CCSE Computer and Information Science
www.ccsenet.org/cis
Vol. 3, No. 1
February 2010

Documenting Software Requirements Specification: A Revisit

Mingtao Shi
FOM Fachhochschule für Oekonomie & Management
University of Applied Science
Bismarckstr. 107, 10625 Berlin, Germany
Tel: 49-171-2881-169   E-mail: Consulting_Shi@yahoo.de

Abstract

Software Requirements Specification (SRS) is the key documentation, defining the functional and non-functional system requirements. By revisiting a number of in the literature much discussed key aspects related to SRS and extracting essential views from the author's daily work experience, this papers stresses the importance of the SRS and examines the process, which enables the emergence of a quality SRS. Business information must be acquired, discussed, analysed and digested, which forms the inputs for the documentation of SRS. A SRS typically consists of a set of documents, including the SRS itself, which is a written description of business requirements and system features, and analysis models.

Keywords: Requirements engineering, Software Requirements Specification (SRS), Requirements elicitation, Requirements modelling, SRS Tools

1. Software Requirements: An Overview

It is hardly contentious that requirements engineering as the functional interface between IT and the business domain has long become the initiating activity of the software development process, the importance of which cannot be overestimated (Paech & Rolland, 2008). What is equally uncontroversial is the notion contending that the success of software systems significantly depends upon the quality of the documentation in all key phases of software engineering, including requirements, design and construction, test and debugging, and release and delivery phases. This is particularly true for larger and more complex systems. Thus, meaningful requirements analysis and documentation is positively related to higher quality of software development projects. Importantly, Software Requirements Specification (SRS) is the key document in the requirements phase, laying the foundation for the realisation of functional and non-functional system requirements.

Many industrial firms today still tend to undervalue the necessity of requirements process in their daily practice, because the effects and benefits of related activities were allegedly "invisible". In fact, exactly the opposite is true. A number of authors have admirably written about the treatment of requirements in great detail (Pohl, 2007, Robertson & Robertson, 2006, Hull, Jackson & Dick, 2005). But to deal with this issue completely in compliance with theory would consume much firm-internal human resources and time capacity that is virtually non-existent, especially in small and medium-sized firms.

Therefore, this paper is intended to re-visit and streamline some of the key concepts related to the documentation of SRS from the perspective of an industry practitioner working in the area of requirements engineering, and by doing so, shows possible ways of effective and efficient SRS documentation. Efficiency is especially essential for firms with resource constraints.

2. Informational Sources of SRS

A formal SRS is normally created for more complex software system. It is often necessary to organise a launch meeting, while initiating the larger software project. It is vastly beneficial if the project members can enlist the support of the senior management and the top managers of the firm can show up in the meeting, in which analysts, designers, developers and testers can communicate with each other, responsibility areas are laid down and action plan, schedule, resources and budget are planned as precisely as possible.

SRS are based upon the needs stemming from the business domain. Rupp (2007, p. 115-134) discussed the methodologies of information elicitation extensively and differentiates among creativity, observation, questioning, history-oriented and other supplementary techniques such as Work-Shop and Mind-Mapping. Although all techniques have advantages and disadvantages, creativity techniques are generally suitable for advanced analysis and observation and questioning techniques are less time-consuming and also effective.

The information gathered in the first round does not represent meaningful or structured system requirements in most cases. Therefore, discussions afterwards within the system development team are necessary and highly recommended. To re-examine the "raw" information by using flipcharts or whiteboard in a group deepens the understanding of the nature of the elicited information and helps to restructure the available information in such a way that later documentation work will become easier and more effective. If uncertain or opaque information should arise during the group discussions, iterative interactions between the requirement engineer and the business is indispensible. This step taking place after information gathering through iteration is strikingly important to ensure the quality of the requirements analysis.

Once the required system features are seized and become intelligible to the analyst, the analysis work may begin. Business information needs to be converted into technical data, functionalities and system behaviour. Software authority Pressman (2009, p. 148-199) and Pressman (2004) recommended scenario-based, flow-based, class-based and behaviour-based techniques to analysts. A detailed delineation of these techniques is beyond the scope of this paper. However, it is worth mentioning that scenario-based modelling mainly includes use-cases and activity diagrams (swimlane diagrams). Although the requirement engineer can begin to conceive use-cases in the previous elicitation stage, individual user stories ought to be refined further and the relationship among the generated use-cases should be sorted out in this modelling stage of the requirement engineering. The use-cases themselves should become more rigour, optimally following a stringent template that includes the essential information for designer and programmer. Some authors emphasise other techniques such as Entity Relationship Diagram (ERD). Diagrammatical or graphical notations of modelling techniques are highly coveted in the daily business because of the ease of understanding, although the textual use-cases are also salient.

It is vital to point out that the modelled system is not necessarily always realistic for the software development. Negotiation between the future system users and system developers is always needed. Making efforts towards a wish-reality convergence and maintaining a friendly and motivating project environment is another challenge for the requirements analyst.

3. Creation of SRS

Firms that do carry out requirements engineering tend to only focus on models and diagrams, because the related activities are usually less time-consuming. However, modelling the user requirements is not sufficient for the software development team to design and construct a solid system. Experience shows that the requirement engineer should generally formulate a written document, the Software Requirements Specification. In fact, the models (use-cases and diagrams) build in the analysis/elaboration stage are appendices and attachments of SRS. SRS can be written while or after modelling the requirements.

