Towards improving the recognition and correction of poor requirements

Xuan-Linh Tran B Maths & Comp Sc, Master Info Tech

Joseph E. Kasser DSc, CEng, CM, MIEE

Systems Engineering and Evaluation Centre

University of South Australia

Mawson Lakes

South Australia, 5095

Abstract: *It has been documented in the last decade that poor requirements are one of the major causes of project failures (Standish 1995; Kasser and Williams 1998). Furthermore, a simple and effective solution to the problem is yet to be established. This paper discusses the use and benefits of a tool that represents a simple concept for early detection and preventing certain categories of poorly written requirements. The paper subsequently discusses early results from the utilisation of the tool and proposes further research based on the findings.*

Keywords*: Requirements engineering, postgraduate education, tools*

1. INTRODUCTION

It has been well documented in the last decade that poor requirements engineering, analysis and documentation are one of the main ingredients in the recipe for project failures (Hooks 1993; Kasser and Schermerhorn 1994; Jacobs 1999; Carson 2001). Nevertheless, the requirements engineering process is still undetermined as Goldsmith (2004) begins in his recent book that the process of "*defining business requirements is the most important and poorest performed part of system development"* . In fact, of the many commercially available tools that have been developed over the last decade for requirements engineering, none can assist users to differentiate between a good and a bad requirement.

Now in order to derive an effective solution to the problem of poor requirements, the following key questions have been identified:

1. How do we minimise the production of poor requirements?
2. How do we identify poor requirements?
3. How do we facilitate good requirements?

This paper briefly discusses the necessity of identifying, categorising and documenting the characteristics of good requirements through the use of TIGER PRO (Tool to InGest and Elucidate Requirements), a prototype software tool developed at the Systems Engineering and Evaluation Centre (SEEC) and some of the experience from the use of the tool in requirements workshops.

2. BACKGROUND

Of the many commercially available tools that have been developed over the last decade for requirements engineering including those listed in Table 1, none can assist users to differentiate between a good and a bad requirement.

Table 1 – List of various commercial requirements management tools.

Kasser and Schermerhorn (1994) showed how poorly worded requirements were able to easily add $500,000 to a project's cost and identified an initial set of requirements for writing good requirements to prevent that cost escalation. These requirements were later updated and published (Kasser 1995) as:

1. [The imperative construction section of a requirement] shall be written so as to be:

   - **Complete** - One of the basic categories of metrics.
   - **Testable** - If you can't test it, you can't demonstrate compliance.
   - **Relevant** - If it's not relevant to the system mission, it's not a requirement. This requirement is present so that (1) inherited requirements are considered

carefully before acceptance, and (2) people's wish lists are not accepted without discussion.

- **Achievable** - If you can't meet it, don't bother to write it as time will be wasted trying to test it, or discuss it during the later stages of the System Life Cycle (SLC).

- **Allocated as a single thought to a single requirement** - Each paragraph must also have a unique section number. This simplifies determination of completeness, ensuring compliance and testing.

- **Grouped by function** - Simplifies determination of completeness, ensuring compliance and testing.

- **Traceable;** both upward back to the source, and downwards into lower level documents.

2. To simplify determination of completeness, ensuring compliance and testing, Requirements **shall not** be written so as to be:

   - Redundant

   - Overlapping

   - Vague

3. In keeping with this theme, the following [poor] words shall never appear within the text of a requirement:

   - **Including**, **i.e.**, **e.g.**, **etc.** - These words imply that the requirement is a subset of an unspecified (and consequently un-testable) superset. You may use them to provide background information in the document. However, in the requirement, spell out each instance. Don't leave anything to the imagination.

   - **will** - A descriptive word to be used in the extrinsic information to provide background to the requirement when describing what the system will do once it is built.

   - **must** - It's an instruction not a requirement.

   - **should** - The conditional tense. It's a goal not a requirement.

Kar and Bailey (1996) also supported the above requirements but re-stated two extra requirements as desired characteristics:

- **Necessary** – relevant

- **Consistent** - not redundant and not contradictory.

They also added other desired characteristics including:

- **Rationale** for the requirement.
- **Verification** methodology.
- **Risk**.
- **Implementation-free**.

