

REQUIREMENT DEFECTS DISCOVERY AND ANALYSIS IN AN ACADEMIC ENVIRONMENT*

Michael K. Baldwin


Department of Computer Science
Tennessee Technological University
mkbaldwin21@tntech.edu

Abstract


Development of the Software Requirements Specification (SRS) document is a critical task in the software development process as these requirements set the basis for further system development and overall project cost and schedule estimation. [1] The focus of this research study* is to identify the most frequently occurring defects, i.e., any deficiency that can affect the development process [2], in requirements specification documents developed by students conducting class term projects on system software development in an academic environment. The study itself consists of taking the set of SRS documents and inspecting them to locate requirements defects utilizing a checklist, based on the one described in [3]. The results of this study will identify checklist items most relevant to student projects in an academic setting and help to develop a new checklist better suited in such environment.

***This research was conducted as part of the course work for CSC 6700
(Advanced Topics in Software Engineering) during the fall semester of 2006.**

Introduction

- 
- It has been shown that complete, consistent, unambiguous, and correct documentation during the entire software development process improves the overall quality of a system. [2]
 - Other studies have also confirmed this by showing that poorly specified requirements are one of the major causes of problems that could have been caught or prevented earlier during the development process. [3]
 - The ultimate goal of this study was to determine what kind of defect, i.e. any deficiency in the requirements which can affect the software development process [2], occurs the most frequently.
 - By analyzing defects, overall requirements quality can be improved and defects can be prevented during requirements specification.

Methods Used

- 
- The study consisted of inspecting a set of software requirement specification (SRS) documents to locate defects utilizing a checklist, based on the one described in [6].
 - Each defect found was recorded and later analyzed all together to identify most frequently occurring defects and root causes of such defects.
 - The checklist was designed to aid in the defect detection process, and it also allowed for simple defect classification.
 - The categories and individual rules that are a part of the checklist were selected because of their applicability to an academic environment.

