The Elusive Definition of Requirements Completeness

Luke Kasper
Phil Laplante

Penn State

IEEE Standard 830 "defines the following desirable qualities of a system or software requirement specification (SRS)

- correct,
- unambiguous,
- complete,
- consistent,
- ranked for importance and/or stability,
- verifiable,
- modifiable,
- traceable [1].

Of these qualities, completeness is perhaps the most difficult to achieve [2]. While it is well understood that an incomplete SRS can lead to disastrous effects [2] [3][4], the definition of "completeness" itself is inconsistent throughout the literature. The purpose of this article is to discuss the various definitions of "completeness" in order to help engineers to better communicate their intent and measure this elusive quality. The references also provide a starting point for those interested in further studying the notion of requirements completeness.

1. Requirements Completeness Defined

Boehm notes that a requirement is complete "to the extent that all of its parts are present and each part is fully developed" [4]. This definition is corroborated in [3]: "[completeness is] the sense of whether the specification contains all requirements and whether all requirements contained in the specification are completely specified".

Some authors describe " feasible completeness" [5], which is achieved when "there is no relevant statement about the domain, not yet included in the model, such that the additional benefit to the conceptual model from including the relevant statement exceeds the drawbacks of including it." Other authors discuss completeness "with respect to a reference model" [6]. That is, requirements are complete when "all requirement model elements satisfy a match" in a specific interaction pattern library. But these definitions imply the existence of such models or libraries, which in some domains may not exist, presenting a challenge to requirements completeness.

Still others introduce “internal completeness” [7] [8], which essentially means that if a concept is mentioned in a requirements specification, it must be elaborated upon within the same document. Multiple other works by Jaffe, Leveson and Hiemdahl follow a common definition stating that requirements are complete when “a response is specified with respect to every possible input and input sequence” [7][9][10]. In [10], the authors also point out that completeness is satisfied only when “every [has] a deterministic behavior defined for every possible input event.” This definition is offered within the context of safety crucial real-time systems, however.

Other definitions use a different perspective entirely. For example some authors look at the problem from an end-result perspective to state that completeness is “a lack of ambiguity from the implementation perspective. The specification is incomplete if the system behavior is not specified precisely because the required behavior for some events or conditions is omitted or is subject to more than one interpretation” [11]. This definition seems to imply, however, that the ends justify the means. Some authors tackle completeness from the perspective of goal satisfaction, that is, requirements are complete if all stated goals are satisfied [12]. The authors of [13] extend this definition by adding the caveat that “completeness is a notion relative to what is known about the domain.” Similarly, in [14] an attempt is made to find incomplete requirements by comparing the items in the requirements specification with a particular ontology. This observation reinforces the notion that the domain, or what is available in an ontology, is not nearly sufficient to ensure completeness and may provide a false sense of security, therefore making completeness validation more difficult.

Similar to this approach of utilizing the product of the requirements to help define whether the requirements were complete, is the definition of completeness as the “property that requirements are sufficient to distinguish the desired behavior of the program from that of any other undesired program that might be designed” [15]. This kind of definition can further increase understanding completeness, but provides little assistance in testing for completeness.

Some authors have rather limited definitions of completeness due to the particular domain addressed, without specifying a particular nomenclature for the definition. For example, the authors of [16] define completeness as simply the “presence of all event handlers or actions for condition guards of all events”.

Unfortunately, some studies discuss requirements and their completeness without further defining exactly what completeness means in said domain [17][18][19][20]. In [20], for example, it appears as if the authors utilized a ruleset within the tool studied to verify for completeness, but unfortunately neglected to include that ruleset. Others discuss “reasonable completeness”, with no further definition [21].

Conclusions and Future Work

It is difficult enough to achieve requirements completeness, especially when the notion of completeness itself is inconsistent. The definition proposed by Boehm [4] seems reasonable, but it is important to interpret completeness within each application domain and in the context of the operational

environment for the system in question. It would be helpful if Boehm's or another definition would emerge as a universal standard. But the problem of achieving requirements completeness, whatever that means, is still a very challenging one.

Future articles in this newsletter will discuss various ways of increasing requirements completeness. …

References

[1] IEEE Standard 830-1998, *Recommend Practice for Software Requirements Specifications*, IEEE Standards Press, Piscataway, NJ, 1998.

[2] Phillip A. Laplante, *Requirements Engineering for Software and Systems*, CRC/Taylor & Francis, 2009.

[3] Menzel, I., Mueller, M., Gross, A., Doerr, J., "An Experimental Comparison Regarding the Completeness of Functional Requirements Specifications," *Proc. 18th IEEE Int. Conf.* on *Req. Eng.*, Sep. 27 -Oct. 1, 2010, pp. 15-24.

