A Prototype Tool for Improving the Wording of Requirements

Joseph Kasser D.Sc., C.Eng., CM
Systems Engineering and Evaluation Centre
University of South Australia
School of Electrical and Information Engineering F2-37
Mawson Lakes Campus
South Australia 5095
Telephone: +61 (08) 830 23941, Fax: +61 (08) 830 24723
Email: joseph.kasser@unisa.edu.au

Abstract The majority of work on a project is in response to a set of need statements or requirements, consequently errors in requirements are a major cause of project cost and schedule overruns. This paper presents a simple concept based on the FBRET approach (Cook et al. 2001) that can minimise the production of poorly articulated requirements, and describes a prototype software tool that implements the concept.

INTRODUCTION

The majority of work on a project is in response to a set of need statements or requirements, consequently errors in requirements are a major cause of project cost and schedule overruns. These errors may be in the form of

- **Missing requirements –** results in lack of functionality and or performance.
- **Changing requirements –** results in extra work to be performed, lack of functionality and or performance, due to poor change management.
- **Wrong requirements -** results in lack of functionality, and or performance or erroneous functionality, and or performance to be delivered.
- **Poorly articulated requirements -** results in lack of functionality and or performance, extra work, erroneous functionality and or performance.

The focus of this paper is on the poorly articulated requirements, which have been shown to be a major cause of project cost and schedule overruns (Standish 1995; Kasser and Williams 1998). Kasser (1995) has shown how the cost of poorly written requirements can easily escalate the cost of a project by $500,000. Jacobs (1999) states that a 1997 analysis of the software development process performed at Ericsson identified *"Missing Understanding of Customer Needs*" as the main obstacle for decreasing fault density and lead-time. Related findings were aggregated under the heading "*No Common Understanding of 'What to Do*'". The counter measures to overcome these problems focussed on testing the quality of the requirements rather than producing good requirements. There was no proposal on how to get clear requirements, nor was there a clear understanding of what constituted a clear requirement. Applying Total Quality Management to Systems Engineering, this paper presents a simple concept for both preventing the production of poorly articulated requirements as well as testing for them after they have been produced, and describes a prototype software tool that implements the concept.

BACKGROUND

Requirements for writing requirements. Poorly articulated requirements have long been identified as a cause of project cost and schedule escalation (Hooks, 1993). To attempt to overcome this situation, Kasser and Schermerhorn (1994) identified a set of requirements for writing requirements. These were stated, as [the imperative construction section of a requirement]

1. Shall be
   - Complete
   - Verifiable
   - Achievable
   - Allocated as a single item to a single requirement number (concise)
   - Relevant
   - Grouped
   - Specific
   - Traceable

This work was partially funded from the DSTO SEEC Centre of Expertise Contract.

Poor Word	Occurrence	Category of Defect
Including	0	Unspecified superset
Will	0	By convention, not a requirement
Must	0	By convention, not a requirement
Shall	**1**	**Multiple requirement paragraph**
Should	0	By convention, not a requirement
Appropriate	0	Not measurable
For example	0	Descriptive, not measurable
Best practice	0	Descriptive, not measurable
Large	0	Not measurable
Many	0	Not measurable
And	0	Possible multiple requirement paragraph
Or	0	Possible multiple requirement paragraph

Table 1: Partial List of "Poor Words" in Requirements

2. Shall not be
- Vague
- Overlapping
- Redundant
- Contradictory

Written requirements that do not meet the "requirements for writing requirements" are poorly written requirements – by definition.

Kar and Bailey (1996) restated two of Kasser and Schermerhorn (1994)'s requirements as desired characteristics,

- Necessary - relevant.
- Consistent - not redundant and not contradictory).

And added the following desired characteristic

- Implementation-free.

Robertson and Robertson (1999) describe a Quality Gateway that tests requirements for:

- Completeness
- Traceability
- Consistency
- Relevancy
- Correctness
- Ambiguity
- Viability
- [not] Solution-bound
- [not] Gold plated, and
- [lack of] Creap.