Results Overview

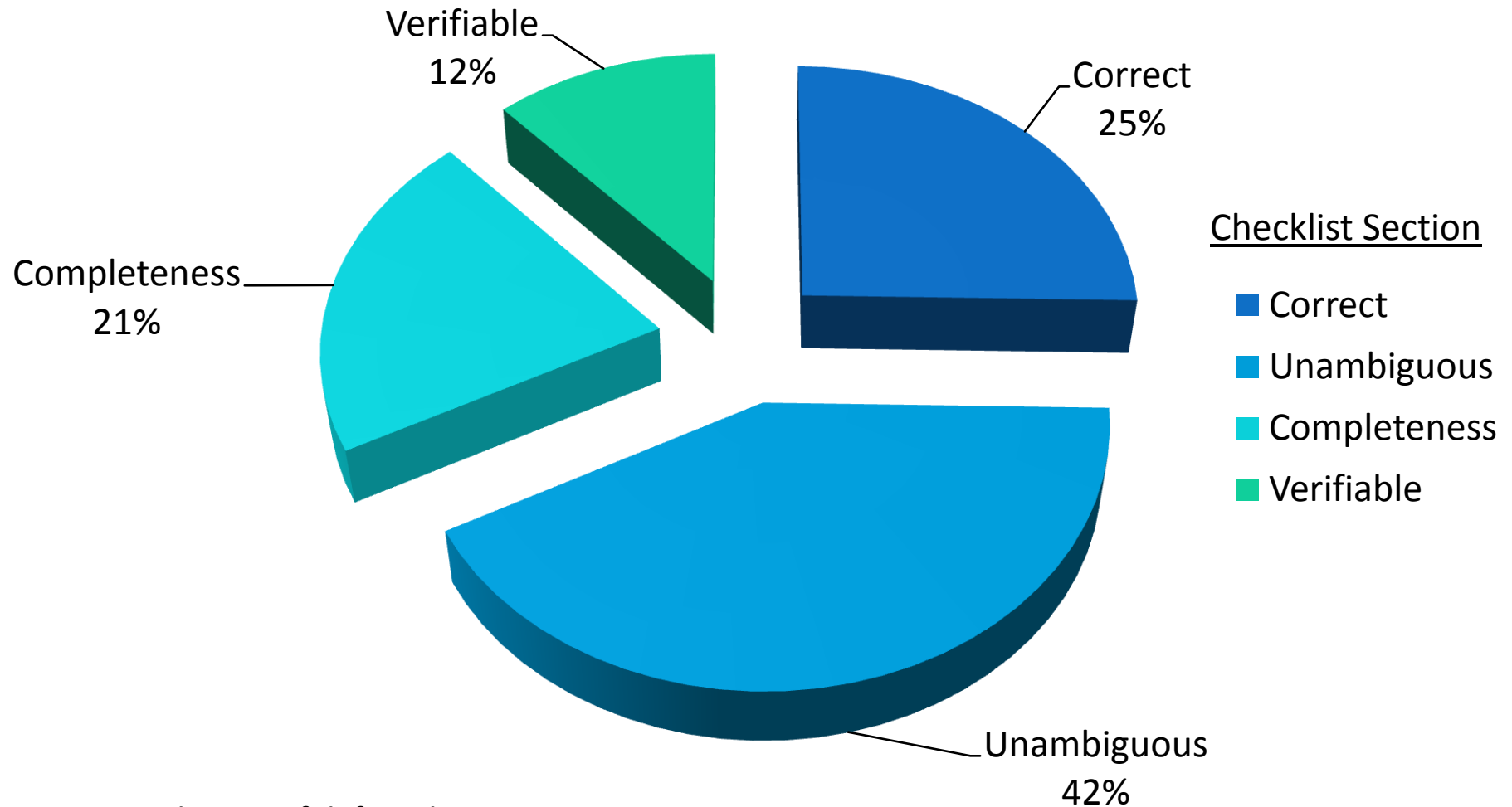


Fig. 1: Distribution of defects by category.

Defect Analysis in Correctness Category

1. Are all of the requirements necessary to meet the system objectives?
2. Are there requirements that seem illogical?
3. Are the requirements free of syntactical and grammatical errors?
4. Are all requirements listed in appropriate sections of the SRS?
5. For specific functions, are all specified inputs/outputs necessary and if so, are they described correctly?
6. Are contents, formats and constraints of all display outputs described?
7. Are there requirements that should really be considered as design?

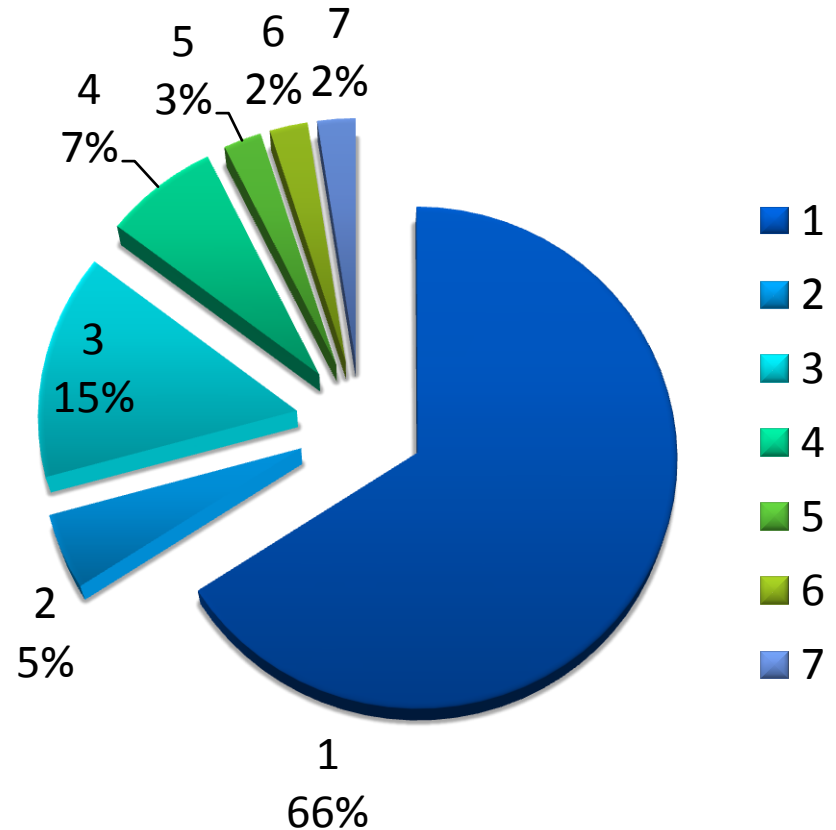
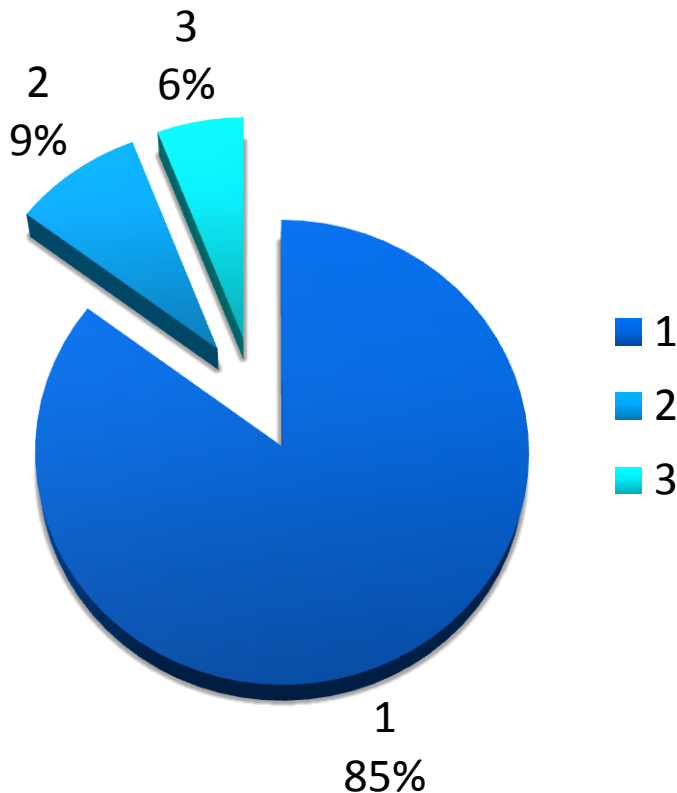


