRS2XESAMPLEFLAG

The SRS (Source-Receptor Sensitivity) 2XESAMPLEFLAG is an extension to a web-based metrological analysis tool. When a known emitter releases xenon which might affect a xenon sample, the software is designed to flag this sample. SRS2XESAMPLEFLAG is supposed to run on any Unix-based system. The SRS from 2009 has version no. 1.0. The structure follows an organization-specific template (Same as CTBTO_WMO_SEA). The introduction, Section one, is fairly similar to IEEE 830. Section two is not an overview of the system, but a list of references. The requirements are not collected under one section, instead different types of requirements have their own headings. Examples of functional requirements are: "*1.6 The software shall be capable to choose between those available ATM (Atmospheric Transport Modelling) models that were utilized to generate the SRS (Source Receptor Sensitivity) data pertaining to the sample. Further requirements are specified in requirements 5.x*," and "*3.1 The software shall be capable to parse the information of the emissions from known xenon sources via a xenon sources inventory input file, parsed by the command line option -e<xenon-emissions-file>*." Examples of usability requirements are: "*14. All documentation will be written in English in MS Word format*," and "*20. There shall be a help function implemented that is called with the '–help' option and upon any users' choice of options that is non-compliant with the required syntax*." Example of a performance requirement is: "*25. Processing of a 21 days SRS field and preparation of the data shall not last longer than 5 minutes per sample if the full 21 days release (emission) period is examined*."

4.9 STEWARDS

STEWARDS (Sustaining the Earth's Watersheds -- Agricultural Research Data System) is a system designed to manage data related to water, soil, management and economics. It was developed to be part of the Conservation Effects Assessment Project (CEAP) which assesses the environmental effects of a conservation program implementation. The SRS with version no. 1.1 is from 2006. Examples of functional requirements are: "*FR-2.3: Browse, query, and download individual sampling station data and metadata. Provide access to the data via browsing of sites, stations, and instruments; allow for simple queries to individual datasets; provide a metadata search tool to query dataset parameters; and allow for downloading of datasets (full or partial)*," and "*FR-2.5: Generate tabular reports of selected data. Provide access to CEAP-related reports, tables and project documents*." Example of a performance requirement is: "*PR-2: Loading speed: The data system shall load as quickly as comparable productivity tools on whatever environment it is running on*," and a example of supportability requirement is: "*SQ-1: Portability: This database will be built for a particular system and may not be portable but results to queries will be portable between many environments*." Example of design restriction: "*SR-5: Relational Database Management System -- As the primary data storage mechanism for the corporate standard relational database management system, Microsoft SQL Server will be required to support system functionality*."

5. Analysis and discussion

In this section we discuss the above presented results. It follows the order of the presented questions in the outline for analysis as presented in Section two.

5.1 The structure of the different SRSs

Four of nine SRSs followed IEEE 830 -- EVLA CB, I15 RCLS, MDOT VII DUAP and STEWARDS, and except from MDOT VII DUAP, this was also stated in the SRS. Of the remaining five, four of the SRSs -- CTBTO_WMO_SEA, NPOESS DE, OSSAFFCM and SRS2XESAMPLEFLAG -- were structured in a fairly similar way. These four all started with an introductory section called scope, continued with references or referenced documents in Section two, and the requirements began in Section three. In NPOESS DE it was stated that it was based on the IEEE standard 12207. However, in that standard no given structure is described, so it must have been used in some other way than following a structure. In the SRS, it was also stated that the requirements management tool DOORS was used to create the SRS. NPOESS DE was the only SRS where it was stated that some kind of tool had been used. The last SRS -- APAF -- was structured almost as the other four in the non-IEEE 830-group, with the exception that Section two was called "requirements specification descriptions." For the SRSs who did not follow IEEE 830, including APAF, all had system overview and document overview in the first section. This is fairly similar to the sub-headings "system type" and "overview of the subsequent parts of the specification" in IEEE 830. Beyond these two subheadings, all SRSs, both the ones following IEEE 830 and the rest, had other subheadings as well in the first section. These were things like project identification, rationale or definitions.

Another similarity between all SRSs is that they had the specific requirements starting in Section three. The CTBTO_WMO_SEA and SRS2XESAMPLEFLAG turned out to have the same author, which explains why they are structured similarly. These SRSs follow an organization-specific template, called DSTD, which is stated in the document overview-section of the SRSs. In no other SRSs such information was given.