[4] Boehm, B.W., "Verifying and Validating Software Requirements and Design Specifications," *Software,* vol.1, no.1, Jan. 1984, pp.75-88.

[5] Espana, S., Condori-Fernandez, N., Gonzalez, A., Pastor, O., "Evaluating the Completeness and Granularity of Functional Requirements Specifications: A Controlled Experiment," *Proc. 17th IEEE Int. Conf.* on *Req. Eng.*, Aug. 31 2009, Sep. 4, pp. 161-170.

[6] Kamalrudin, M., Hosking, J., Grundy, J., "Improving requirements quality using essential use case interaction patterns," *Proc. 33rd Int. Conf. on Softw. Eng.*, May 21-28, 2011, pp. 531-540.

[7] Heimdahl, M.P.E., Leveson, N.G., "Completeness and consistency in hierarchical state-based requirements," *IEEE Trans. on Softw. Eng.*, vol. 22, no. 6, Jun 1996, pp. 363-377.

[8] Jaffe, M.S., Leveson, N.G., "Completeness, Robustness, And Safety In Real-time Software Requirements Specification," *Proc. 11th Int. Conf. on Softw. Eng.*, May 15-18, 1989, pp. 302-311.

[9] Heimdahl, Mats P. E., Leveson, Nancy G., "Completeness and Consistency Analysis of State-Based Requirements," *Proc. 17th Int. Conf. on Softw. Eng.,* Apr. 1995, pp. 3, 23-30.

[10] Heimdahl, M.P.E., Czerny, B.J. , "Using PVS to analyze hierarchical state-based requirements for completeness and consistency," *Proc. IEEE High-Assurance Syst. Eng. Work.*, Oct. 21-22, 1996, pp.252-262.

[11] Sheldon, F.T., Hye Yeon Kim, Zhihe Zhou, "A case study: validation of guidance control software requirements for completeness, consistency and fault tolerance," *Proc. Pacific Rim Int. Symp. on Depend. Comp.*, 2001, pp. 311-318.

[12] Hassan, R., Eltoweissy, M., Bohner, S., El-Kassas, S. , "Formal analysis and design for engineering security automated derivation of formal software security specifications from goal-oriented security requirements," *IET Software*, vol.4, no.2, April 2010, pp.149-160.

[13] van Lamsweerde, A., Letier, E., "Handling obstacles in goal-oriented requirements engineering," *IEEE Tran. on Softw. Eng.,* vol. 26, no. 10, Oct. 2000, pp. 978-1005.

[14] Kaiya, H., Saeki, M., "Ontology based requirements analysis: lightweight semantic processing approach," *Proc. Fifth Int. Conf. on Qual. Softw.,* Sept. 19-20, 2005, pp. 223-230.

[15] Medikonda, B.S., Panchumarthy, S.R., "A framework for software safety in safety-critical systems," *SIGSOFT Softw. Eng. Notes, vol.* 34, no. 2, Feb. 2009, pp. 1-9.

[16] Lian Yu, Shuang Su, Shan Luo, Yu Su, "Completeness and Consistency Analysis on Requirements of Distributed Event-Driven Systems," *Proc 2nd IFIP/IEEE Int. Symp. on Theor. Aspects of Softw. Eng.,* June 17-19, 2008, pp. 241-244.

[17] Cox, K., Bleistein, S.J., Reynolds, P., Thorogood, A., "A contingency view of organizational infrastructure requirements engineering," *Proc. ACM Symp. on Appl. Comp.*, 2006, pp. 1497-1504.

[18] Hassan, R., Bohner, S., El-Kassas, S., Eltoweissy, M., "Goal-Oriented, B-Based Formal Derivation of Security Design Specifications from Security Requirements," *Proc. 3rd Int. Conf. on Avail., Rel. and Se.,* Mar. 4-7, 2008, pp. 1443-1450.

[19] Kumar, M., Ajmeri, N, Ghaisas, S, "Towards knowledge assisted agile requirements evolution," *Proc. 2nd Int. Work. on Rec. Syst. for Softw. Eng.*, 2010, pp. 16-20.

[20] Cardei, I., Fonoage, M., Shankar, R., "Model Based Requirements Specification and Validation for Component Architectures," *Proc. 2nd Annual IEEE Syst. Conf.,* Apr. 7-10, 2008, pp. 1-8.

[21] Hassan, R., Eltoweissy, M., Bohner, S., El-Kassas, S., "Goal-Oriented Software Security Engineering: The Electronic Smart Card Case Study," *Proc. Int. Conf. on Comp. Sci. and Eng.,* vol.3, Aug. 29-31 , 2009, pp. 213-218.