A SRS must explicitly spell out what the system should precisely perform on the one hand and address the following issues on the other:

- Requirements should be labelled to establish linkages with use-cases and later test-cases. While *labelling the requirements*, it may be beneficial to do this in a hierarchical structure without using numerical structure. Numerical structure is less flexible for later structural changes. One non-numerical example would be: Banking.Retail.Accout.OpenAccount;
- *Open issues* that were not considered as user needs were elicited and elaborated and became visible in the modelling or specification stage must be registered in a list within the SRS and the business and technical implication of these issues be clarified gradually;
- Statements and notices that are not generally known to all participating parties (stakeholders) but are vital components of the SRS should be captured in a *reference system*, so that readers can clearly comprehend, from which authorised informational source certain requirements stem;
- In many cases, systems exist not only to support core businesses of the users but also their *ancillary processes*. A banking system, for example, must support not only core transactions such as standing orders and teller/till activities, but also reporting/controlling and accounting procedures. A SRS must mention and describe the needs of these ancillary processes that must be dealt with by the system and also establish corresponding requirements in additional chapters;
- Do not forget *interfaces towards external systems*. If these interfaces are based upon standards (e.g. ISO or IEEE interfaces), uncertainties exist but are not significant. On the contrary, if interfaces are proprietary, the degree of uncertainties rises exponentially. It is recommended to involve as less external interfaces as possible and standard-based interfaces only. The SRS should map the communication mode and content of these interfaces in great detail in a separated chapter;

- A successful SRS always includes a number of *important appendices*, such as use-cases, modelling diagrams, data dictionary, preliminary (or more precise) UI design, possibly test cases and test plan.

Wiegers (2003, p. 171-181) showed an effective SRS template, which can be used in the daily businesses immediately, although not necessarily all chapters must covered by the specification writers. Levels of details and areas to be covered in the specification vary from system to system. In order to reduce resources to be deployed, it is suggested that the analysts formulate the SRS as concisely as possible (KISS: Keep it Simple, Stupid) but as detailed as necessary. The capability of the analysts to write a powerful SRS grows as the experience of the analysts grows. In other words, the SRS must be seen as a routine task of the analysts, a viewpoint that is not shared by all analysts. Many of them are excel at analysing and modelling the requirements and delineating user scenarios but reluctant to write a SRS and frequently delegate this task to colleagues responsible for documentary tasks. This kind of modus operandi usually consume more but not less firm-internal resources, because of the different skills required and intensive communication necessary for complex information.

While writing a SRS, analysts also need to pay attention to the linguistic aspects. Rupp (2007, p. 140-174) intriguingly pointed out that deletion, generalisation and distortion cause substantive quality loss of an SRS and introduces a number of rules to effectively tackle this issue. A detailed discussion of this aspect is beyond the scope of this paper.

A completed SRS should be reviewed by the project team, especially by the requirements analyst. During this validation process, open issues must be clarified and the SRS updated. Furthermore, erroneous, inconsistent and conflicting statements in the SRS must be corrected. A SRS is baselined when the validation of the requirements is completed. An experienced analyst external to the project may review the SRS again to re-confirm the quality of the SRS.

4. SRS Tools

As stated, a SRS is usually not one document, but a set of documents. Firms adopting a SRS tool need to apply a multi-faceted selection strategy but should not only concentrate on one feature (SRS template) that the software may provide. There are plenty of software tools that support the creation and management of SRS. No matter whether open source (e.g.. BOUML) or licensed tools (e.g. IBM Rational Rose) are used, some key issues must be effectively dealt with. The tool should:

- enable the creation and management of textual use-cases;
- provide editable template for SRS;
- integrate diagrammatical, mathematical and logical illustration for requirements modelling (e.g. use-case diagram, activity diagram, entity-relationship-diagram, state diagram, class diagram, data dictionary);
- establish linkages among use-cases, software requirements and test-cases;
- support the management of documentation, including change orders, release and version management.

5. Conclusion

Documentation of SRS is the key activity to elicit and understand, elaborate and illustrate the business requirements and forms a crucial building block in the process of software engineering, linking business-oriented delineation with code-oriented modelling. In order to acquire the necessary information for a SRS, a requirements engineer may experience a number of procedural stages, in order to ensure quality. These stages typically include project launch, requirements elicitation, elicitation refinement, requirements modelling and negotiation. This paper, subsequently, discusses some of the most essential aspects during the documentation of a SRS and briefly looks at the tool requirements for supporting the creation and management of SRS.

References

Hull, E., Jackson, K., & Dick, J. (2005). *Requirements engineering.* (2nd ed.). Berlin: Springer Verlag.

Paech, B., & Rolland, C. (2008). REFSQ'08 international working conference on requirements engineering: Foundation for software quality. In B. Paech, & C. Rolland (Eds.), Requirements engineering: Foundation for software quality (pp. 1-5). Berlin/Heidelberg: Springer Verlag.

Pohl, K. (2007). *Requirements Engineering: Grundlagen, Prinzipien*, Techniken. Heidelberg: dpunkt.verlag GmbH.

Pressman, R. S. (2004). *Software engineering: A practitioner's approach.* (6th ed.). New York, NY: McGraw Hill.

Pressman, R. S. (2009). *Software engineering: A practitioner's approach.* (7th ed.). New York, NY: McGraw Hill.

Robertson, S., & Robertson, J. (2006). Mastering the requirements process. (2nd ed.). Westford, MA: Pearson Education Inc.

Rupp, C. (2007). Requirements-Engineering und Management: Professionelle, iterative Anforderungsanalyse für die Praxis. (4th ed.). München/Wien: Carl Hanser Verlag.

Wiegers, K. E. (2003). *Software requirements.* (2nd ed.). Redmond, Washington: Microsoft Press.