Position Papers of the Federated Conference on Computer Science and Information Systems pp. 265–269
DOI: 10.15439/2016F468
ACSIS, Vol. 9. ISSN 2300-5963

Completeness and Consistency of the System Requirement Specification

Jaroslaw Kuchta
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
Narutowicza 11/12, 80-233 Gdansk, Poland
Email: qhta@eti.pg.gda.pl

***Abstract*—Although the System Requirement Specification, as a first formal and detailed document, is the base for the software project in classic software methodologies, there is a noticeable problem of assuring the completeness of this document. The lack of its completeness causes uncertainty of the project foundations. This was one of motivations for agile methodologies – if the SRS cannot be easily validated, if it can change in late project phases, then get rid of the SRS. Replace formal requirements with *user stories*. However *user stories* are also requirements - mostly functional requirements. As agile methodologies focus on functional requirements, it is easy to forget quality requirements.**

In this paper we show the impact of quality requirements analysis on functional requirements exploration. Although in our experiment we noticed considerable large functional requirements increment, we went further and examined the impact of SRS consistency on its completeness. The research has shown that the increment of the revealed requirements count may be almost three times greater, compared to the standard requirement specification method.

Keywords: system requirements, SRS, quality, completeness, consistency,

I. Introduction

Starting the software project, the customers may be uncertain of their expectations, or may simply be unable to imagine the whole complexity of the software system. The developer's task at this very early stage of the project is to reveal as much of the requirements as is possible. In other words – to assure the completeness of the System Requirement Specification. However, there is a noticeable problem with the completeness of the SRS; with its measurement and event with its definition. The formal definition for the requirements completeness says that the SRS is complete when the "*responses of the software to all realizable classes of input data in all realizable classes of situations is included*" [1]. This definition is not very usable, as we do not know the number of "*all realizable classed of input data*" and "*all realizable classes of situations*". Other definitions [2] are also unusable. This leads us to the problem: "*how can we be sure that the requirement specification is complete without knowing what the complete requirement specification is?*" [3]. As it is an impossible situation, we can not determine the absolute completeness; we may only define some metrics of a relative completeness (as the relative increment of the elicited requirements) [4].

The whole system quality depends on the completeness of the requirement specification. We may find a huge set of quality metrics in [5]. As we take a close look at these metrics, we may see that many of them are evaluated relatively to the number of functions described in the requirement specification. As we do not know the completeness of the requirement specification, we cannot be sure about the reliability of these metrics and the total quality of the system.

The uncertainty of the SRS completeness causes the risk of the requirements change during the development process [6]. The risk is extremely high in the classic "waterfall" software development model and it is one of the reasons why the iterative and incremental models became much more popular [7]. Often, an "agile" development process is taken [8], when the requirements are revealed in the user acceptance tests of the working (but incomplete) system. Although this model goes well in the normal situations, there may be some rare and critical conditions, which are hard to reveal based solely on the customer's claims. The author's research has shown, that the problem of the SRS completeness uncertainty may be resolved indirectly – with the measurement the SRS consistency. This measurement (in a graph theory sense) is easy, and has the large impact on the number of the revealed requirements. Using this method, the author has achieved the substantial increment of the requirements (almost three times) compared to the standard method of the requirement specification.

The term "*consistency*" has somehow fuzzy meanings in software engineering: we may understand it as *a lack of contradiction between two, or more, requirements* [9][10]; and also as *traceability*, which means the *ability to trace the requirements to the software solutions* [11][12]. In this paper, we understand consistency as a *degree of coupling* between the requirements. To measure the degree of the internal coupling of the System Requirements Specification, it must be treated not only as a text document, but also in some quasi-graph form; where the vertices represent the requirements, and the edges represent the references between the requirements. The references between the requirements should be created when one requirement *impedes* the other (e.g. the "data backup" requirement impedes the "data restore" requirement). These are called "trace" references. They should also be created when two requirements *complement* each other (e.g. "logging in" complements "logging off"). Other logical references between requirements should be added "manually" during the requirements specification.

There may be several types of requirements: functional requirements, data requirements, and quality requirements. Some of the quality requirements touch the reliability of the system. System reliability depends on the resolution if critical situations which impede the functional requirements. Detection of unresolved critical situations (i.e. not resolved with functional requirements) leads to the assumption that some functional requirements may be missing. However, this assumption must be confirmed (or denied) in the detailed analysis of the SRS.