Fig. 2. Correctness defects by rule number.

Defect Analysis in Unambiguous Category



1. Is each individual requirement described in a discrete and unambiguous manner and with "one and only one" possible interpretation?
2. Do requirements consist of non-standard terms that are not defined or defined in a way that causes confusion?
3. Are the requirements distinct enough to generate test cases and detailed design specification?

Fig. 2. Unambiguous defects by rule number.

Defect Analysis in Completeness Category

1. Are all requirements adequate to meet objectives?
2. Are all requirements defined?
3. Is there any information missing from any individual requirement that is needed to effectively implement the system?
4. Is a standard format for requirements documentation with all essential parts present?
5. Is there any TBD (To Be Determined) section for any of the requirements?
6. Do the requirements provide an adequate basis for design?
7. Is all functionality related to individual requirements included?
8. Are all functions refined or elaborated to an appropriate level of detail?
9. Should the requirements be expressed in more or less detail in order to be effective?

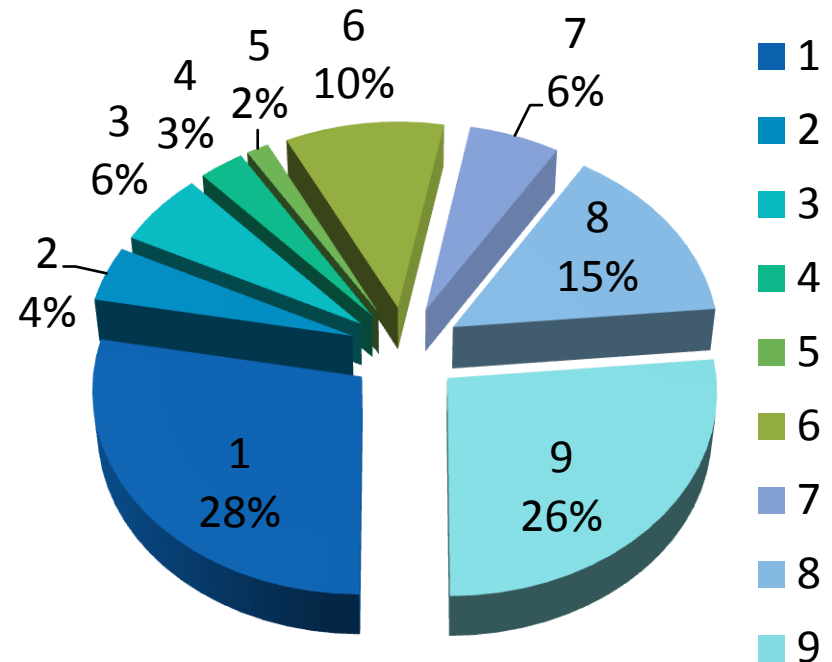
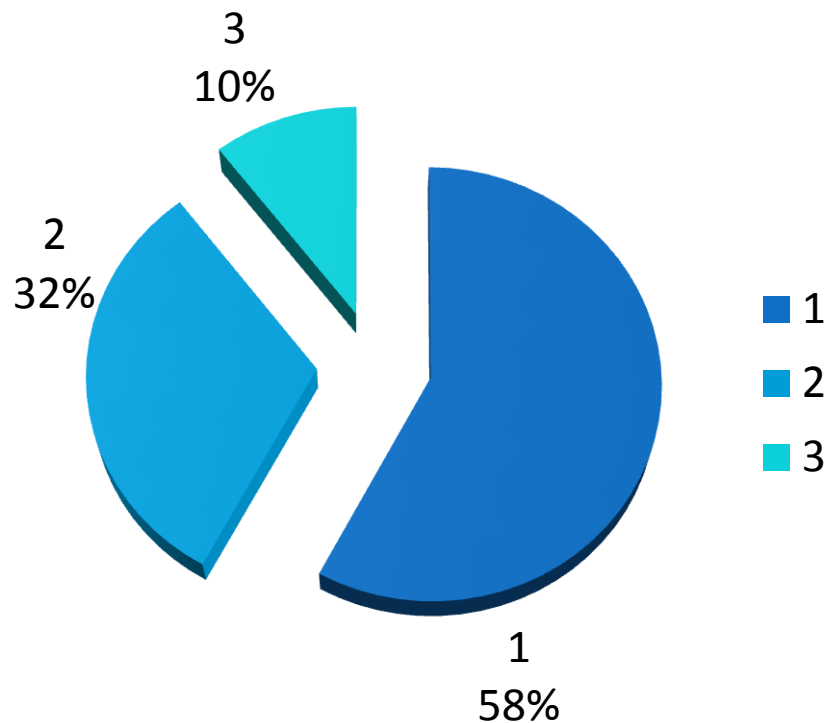