However the Quality Gateway is based on a template rather than a tool. Moreover, during the years since the earlier papers were published, requirements management tools have proliferated, but they do not seem to be addressing the number of poorly written requirements. Thus Carson (2001) addresses some of these same issues seven years after Kasser and Schermerhorn (1994) raised them. The application of tool technology to the subject of reducing poorly written requirements is long overdue.

THE APPROACH

A number of "poor words" can be identified from the published papers, that when used in written project requirements, result in project requirements that violate the requirements for writing requirements. A subset of these words is shown in Table 1. Except for the word "shall" all other words must[1] be prohibited from appearing in a requirement statement. The word "shall" is allowed to appear once since it signifies a requirement. However if it appears more than once in the statement, then more than one requirement is deemed to exist in the statement. This is a multiple requirement in a statement scenario, which complicates traceability, and hence is prohibited from occurring by definition.

Elimination of these "poor words" from Requirements documents would reduce the effect of poorly written requirements on a project. The effect of these "poor words" was shown to postgraduate students in two ways.

1. They were presented in a Requirements Workshop module, which was, incorporated in coursework leading to the Master of Software Engineering (MSWE) and Master of Computer Systems Management (CSMN) degrees at University of Maryland University College (UMUC). The hypothesis for the workshop was preventing poorly articulated requirement from being written since once a set of "poor words" had been identified, the next step was to sensitise project personnel writing requirements to the "poor words" and their

[1] If this were stated as a requirement the word "shall" would be used.

effect on projects. During the course of the workshop, the students read a requirements document seeking out "poor words."

2. Students in one class in the MSWE/CSMN program covering the requirements phase of the system and software lifecycle (SLC) wrote requirements documents, which were then used by students in other classes covering software design, maintenance or testing. The students in the classes covering phases of the SLC subsequent to the requirements phase experienced the effect of the "poor words".

The result of this two-fold approach was effective, student produced requirements documents improved over several semesters to the point were the documents were no longer useable as examples of poorly written requirements documents.

The next stage in the process was to automate the "poor word" search by the use of a tool.

THE TOOL CONCEPT

Modern requirements management tools do not seem to have the capability of checking for "poor words" in requirements although James (1999) described the concept in the Precept Counsellor Review Mode and Bellagamba (2001) described an approach implemented in *Mathematica*, an interpreting language, because of its string-matching capabilities. The author of this paper thus wondered if there was an inherent difficulty in the implementation of the function in a Requirements Engineering tool and developed an operations concept for a tool that would perform the following functions.

1. Take a set of requirements exported or read from an existing requirements management tool.
2. Feed each requirement into a text parser, which matches each word in the list of "poor words" against the requirement.
3. Produce a report documenting each occurrence of a poor word.
4. Produce a Figure of Merit (FOM) for the document. The FOM is ratio of the number of requirements in a document and the number of defective requirements (those containing "poor words").
5. The "poor words" are stored in a table that can be edited without reprogramming the tool, so as to allow the user to add new "poor words" as, and when, they are identified.

A stand-alone implementation was chosen for several reasons including

- It was easier to construct a stand-alone prototype to perform the specific function.
- The tool could then be used without having to purchase and learn to use an expensive requirements management tool.
- Since requirements management tools were not offering the function, a stand-alone approach would avoid and potential implementation problems in the tool specific programming language. The focus was on the functionality not on extending a particular requirements management tool.

THE IMPLEMENTATION OF THE PROTPTYPE TOOL

Sets of sample requirements were obtained by extracting requirements from DOORS into text format. The prototype tool was written in Borland's Delphi (Visual Pascal) in the FBRET context (Cook et al. 2001) using the methodology proposed by (Kasser 1997). The implementation constraint was to develop a table-driven approach, so that the vocabulary of the basic software tool could be extended without any further programming. Thus a table of poor words was developed against which all the requirements had to be tested. The table contained the following information

- The "poor word".
- The number of times the word was allowed to appear in the requirement. For most of the time, the number was 0.
- The category of defect associated with the "poor word". Categories included multiple requirements in a single paragraph, and unverifiable requirement.