Biggest differences in the outlines of the SRSs were within the second section and how requirements were divided into groups. The second section of the SRSs was, as stated above, either "overview," as in IEEE 830, or a section with references to other documents. APAF, which stood out a bit from the others, had a second section that looked nothing like any of the other ones. It was called "requirements specification description," an explanation of necessary attributes for requirements. These attributes were for example that each requirement should have a unique identifier, be necessary for the system or formulated in a clear and concise way. It can be claimed that such statements feel a bit superfluous and belong in the requirements engineering literature and not in an SRS. Others seem to agree on that, especially since no other SRSs had such explanations of the nature of requirements.

So, except from APAF, the SRSs appeared to follow either IEEE 830, starting with an introduction, followed by a system overview and then the actual requirements, or another, unnamed structure, starting with scope, followed by a list of referenced documents, and then the actual requirements.

5.2 Language and identifier

All SRSs and requirements in them are written in an informal language, which is consistent with the claim made in chapter two, that contemporary SRSs use natural, informal language. Maybe the use of informal language also is because the systems are not extremely large or complex. Some of them were just small extensions to existing systems, for example the SRS2XESAMPLEFLAG or the OSSAFFCM. These "smaller" systems also had a fairly low number of requirements.

All requirements had unique identifiers, that in most cases were hierarchically numbered, but in APAF, MDOT VII DUAP and STEWARDS there were letter-number combinations. This is consistent with the statement both Wiegers (1999) and Eriksson (2007) makes; that hierarchical numbering is the conventional way to label requirements. No other types of labeling were present in the SRSs analyzed.

5.3 Categorization of requirements

In all cases, requirements were divided into groups depending on character of the requirements. These groups were not the same as the ones in chapter two; instead the requirements were grouped differently in each SRS, with the exception of CTBTO_WMO_SEA and SRS2XESAMPLEFLAG, who, as mentioned above, have the same outline for the entire SRS. Except for MDOT VII DUAP, one group of requirements was named almost the same in all SRSs: capability/functional requirements. Another group called external interface requirements was also very common. This group is mentioned by Wiegers (1999) as the first group of requirements in the extended IEEE 830, and appears in six of the nine SRSs. A third group of requirements was also present in six of the nine SRSs, namely security and privacy requirements. Table 2 presents how requirements were structured in the SRSs. In most SRSs, requirements were divided into 6-10 groups. It would seem logical if systems with lots of requirements had a larger number of groups;

however, this is not the case. Interesting to note is that, for example, OSSAFFCM with a fairly low number of requirements had the largest number of groups of requirements (18), while I-15 RLCS, with a great number of requirements, had a much smaller number of groups of requirements (6).

The MDOT VII DUAP varies from the others since the naming of the groups of requirements was done much differently. Most categories were called "services" of some kind, such as input services, administrative services and computational service. Also, the major part of requirements in this SRS had a reference document and some of them had comments to help the interpretation of them.

When analyzing the SRSs, it was quite difficult to identify type of requirement. Especially the non-functional ones were tricky to categorize into Eriksson's (2007) categories. None of the SRSs used the same categorization as Eriksson (2007), which made it necessary to categorize requirements from the category given in the SRS to the corresponding category in the outline for analysis. However, in most cases the name of the group of requirements corresponded to one of the FURPS+ components or one of its keywords. For example, EVLA CB had groups of requirements called "serviceability" and "maintainability," which both are FURPS+ keywords for supportability. Functional requirements were sometimes also necessary to identify. The category "security and privacy requirements" which appeared in six of the nine SRSs were usually stated under its own heading. Since "security" is one of the FURPS+ keywords for functionality and requirements usually followed a noun-verb construction such as "the System shall be capable of generating backups for all NDE data, procedures, and software (requirement 3.7.2 in NPOESS DE)," they were counted as functional requirements. However, in STEWARDS, the category "security requirements" is listed as a subcategory of "other non-functional requirements." When looking at specific requirements, there was for example one that looked like this: "*SCR-4: Availability The fourth consideration for security requirements is availability. The system must be available to the intended audience 24 hours per day, 7 days a week with, 99% availability and a tolerance of -5% (not less than 50% of working hours in any week). For this system, availability will be concerned with the reliability of the software and network components. Intentional 'denial of service attacks' is not foreseen as a significant concern* (STEWARDS, p. 13)." This could be seen as a non-functional requirement, which in the categorization belongs to the group "reliability." This example shows that the categorization of requirements can differ a lot.