II. Measuring the SRS Completeness and Consistency

To evaluate the SRS quality we need a set of precise and objective metrics. Traditionally, when an SRS document is text written, we may count some specific weak phrases (as "adequate", "not limited to") [13]. Having the SRS document stored in the quasi-graph form, we defined other metrics of the SRS completeness and consistency.

A. Metrics and Measures

We distinguished metrics and measures in quality measurement. We treat a *metric* as a quality factor to be measured, and a *measure* as a method of the measurement. We divided measures between direct or indirect measures. We counted *direct measures* directly in the document quasi-graph storage; and we calculated *indirect measures* using some formulas. Every direct measure results in some absolute number counted directly by simple graph analysis (e.g. the number of "trace" references). These we called objective measures, as they may be objectively evaluated. Objective measures, although easy to evaluate, are insufficient to express a complete document quality. Some human analysis is needed to reveal the missing requirements, or to find inconsistent ones. The values resulting from this analysis are called expert measures, as the document review must be done by and expert. The review of the SRS should contain a list of missing requirements and a list of quality remarks to already defined requirements. As expert measures are less reliable than objective ones, the usage of expert measures is minimized. Two expert measures are used in this paper: one for completeness and one for consistency. Indirect measure expression usually divides two direct measure results. The result value is scaled from zero to one. Zero means the worst result and One means the best (ideal) result. When the numerator measure gives the number of "negative" elements (e.g. missing elements), then the function for an indirect measure is defined by a formula:

$$F(m,n) = Floor(1 - m/n, 2)$$

where the Floor function rounds the first argument towards zero (with the precision of two decimal digits); m and n are direct measures. Rounding towards zero is needed as the result value of 1.0 can only appear in the ideal situation (e.g. no missing elements).

If one metric is evaluated with several measures, some aggregate function is needed, e.g. a weighted mean function:

$$Avg(m_1, m_2, ...m_n) = \frac{w_1 m_1 + w_2 m_2 + ... + w_n m_n}{\sum_i w_i}$$

where $m_1, m_2, ...m_n$ are component measures, and $w_1, w_2, ...w_n$ are their weights respectively.

Some other terms used in metrics and measures definitions (below) need explanation. These terms are: element, meta-class, relationship, reference and solution. An *element* means an item of the software project (e.g. the system requirement). Each element has its *meta-class*, which describes its possible and required properties and relationships (e.g. FunctionalRequirement is one of the meta-classes). *Relationships* are not only *references*, which are meaningful only in the development process, but also associations meaningful in the runtime. Solution means here an element, or a set of elements, defined with a "trace" reference as the elaboration of some element.

B. SRS Completeness Metrics and Measures

Proposed metrics of the SRS Completeness (CP) are: Formal Completeness (FCP), Semantic Completeness (SCP) and Reference Completeness (RCP) – see fig.1. *Formal Completeness* (FCP) involves *Template Completeness Factor* (TCPF) and *Definition Completeness Factor* (DCPF). First factor (TCPF) tells us, how completely a template of the document is filled. We count the number of elements required by a document template (the number of required meta-classes), and search missing ones. Second factor (DCPF) represents the number of elements with incomplete definition (with one, or more, properties required by meta-classes missing).

Semantic Completeness (SCP) uses an expert measure – *Missing Semantic Element Count* (MSEC). The missing elements must be listed by an expert revising the document. To get the value scaled from Zero to One we must divide MSEC not only by the *Total Semantic Element Count* (TSEC), but by the sum of TSEC and MSEC.

Reference Completeness (RCP) depends on two measures. First; a *solution completeness factor* (SLCF), evaluates the "trace" references leading to the "solutions". Second; an *internal reference factor* (IRFF), evaluates all missing references, that are required by the elements' meta-classes.

C. Consistency Metrics and Measures

Consistency of the SRS (CS) depends on two metrics: Formal Coherence (FCH) and Semantic Consistency (SCS) – see fig. 2.

Formal Coherence (FCH) is measured using graph-theory. When the document (SRS) is represented in a quasi-graph form, nodes represent project elements (requirements) and edges represent relationships (references). Although the references are directed, we evaluate Weak Coherence Factor (WCHF), which is appropriate for undirected graphs (edge direction is examined while measuring correctness – beyond this paper). WCHF is based on Weak Coherence Count (WCHC), which means a count of subgraphs which are