Applying an object-oriented perspective, Kasser (2000) in the context of managing change over the SLC and defined the following set of Quality System Elements (QSE) as additional attributes of a requirement:

- **Unique identification number** - the key to tracking.
- **Requirement -** the imperative construct statement in the text mode, or other form of representation.
- **Traceability to source(s) -** the previous level in the production sequence.
- **Traceability to implementation -** the next level in the production sequence. Thus requirements are linked to design elements, which are linked to code elements.
- **Priority -** knowing the priority allows the high priority items to be assigned to early Builds, and simplifies the analysis of the effect of budget cuts.
- **Estimated cost and schedule -** these feed into the management plan and are refined as the project passes through the Systems Development Life Cycle (SDLC).
- **The level of confidence in the cost and schedule estimates** - these should improve as the project passes through the SDLC.
- **Rationale for requirement -** the extrinsic information and other reasons for the requirement.
- **Planned verification methodology(s) -** developing this at the same time as the requirement avoids accepting requirements that are either impossible to verify or too expensive to verify.
- **Risk -** any risk factors associated with the requirement.
- **Keywords -** allow for searches through the database when assessing the impact of changes.
- **Production parameters -** the Work Breakdown Structure (WBS) elements in the Builds in which the requirements are scheduled to be implemented.
- **Testing parameters -** the Test Plans and Procedures in which the requirements are scheduled to be verified.

- **Traceability sideways to document duplicate links -** required when applying the QSE to an existing paper based project.

However, Carson (2001) still addresses some of the issues associated with poor requirements, where one of these issues is the syntax and language. Therefore, little seems to have changed over the years. The problem of poor requirements seems to be in the category of those problems for which a complete solution cannot be found, and consequently tends to be lived with.

3. PART OF THE SOLUTION - THE OBJECT-ORIENTED APPROACH

To minimise poor requirements, we need to identify their presence as early as possible in the production process. One immediate solution is to use a software tool that performs automatic syntax checking of poor requirements either as they are typed from the keyboard or ingested from a requirements document.

Various approaches to automating the detection of poor requirements have been discussed by: Alvarez, Castell et al. (1996); Kenett (1996); Wilson, Rosenberg et al. (1997); James (1999); Robertson and Robertson (1999); Bellagamba (2001). Kasser (2002) developed a prototype software tool that applied an object-oriented approach to the problem. The prototype evolved into FRED - The First Requirements Elucidator Demonstration tool (Kasser 2004) and then into a suite of simple tools known as PETS - Prototype Educational Tools for Systems and software engineering (Kasser, Tran et al. 2003) and TIGER PRO as the most recent tool.

TIGER PRO also saves time and minimizes biases and human errors. For example, in a recent requirements workshop, the tool checked 37 requirements in less than a minute, while the participants took up to 20 minutes to perform the same activity.

To facilitate the development of this tool, we first need to identify what constitutes poor words in the requirements. Then , we separate the poor words into various categories. These issues formed the basis for the development of the first tool FRED and subsequently TIGER PRO. These issues are discussed below.

3.1 Poor Word Identification

By definition, poorly written requirements are written requirements that do not meet the “requirements for writing requirements”.

Poor words are certain words that, if present in a requirement would point to the presence of a defect in the requirement, making the requirement unverifiable, or by virtue of having multiple requirements in the paragraph complicates ensuring completeness of testing.

Gathering information from experience and the literature (Hooks 1993; Kasser 1995; Kar and Bailey 1996; Wilson, Rosenberg et al. 1997), a generic list of these poor words was identified and associated with the tool (Kasser 2002), some are shown in Table 2. The list comprises a matrix with three columns: the poor word, the number of occurrences allowable for the poor word and the type of defect of the poor word. Except for “shall” which is allowable once since it signifies a requirement, all other poor words are not

allowable in a requirement. The poor word list grows by experience, expandable by users of the tool as they discover new ones. Any time the Test and Evaluation function have to clarify the meaning of a word, it is a candidate for the “poor words” list on future requirement documents.

Poor Words	Occurrence	Category of Defect
Adequate	0	Descriptive, not verifiable
And	0	Possible multiple requirement paragraph
Appropriate	0	Descriptive, not verifiable
Best practice	0	Descriptive, not verifiable
But not limited to	0	Unspecified superset, not verifiable
Easy	0	Descriptive, not verifiable
For example	0	Descriptive, not verifiable
Including	0	Unspecified superset, not verifiable
Large	0	Descriptive, not verifiable
Many	0	Descriptive, not verifiable
Maximize	0	Descriptive, not verifiable
Minimize	0	Descriptive, not verifiable
Must	0	Use of wrong word
Or	0	Possible multiple requirement paragraph
Quick	0	Descriptive, not verifiable
Rapid	0	Descriptive, not verifiable
Shall	**1**	**Multiple requirements in requirement**
Should	0	Use of wrong word
Sufficient	0	Descriptive, not verifiable
User-friendly	0	Descriptive, not verifiable
Will	0	Use of wrong word

Table 2: A partial list of “Poor Words” in requirements

The reason for the undesirability of multiple requirements in a requirement can be shown from the following example taken from the construction of a test plan, which demonstrates the work that TIGER PRO could have expedited:

Consider the following requirement (ST-DADS 1992):

204.1 DADS shall automatically maintain statistics concerning the number of times and the most recent time that each data set has been accessed. These same statistics shall be maintained for each piece of media in the DADS archive.