To summarize this, it can be concluded that none of the SRS had the same structure of requirements as described in the theoretical background in Section two. Instead, all SRSs had individual organizations of requirements, except from CTBTO_WMO_SEA and SRS2XESAMPLEFLAG who had the same organization of requirements, probably because they have the same author. The great variety of ways to organize the requirements into groups made that a re-categorization was done to fit them in the outline for analysis and be able to compare results, especially in the counting of the requirements.

5.4 Number of different categories of requirements

Functional requirements were more frequent than non-functional ones and design requirements, which confirms statements made earlier. It also supports the statement that non-functional requirements are difficult to evaluate and measure, since they involve so many aspects. Another claim is that functional requirements are specified clearer. In some cases, the functional requirements were extremely detailed, while there were only a few non-functional requirements in the same document. This was particularly noticeable in CTBTO_WMO_SEA, where functional requirements were very detailed with a great number of sub-requirements: "*1.2.1.4.1.1.1 For the nuclide it shall be possible to enter the nuclide symbol name explicitly (e.g. Xe-133). The name will be translated into the nuclide specific half-life time (via a lookup table offered by the commission). If 'Tracer' is given, a zero half-life time shall be applied (indicated by a '999.9' in the configuration file 'SPECIES')*" while a non-functional requirement or design restriction could be as short as this: "*12 Source code will be documented in the code according to suitable standards*." However, these two requirements are, especially the latter one, rather extreme and should not be viewed as facts, only as examples or end-points of a scale on how detailed requirements can be. Most of the time, functional and non-functional requirements seemed to have been given the same level of detail.

Eriksson (2007) divided non-functional requirements into four groups; usability, reliability, performance and supportability. Of these four groups, performance requirements were most frequent. In four of the SRSs, performance requirements were labeled as an own group of requirements. This was the only group of non-functional requirements that was consistent with Eriksson's classification. As discussed above, the division of requirements into different groups in each of the SRSs seemed to follow individual templates.

Design restrictions requirements were not very common. One reason for that could be that they are formulated to a high extent in other documents. For example, in STEWARDS there is a reference to a site called "USDA Web Style Guide," MDOT VII DUAP has a reference to a document called "Systems Engineering Methodology version 1.0" and in the SRS for EVLA CB there is a reference to a document called "EVLA Correlator Monitor and Control System, Test Software and Backend Software Requirements and Design Concepts."

Regarding non-functional requirements, the most frequent were performance requirements. Maybe this is because performance requirements often are expressed with time and thereby the easiest one to measure. One example on this is the requirement 3.10.2.3 from NPOESS DE: "*The System shall deliver A-DCS telemetry data from IDPS to the US Global Processing Center within 1minute of their receipt*." Comparing this to a requirement from OSSAFFCM: "*3.11.2 The software will be easily learned and used. This usability will be supported by documentation accompanying the software*." It is clear that that is hard to measure. As Avison and Fitzgerald (2006) state, nonfunctional requirements can be hard to evaluate, especially ones that not are about time.

Wrapping this up, functional requirements were more common than non-functional ones and design restrictions. This confirms statements about functional requirements and that they are the most frequent ones in SRSs. Sometimes functional requirements were more detailed than non-functional ones, but this was only apparent in special cases and not something consistent. According to the results, it is not true that functional requirements are described clearer or in more detail than non-functional requirements or design restrictions.

6. Conclusions

The purpose of this research was to identify similarities and differences in SRS composition. From the analyses of nine SRSs it can be concluded that there seems to be two major types of structures for SRSs. They either follow IEEE 830 with its introduction -- overview -- list of requirements structure, or have an outline which is introduction -- references -- list of requirements. There was no similar pattern regarding the organization of the actual requirements. With the exception of CTBTO_WMO_SEA and SRS2XESAMPLEFLAG who have the same origin and were structured the exact same way, no other SRSs had a similar way to organize requirements. The only similarity in organization of requirements was that in all cases, requirements were divided into groups, and they had unique identifiers. Three groups of requirements were frequently used: functional/capability requirements, external interface requirements and security/privacy requirements. The differences in SRS composition are, however, not very extreme. This finding is not consistent with the results reported by Power (2002), who claims that SRS structures vary a lot.

Looking at specific requirements, functional requirements outnumbered the other categories by large. Even combined, there were less non-functional requirements than functional ones. This does confirm the claim that it might have been better to follow Wiktorin (2003) suggestions dividing functional requirements into groups. However, there were few requirements in the SRSs that clearly could be regarded as functional requirements. Since the functional requirements category thus became much broader than the different categories of non-functional requirements, it was not very surprising that there were many more functional requirements.