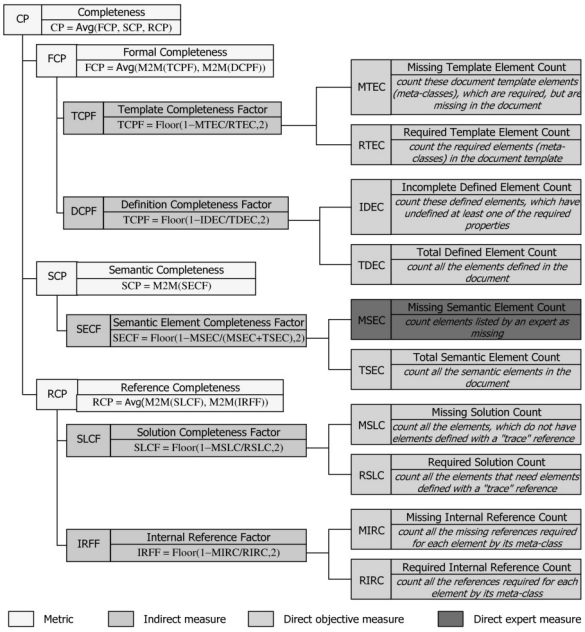

Fig. 1. Completeness metrics and measures

internally coherent, but mutually separate. Besides WCHF, we may also evaluate Relationship Strength Factor (RSTF), which represents the number of bridges in the graph (i.e. references which solely join two parts of the graph).

Semantic consistency (SCS) depends on Semantic Consistency Factor (SCF), which is measured by an expert. The expert must judge if the semantic elements (requirements) are consistent with other elements, and mark inconsistent ones.

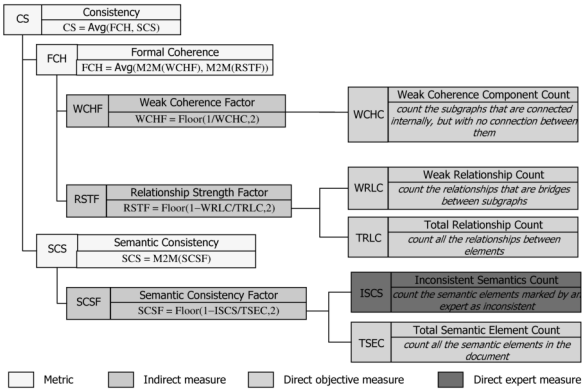

Fig. 2. Consistency metrics and measures

III. Experiment

A set of metrics and measures for completeness and consistency (shown above) was implemented in the IQuest system, which evaluated not only completeness and consistency, but also correctness, understandability, modifiability and verifiability. However we focus in this paper on the completeness and consistency, as these metrics showed the most interesting coincidence.

The IQuest system allowed us to import a sample SRS document written in Microsoft Office document format as an object-based requirements model where each requirement became an object and these objects were linked with references entered by a requirement manager. We used this object model to evaluate the consistency of a sample requirement specification. The IQuest system evaluated objective quality measures automatically, however an expert had to entered few data to evaluate the expert measures.

A. Evaluating the Quality of a Sample SRS Document

The sample SRS document for a Web Supermarket was elaborated for the research. The specification was presented as the set of requirement specification tables (see fig. 3). Note that references used to evaluate the SRS consistency are marked with '#'

FREQ_19	Order placing
Description:	The customer places an order with web service. S/he must find and select an article, determine the quantity and choose a credit card (if has more than one). System determines the total payment amount (price range!) and sends acknowledgement to the customer with the date of deliverance determined.
Applies to:	#USER_01 Customer
Results from:	#USER_01.TASK_04 Ordering
Associations:	#FREQ_15 Article searching #FREQ_17 Article selecting #FREQ_20 Order acknowledging
Sources:	#STKH_01 WDS Directors Board #STKH_07 WDS Sales Director
Priority:	very high

Fig. 3. Example of the functional requirement

Three incrementing versions were prepared and their quality was evaluated. The completeness was decided (for the research) to be the most important factor for requirements specification. The correctness and consistency were decided to be less important and the importance of other metrics was only supplementary. The weights of 40, 20, 20, 10, 5, and 5 were assigned to the quality factors. All the metrics were scaled from 1 (worst) to 6 (best).

The objective measures was not only ones which led to new requirements definition. In the second version also a business expert was asked to find out missing requirements.

B. First version – no quality requirements defined

First version of the SRS document was prepared very thoroughly – as much as 69 functional requirements were specified. Although the quality requirements were not specified, its quality was evaluated. No expert was asked for evaluation because the specification was not finished at that moment. The results are shown in the tab. I.

As no quality requirements were specified, the low value of completeness agreed with expectations.