There were two initial design alternatives,

- Load each requirement, and parse it for all the words in the "poor word" table.
- Load the "poor words" table and then for each word in the table, check that it was or was not contained in each requirement.

The design finally implemented was to load the "poor word" table into memory, then one at a time, read each requirement and parse it for all words contained in the "poor word" table, and display or print a report.

A typical screen display from an early prototype of the tool is shown in Figure 1. It was simple to construct, was able to parse requirements, and provide an indication about their quality in the form of the FOM. The hardest part of the implementation was determining a table driven frame approach for storing the "poor words." The tool, being a prototype, displayed status information and showed each requirement being parsed, as well as the summaries. Further "bells and whistles" such as histograms, were implementable by the use of

component libraries, but were not incorporated in the prototype since they didn't add anything to the concept demonstration.

CONCLUSIONS

The prototype demonstrated that the concept was feasible. The report pointed out potential poorly written requirements, which then had to be analysed and repaired to become specific and verifiable. During the discussions in the process of repairing the requirements, ambiguities can be identified and resolved which leads to fewer missing requirements providing a multiplier effect which is more than just cleaning up poorly worded phrases.

The tool's function is useful for people who have to review requirements document in short-turn around situations as well as for those persons who are writing the documents.

Adding the functionality provided by this simple prototype tool to current generation requirements tools would result in better-written requirements. The requirements might still be poor for other reasons, but at least they would not contain poor words.

Future research could extend the prototype to test for the failure to meet some of the other "requirements for written requirements."

REFERENCES

Bellagamba, "Program to Identify Potential Ambiguities in Requirements Written in English", *The 11th INCOSE International Symposium, Melbourne, Australia, 2001.*

Carson, "Keeping the Focus During Requirements Analysis", *The 11th INCOSE International Symposium, Melbourne, Australia, 2001.*

Cook S.C., Kasser J.E. (2001) Asenstorfer, J., "A Frame-Based Approach to Requirements Engineering*", 11th International Symposium of the INCOSE*, Melbourne, Australia.

Hooks, I., "Writing Good Requirements", *Proceedings of the 3rd NCOSE International Symposium, 1993,* http://www.incose.org/rwg/writing.html, last accessed November 1, 2001.Jacobs, S., "Introducing Measurable Quality Requirements: A Case Study", *IEEE International Symposium on Requirements Engineering*, Limerick, Ireland, 1999.

James, L., "Providing Pragmatic Advice On How Good Your Requirements Are - The Precept "Requirements Councellor" Utility", *The 9th INCOSE International Symposium, Brighton, England, 1999.*

Kar, P., Bailey, M., "Characteristics of Good Requirements*", The NCOSE 6th International Symposium*, Boston, MA, 1996.

Kasser, J.E., Schermerhorn, R., "Determining Metrics for Systems Engineering", *The NCOSE 4th International Symposium*, San Jose, CA., 1994.

Kasser, J.E., "Improving the Systems Engineering Documentation Production Process", *The NCOSE 5th International Symposium*, St. Louis, MO., 1995.

Kasser, J.E., "Yes Virginia, You Can Build a Defect Free System, On Schedule and Within Budget", *The INCOSE 7th International Symposium*, Los Angeles, CA, 1997.

Kasser, J.E., Williams, V., "What Do You Mean You Can't Tell Me if My Project is in Trouble?", *First European Conference on Software Metrics (FESMA 98),* Antwerp, Belgium, 1998.

Robertson, S., Robertson, J., "Reliable Requirements Through the Quality Gateway", *10th International Workshop on Database and Expert Systems Applications*, Florence, Italy, 1999.

Standish (1995), *Chaos*, The Standish Group, http://www.standishgroup.com/chaos.html, last accessed March 19, 1998.