It can also be concluded that number of requirements is not a reason for the choice of structure of SRSs. For example, the OSSAFFCM which had the largest number of groups of requirements had a very small number of actual requirements. It can be stated that it seems unnecessary to have so many groups of requirements when in some cases there was only one or two requirements in each category, but the reason for this is not clear. A possible suggestion for further research would then be to pick out a couple of SRSs and contact the originators to get the answer on such questions.

It can be stated that following standards, like the IEEE 830, simplifies documentation of requirements. If every requirements engineer documented requirements in his/her own way, the development team would have to put a lot of time in interpreting the SRS for every new project. This is of course why standards are developed in the first place, and it can be concluded that standards are useful, as long as they are applicable to the organizational context and allow modifications. Another benefit of using standards is that they do not need to be organization-specific. If the development team works independently and is not bound to one specific company, they can use the same template for different projects in different organizations.

Regarding the specific groups of requirements, it might be possible to use templates as well. The literature showed a number of different ways to categorize requirements (Avison & Fitzgerald, 2006; Eriksson, 2007; IEEE, 1998a; Wiktorin, 2003), and the results in this research showed a great variety in the organization of specific requirements. The main conclusion from the research on SRSs is that a more general classification might be useful, particularly for independent development teams, or when development teams from different companies cooperate in projects. However, it can also be concluded that a classification should focus more on non-functional requirements and thereby decrease the risk of having a too high focus on functional requirements in future systems development processes.

Acknowledgements

We would like to express our thanks to the anonymous reviewers and Professor She-I Chang, whom acted as guest editor, for their comments and suggestions during the process of finalizing this article. This paper is a revised and expanded version of a paper entitled “Revisiting Software Requirements Specifications -- What Could We Learn” presented at

2010 International Conference on Accounting and Information Technology (ICAIT) July 5-7, 2010, Chiayi, Taiwan.

References

Adisa, F., Schubert, P., Sudzina, F. and Johansson, B. (2010) 'Living requirements space: an open access tool for enterprise resource planning systems requirements gathering', *Online Information Review*, Vol. 34, No. 4, pp. 540-564.

Avison, D. and Fitzgerald, G. (2006) *Information Systems Development: Methodologies, Techniques & Tools*, McGraw-Hill, Berkshire, UK.

Cysneiros, L.M. and do Prado Leite, J.C.S. (2004) 'Nonfunctional requirements: from elicitation to conceptual models', *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, pp. 328-350.

Dahlstrand, M., Fredborg, H. and Leandersson, S. (2009) 'Co-design av co-design -- förslag på riktlinjer för arbetssättet och utformningen av kravspecifikationer', Unpublished Bachelor's thesis, University of Borås, Västra Götaland, Sweden.

Daniels, J. and Bahill, T. (2004) 'The hybrid process that combines traditional requirements and use cases', *Systems Engineering*, Vol. 7, No. 4, pp. 303-319.

De Carvalho, R.A., Johansson, B. and Parthasarathy, S. (2010) 'Software tools for requirements management in an ERP system context', *Journal of Software Engineering and Technology*, Vol. 2, No. 2, pp. 101-106.

Droschl, G. (2000) 'Formal specification of a security system module in VDM-SL', Technical report, Institute for Software Technology, Technical University of Graz, Graz, Austria.

Duggan, E.W. and Thachenkary, C.S. (2003) 'Higher quality requirements: supporting joint application development with the nominal group technique', *Information Technology and Management*, Vol. 4, No. 4, pp. 391-408.

Eriksson, U. (2007) *Kravhantering för IT-System*, Studentlitteratur, Malmö, Sweden.

Franko, S. and Hansson, M. (2006) 'Verksamhetsregler vid utformning av kravspecifikation', Unpublished Bachelor's thesis, Lund University, Scania, Sweden.

Grady, R.B. (1992) *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, Upper Saddle River, NJ.

Halvorsen, K. (1992) *Samhällsvetenskaplig Metod*, Studentlitteratur, Lund, Sweden.

Hull, E., Jackson, K. and Dick, J. (2005) *Requirements Engineering*, Springer, London, UK.

Jackson, M. (1995) *Software Requirements & Specifications: A Lexicon of Practice, Principles and Prejudices*, ACM Press, London, UK.

Machado, R.J., Ramos, I. and Fernandes, J.M. (2005) 'Specification of requirements models', in Aurum, A. and Wohlin, C. (Eds.), *Engineering and Managing Software Requirements*, Springer, Berlin, Germany, pp. 47-68.