TABLE I
QUALITY RESULTS OF THE FIRST VERSION OF THE SPECIFICATION

Metric	Weight	Value
Completeness	40	3.1
Consistency	20	5.8
Correctness	20	6
Understandability	10	5
Modifiability	5	5.95
Verifiability	5	5.8
Total quality		4.65

C. The second version – quality requirements consideration

Based on the first SRS, the next version with 36 quality requirements was developed. The quality requirements were revealed according to the template presented by table II. Exceptional, critical and breakdown situations were deliberated for reliability. For each such situation, a functional requirement was specified to prevent, or fix, this situation. That resulted in 99 new functional requirements. The results of the quality evaluation for the second version are shown in table III.

Despite expectation, completeness increased very little (from 3.1 to 3.65). Two explanations were discovered by the detailed result analysis (see table IV). First, it is impossible to achieve high completeness without filling the template completely (FCP). Second, the *Reference Completeness* (RCP) grew, but not satisfying. Despite intensive work, about 30% of elements were left without solution.

D. Third version – consistency consideration

In the third version, the formally "unresolved" goals; advantages, needs, tasks and problems, were deliberated. The emphasis was laid on increasing consistency. Some of the analyzed elements had already solutions in the form of functional requirements, but 30 new requirements had to be specified. The results of quality evaluations are shown in table V.

Although the increment of the consistency was very small (from 5.85 to 5.9), the completeness grew from 3.65 to 5.2. The main reason was the growth of *Reference Completeness* (RCP) from 4.44 to 5.9.

E. Relationship between consistency and completeness

Fig. 4a presents consistency impact on completeness in the three versions of the specification. This impact is significant (not only) in a mathematical aspect. More important is the impact of consistency on the number of functional requirements. As you can see (rys. 4b), this number grew about 3 times (from 69 to 198). It means that without quality evaluation and without consistency increment about 2/3 of functional requirements would be omitted and the requirements specification would be incomplete.

IV. CONCLUSIONS AND FURTHER RESEARCH

Consistency measurement of the Software Requirement Specification requires a quasi-graph representation of this document; with nodes representing requirements, and edges representing references between requirements. The document

TABLE II
THE STRUCTURE OF QUALITY REQUIREMENTS PART OF SRS

8. Quality requirements
8.1. Efficiency requirements
8.1.1. Effectiveness
8.1.2. Load capabilities
8.1.3. Stability
8.1.4. Scalability
8.2. Reliability requirements
8.2.1. Exceptional situations
8.2.2. Critical situations
8.2.3. Breakdown situations
8.2.4. Error resistance
8.2.5. Security and safety
8.2.6. Testability
8.3. Flexibility requirements
8.3.1. Portability
8.3.2. Localizability
8.3.3. Modifiability and configurability
8.4. Usability requirements
8.4.1. Usability
8.4.2. Understandability and learnability
8.4.3. Productivity

TABLE III
QUALITY RESULTS OF THE SECOND VERSION OF SPECIFICATION

Metric	Weight	Value
Completeness	40	3.65
Consistency	20	5.85
Correctness	20	5.9
Understandability	10	5.3
Modifiability	5	6
Verifiability	5	5.8
Total quality		4.9

TABLE IV
DETAIL COMPARISON OF THE COMPLETENESS EVALUATION FOR THE FIRST AND THE SECOND VERSION

Symbol	Metric or measure	v.1	v.2
CP	Completeness	3.1	3.65
FCP	Formal Completeness	4.5	5
TCPF	Template Completeness Factor	0.75	0.87
MTEC	Missing Template Element Count	2	1
RTEC	Required Template Element Count	8	8
DCPF	Definition Completeness Factor	0.94	0.92
IDEC	Incomplete Defined Element Count	19	45
TDEC	Total Defined Element Count	375	578
SCP	Semantic Completeness	-	5.95
SECF	Semantic Element Completeness Factor	-	0.99
MSEC	Missing Element Count	-	3
TSEC	Total Specified Element Count	375	578
RCP	Reference Completeness	4.05	4.44
SLCF	Solution Completeness Factor	0.62	0.7
MSLC	Missing Solution Count	65	71
RSLC	Required Solution Count	176	242
IRFF	Internal Reference Factor	0.99	0.99
MIRC	Missing Internal Reference Count	1	5
RIRC	Required Internal Reference Count	260	577

TABLE V
QUALITY RESULTS OF THE THIRD VERSION OF SPECIFICATION

Metric	**Weight**	Value
Completeness	40	5.2
Consistency	20	5.9
Correctness	20	6
Understandability	10	5.9
Modifiability	5	6
Verifiability	5	5.9
Total quality		5.6