BIOGRAPHY

Joseph Kasser D.Sc. C.Eng, CM, has been a practising systems engineer for 30 years. He is the author of "*Applying Total Quality Management to Systems Engineering*" published by Artech House. Dr. Kasser is a DSTO Associate Research Professor at the University of South Australia (UniSA). He performs research into improving the acquisition process. Prior to taking up his position at UniSA, he was a Director of Information and Technical Studies at the Graduate School of Management and Technology at University of Maryland University College. There, he developed and was responsible for the Master of Software Engineering degree and the Software Development Management track of the Master of Science in Computer Systems Management (CSMN) degree. He is a recipient of NASA's Manned Space Flight Awareness Award for quality and technical excellence (Silver Snoopy), for performing and directing systems engineering. Dr. Kasser also teaches systems and software engineering in the classroom and via distance education.

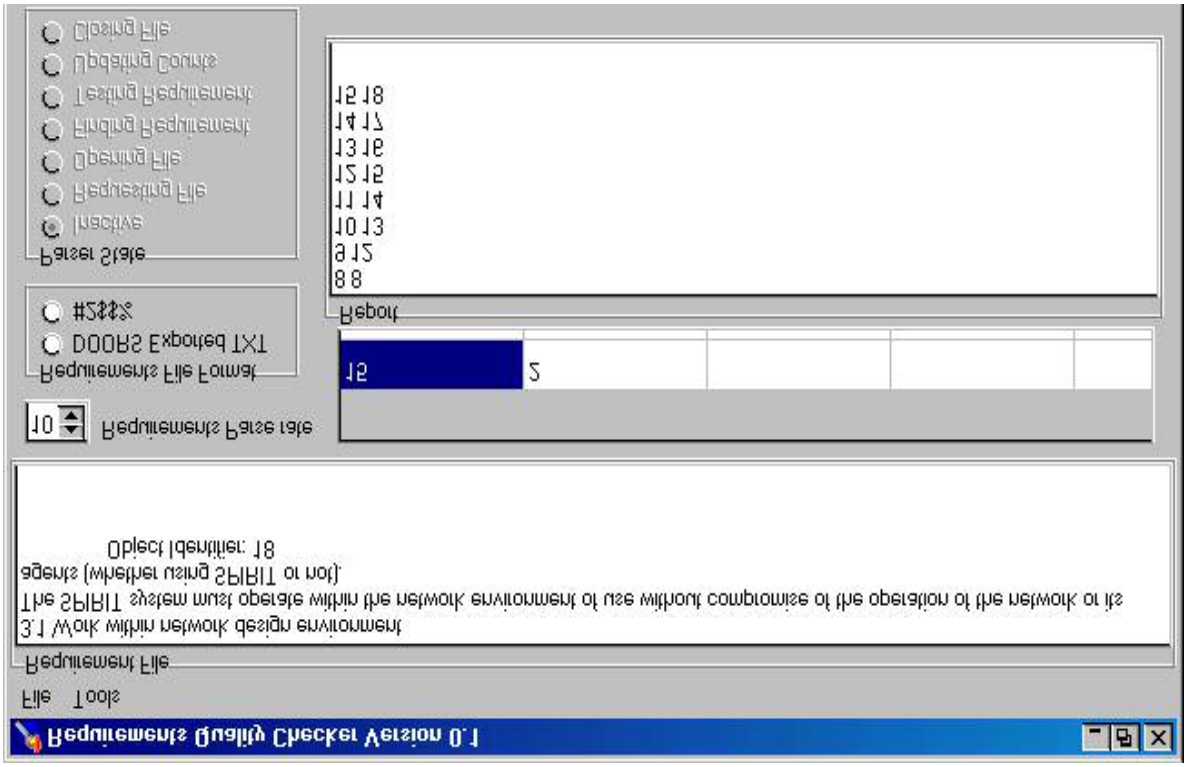

Figure 1 Early Prototype of the Requirements Parser