Fig. 4. Completeness Defects by rule number.


Defect Analysis in Verifiability Category




1. Are all objectives measurable, clear and reasonable?
2. Are there any requirements which cannot be verified due to multiple interpretations?
3. Are there requirements that cannot be verified by a process executed by man or machine?

Fig. 5. Verifiability defects by rule number.



Discussion

- 
- The overall results showed that the majority (42%) of the defects from the SRS unambiguous section of the checklist. Incompleteness and Incorrectness were the second most common defect.
 - 85% of the problems encountered in the unambiguous section were due to the lack of ***“one and only one”*** interpretation.
 - Verifiability of requirements was also a major problem. Many of the documents studied specified requirements such as ease-of-use and efficiency, which are virtually impossible to verify.
 - Over half of the defects associated with completeness were due to requirements which were not defined with enough detail to meet the objectives of the project.

Conclusion

- 
- Based on the results of this study, was determined that a simple checklist, like the one described here, would greatly reduce the number of requirements defects, with a minimal investment in time.
 - In fact this checklist was given to students in CSC-3700 (Software Engineering), during the spring of 2007, to use during the creation of their SRS documents.
 - Other studies have also shown similar results, confirming that inspections can be used to improve software artifacts, such as the SRS document, by ensuring that the artifacts accurately reflect the goals of the software system. [1]
 - Checklist-based reviews improve the inspection process by focusing the reviewers on the most important items in the document, while reducing the time spent looking for the defects. [3,5]
 - Utilization of a checklist-based reading approach could be both beneficial and practical for both student and commercial projects.

Future Work

- 
- 
- This study could be expanded to some other environment, e.g. business environment, with some customization.
 - A final area of related work would be to analyze the SRS documents from the spring 2007 section of CSC-3700 (Software Engineering) to determine if their documents had fewer defects due to the use of this checklist.

References

- [1] T. Javed, M. e Maqsood, and Q. S. Durrani, "A study to investigate the impact of requirements instability on software defects," *SIGSOFT Softw. Eng. Notes*, vol. 29, no. 3, pp. 1–7, 2004.
- [2] F. Shull, I. Rus, and V. Basili, "How perspective-based reading can improve requirements inspections," *IEEE Computer*, vol. 33(7), pp. 73–79, 2000. [Online]. Available: citeseer.ist.psu.edu/519652.html
- [3] B. Boehm and V. Basili, "Software defect reduction top 10 list," pp. 135–137, 2001. [Online]. Available: <http://www.cs.umd.edu/basili/publications/journals/J81.pdf>
- [4] S. Standards and C. of the IEEE, "Ieee recommended practice for software requirements specifications," 1993. [Online]. Available: citeseer.ist.psu.edu/429289.html
- [5] B. Brykczynski, "A survey of software inspection checklists," *SIGSOFT Softw. Eng. Notes*, vol. 24, no. 1, p. 82, 1999.
- [6] A. Siraj, "A software inspection checklist based on ieee recommended practice for software requirements specifications," *Proceedings: International Conference on Software Engineering Research and Practice*, June 2002.
- [7] T. Thelin, P. Runeson, and C. Wohlin, "An experimental comparison of usage-based and checklist-based reading," *IEEE Transactions On Software Engineering*, vol. 29, no. 8, pp. 687 – 703, August 2003.
- [8] G. H. Travassos, F. Shull, M. Fredericks, and V. R. Basili, "Detecting defects in object-oriented designs: Using reading techniques to increase software quality," in *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, 1999. [Online]. Available: <http://www.cs.umd.edu/basili/publications/proceedings/P86.pdf>