Maxwell, J.A. (2005) *Qualitative Research Design: An Interactive Approach*, Sage, Thousand Oaks, CA.

McIlroy, R.C. and Stanton, N.A. (2011) 'Specifying the requirements for requirements specification: the case for work domain and worker competencies analyses', *Theoretical Issues in Ergonomics Science*, doi: 10.1080/1463922X.2010.539287

Nicolás, J. and Toval, A. (2009) 'On the generation of requirements specifications from software engineering models: a systematic literature review', *Information and Software Technology*, Vol. 51, No. 9, pp. 1291-1307.

Power, N.M. (2002) 'A grounded theory of requirements documentation in the practice of software development', Unpublished Ph.D. dissertation, Dublin City University, Dublin, Ireland.

Smith, S., Lai, L. and Khedri, R. (2007) 'Requirements analysis for engineering computation: a systematic approach for improving reliability', *Reliable Computing*, Vol. 13, No. 1, pp. 83-107.

Svenning, C. (2003) *Metodboken* (5th ed.), Lorentz Förlag, Eslöv, Sweden.

The Institute of Electrical and Electronics Engineers (1998a) 'IEEE Standard 830-1998: IEEE Recommended Practice for Software Requirements Specifications', Technical report, IEEE, New York, NY.

The Institute of Electrical and Electronics Engineers (1998b) 'IEEE/EIA Standard 12207.0-1996: (ISO/IEC 12207) Standard for Information Technology -- Software Life Cycle Processes', Technical report, IEEE, New York, NY.

Véras, P.C., Villani, E., Ambrósio, A.M., Pontes, R.P., Vieira, M. and Madeira, H. (2010) 'Benchmarking software requirements documentation for space application', in Schoitsch, E. (Ed.), *Computer Safety, Reliability, and Security*, Springer, Berlin, Germany, pp. 112-125.

Wiegers, K.E. (1999) *Software Requirements*, Microsoft Press, Redmond, WA.

Wieringa, R.J. (1996) *Requirements Engineering: Frameworks for Understanding*, John Wiley & Sons, Chichester, UK.

Wiktorin, L. (2003) *Systemutveckling på 2000-Talet*, Studentlitteratur, Lund, Sweden.

About the authors

Björn Johansson holds a Ph.D. in Information Systems Development from the Department of Management and Engineering at Linköping University. Currently he works as Associate Senior Lecturer at Department of Informatics, Lund University. Previously he worked as a Post Doc. at the Center for Applied ICT at Copenhagen Business School. He is a member of the IFIP Working Groups IFIP 8.6 and IFIP 8.9.

Tanja Rolandsson received her B.Sc. in Informatics and B.A. in English Linguistics from Lund University in 2010. She is currently working for the Swedish Armed Forces as a systems engineer at the Armed Forces Command and Control Regiment with development, implementation and administration of several different weather information systems.

Appendix: SRS Sources

All websites visited January 3, 2010.

1. ASPERA-3 Processing and Archiving Facility
   http://www.aspera-3.org/idfs/APAF_SRS_V1.0.pdf

2. CTBTO_WMO_SEA
   http://www.ctbto.org/fileadmin/user_upload/procurement/2008/RFP2009-0339-CTBTO_WMO_SEA_Software_Requirements_Specification-DISCHENDORF.pdf

3. EVLA Correlator Backend
   http://www.aoc.nrao.edu/evla/techdocs/computer/workdocs/Corr_bkend_Req_Soft.pdf

4. I 15 Reversible Lane Control System
   http://www.dot.ca.gov/dist11/operations/I15RLCS/RFP_DOT2040_SecVID_Apr21_2004.pdf

5. MDOT Vehicle Infrastructure Integration Data Use Analysis and Processing (VII DUAP)
   http://www.michigan.gov/documents/mdot/MDOT_DUAP_SysReq_Final_220099_7.pdf

6. NPOESS Data Exploitation
   http://projects.osd.noaa.gov/NDE/pub-docs/SystemRequirementsDoc.pdf

7. OSSAFFCM Open Source Sustainability Assessment Framework, Format Converter Module
   http://www.openlca.org/uploads/media/Software_Requirements_v1.1_3Feb07.pdf

8. SRS2xESampleFlag
   http://www.ctbto.org/fileadmin/user_upload/procurement/2008/RFP2009-0337-SRS2XeSampleFlag_Software_Requirements_Specification-DISCHENDORF.pdf

9. STEWARDS
   ftp://ftp-fc.sc.egov.usda.gov/NHQ/nri/ceap/stewardssystemdesign030206.pdf