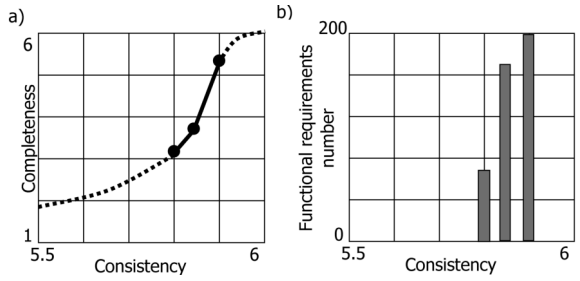

Fig. 4. Consistency impact to completeness (a) and to the number of functional requirements (b) in the three versions of the specification. Dotted line at (a) shows the expected impact.

template defines permitted and required elements that should appear in the document. A set of objective measures may be defined to automate evaluation of the quality of the document. These objective measures are supplemented with expert measures, which are evaluated in the document review process.

The vast impact of consistency on completeness can be noticed while evaluating quality of the SRS document. Consistency does not guarantee completeness, but it may help to reveal much more requirements than using traditional methods.

Basing on this preliminary study we prepared the next experiment. At the first part of the experiment we collected about 50 SRS documents from developers which used the proposed method. Next we gave the same project subjects for other developers using agile methods. We plan to compare the number of functionalities discovered with traditional and with agile methods. However the results will be available after several months.

REFERENCES

[1] "Ieee guide for software requirements specifications," *IEEE Std 830-1984*, pp. 1–26, Feb 1984. doi: 10.1109/IEEESTD.1984.119205

[2] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G. Kincaid, G. Ledeboer, P. Reynolds, P. Sitaram, A. Ta, and M. Theofanos, "Identifying and measuring quality in a software requirements specification," in *Software Metrics Symposium, 1993. Proceedings., First International*, May 1993. doi: 10.1109/METRIC.1993.263792 pp. 141–152.

[3] T. Shell, "System function implementation and behavioral modeling: A systems theoretic approach," *Systems Engineering*, vol. 4, no. 1, pp. 58–75, 2001. doi: 10.1002/1520-6858(2001)4:1<58::AID-SYS6>3.0.CO;2-Z. [Online]. Available: http://dx.doi.org/10.1002/1520-6858(2001)4:1<58::AID-SYS6>3.0.CO;2-Z

[4] R. S. Carson and T. Shell, "Requirements completeness: Absolute or relative? comments on "system function implementation and behavioral modeling"[syst eng 4 (2001), 58-75]," *Systems Engineering*, vol. 4, no. 3, pp. 230–231, 2001. doi: 10.1002/sys.1019. [Online]. Available: http://dx.doi.org/10.1002/sys.1019

[5] S. H. Kan, *Metrics and Models in Software Quality Engineering*, 2nd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002. ISBN 0201729156

[6] E. Knauss and C. E. Boustani, "Assessing the quality of software requirements specifications," in *2008 16th IEEE International Requirements Engineering Conference*, Sept 2008. doi: 10.1109/RE.2008.29. ISSN 1090-705X pp. 341–342.

[7] R. Pressman, *Software Engineering: A Practitioner's Approach*, 6th ed. New York, NY, USA: McGraw-Hill, Inc., 2005. ISBN 0077227808, 9780077227807

[8] S. Ambler, *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. New York, NY, USA: John Wiley & Sons, Inc., 2002. ISBN 047127190X

[9] "Ieee recommended practice for software requirements specifications," *IEEE Std 830-1998*, pp. 1–40, Oct 1998. doi: 10.1109/IEEESTD.1998.88286

[10] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," *the Journal of Information and Software Technology, Volume 45, Issue*, vol. 14, p. 2003, 2003.

[11] T. T. Moores and R. E. M. Champion, "Software quality through the traceability of requirements specifications," in *Software Testing, Reliability and Quality Assurance, 1994. Conference Proceedings., First International Conference on*, Dec 1994. doi: 10.1109/STRQA.1994.526392 pp. 100–104.

[12] G. Kotonya and I. Sommerville, *Requirements Engineering: Processes and Techniques*, 1st ed. Wiley Publishing, 1998. ISBN 0471972088, 9780471972082

[13] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt, "Automated analysis of requirement specifications," in *Proceedings of the 19th International Conference on Software Engineering*, ser. ICSE '97. New York, NY, USA: ACM, 1997. doi: 10.1145/253228.253258. ISBN 0-89791-914-9 pp. 161–171. [Online]. Available: http://doi.acm.org/10.1145/253228.253258