Two vague phrases had to be clarified. In the absence of any traceability to an operations concept, the following clarifications were made:

- The vague phrase “automatically maintain statistics concerning” was interpreted to mean ONLY the “total number of times” and the “most recent time”. Thus it was interpreted as meaning that there was to be no test to determine if DADS kept a log of access information (times and user) as far as this requirement was concerned.

- The term “piece of media“ was interpreted to mean the physical disk on which the data was stored.

The requirement was then split into the following four requirements to simplify tracking the completeness of the test plans:

204.1a DADS shall automatically maintain statistics concerning the number of times ~~and the most recent time~~ that each data set has been accessed. ~~These same statistics shall be maintained for each piece of media in the DADS archive~~.

204.1b DADS shall automatically maintain statistics concerning ~~the number of times and~~ the most recent time that each data set has been accessed. ~~These same statistics shall be maintained for each piece of media in the DADS archive.~~

204.1c DADS shall automatically maintain statistics concerning the number of times~~ and the most recent time~~ that ~~each data set has been accessed. These same statistics shall be maintained for~~ each piece of media in the DADS archive [has been accessed].

204.1d DADS shall automatically maintain statistics concerning ~~the number of times and~~ the most recent time that ~~each data set has been accessed. These same statistics shall be maintained for~~ each piece of media in the DADS archive [has been accessed].

Leaving the sections of the requirement that were not being tested in place but stricken through clearly identified which section of the requirement was being tested. It also clearly showed the defects in the requirement. Note that the phrase ‘has been accessed’ has been moved in the last two sub-requirements to clarify the sub-requirement.

Building the Test Plan was a labour-intensive process. Each requirement had to be manually scanned for vague words, the ‘and’ and ‘or’ words, as well as further occurrences of the word “shall”. Having a TIGER PRO function parse the document and identify requirements needing clarification and splitting to ensure a complete test would have saved at least 200 person-hours of test planning effort; and DADS was not a large project!

3.2 Poor Word Defect Categories

(Kasser 2004) separated the poor words into five defect categories:

Defects type 1 - Multiple requirements in a statement: when “shall” appears more

than once. This implies complication in the Requirements Traceability Matrix (RTM).

Defects type 2 - Possible multiple requirements in a line: when words such as “and”, “or” appear in a requirement statement. This situation needs clarifying as it may be or may be not defective.

Defects type 3 - Not verifiable word: when an un-testable or un-verifiable word appears in the statement. For example, words like “best practice”, “etc.”, etc. In these cases, the meaning needs to be clarified for T&E.

Defects type 4 - Use of wrong word: when the words “should”, “must”, or “will” are used instead of “shall”. This ensures consistency and MIL-STD-961D compliancy.

Defects type 5 - User defined poor word: this type allows users to define words that ‘shall’ not appear in the requirements text for any number of reasons.

Most of the defects detected by the tool should be used as warning signals and for guidance only. It is up to the requirements engineer to decide whether the detected defects by the tool are really defects or not.

Even with today’s technology, no matter how advanced a software tool is, it cannot replace human common sense, judgement and experience. There are certain defects in requirements that are results of human factors such as human level of experience in the systems or the level of complexity in phrase construction dictates the outcomes of requirements. For these defects, the human level of experience, knowledge or expertise in the systems is needed to ensure the completeness and accuracy of the requirements.

3.3 FRED – a basic tool

FRED (First Requirements Elucidator Demonstration) is the first simple stand-alone tool developed based on the concept outlined above (Kasser 2004) . FRED was designed to automate the syntactic analysis on the requirements text. FRED ingested a set of requirements text and performed the elucidating process, namely:

- Scanning through each requirement text to look for presence of poor words using the algorithm:

  look for: “*space”* + *“poor word”* + *“space”*

- Then reporting each occurrence, and

- Finally summarising the result with a Figure of Merit (FOM) as a measurement for the quality of the document based on the presence or absence of “poor words”. The FOM is calculated using the formula:

  *FOM = 100 * (1-number of defects / numbers of requirements)*

However, FRED could only perform the basic steps required by our object-oriented approach with some limitations. Hence, TIGER PRO replaced FRED .

3.4 TIGER PRO – a more complete version

While performing the same core functionality as FRED, TIGER PRO (Tool to InGest and Elucidate Requirements) shown in Figure 1 provides the following additional functionality:

- Converting the ingested requirement document from text format into QSE database format, allowing the storing, editing, and refining of requirements.
- Embedding BADGER, a build-in agent that prompts a step-by-step guide to assis the user in building each requirement line from scratch.
- Documenting acceptance criteria, i.e. documenting verification method for requirement and clarification of poor words (if dealing with documents you cannot change).
- Storing, editing and generating other QSE attributes mainly keywords, rationale, traceability, priority, risk and cost estimates.
- Exporting the report in text and graphical format.

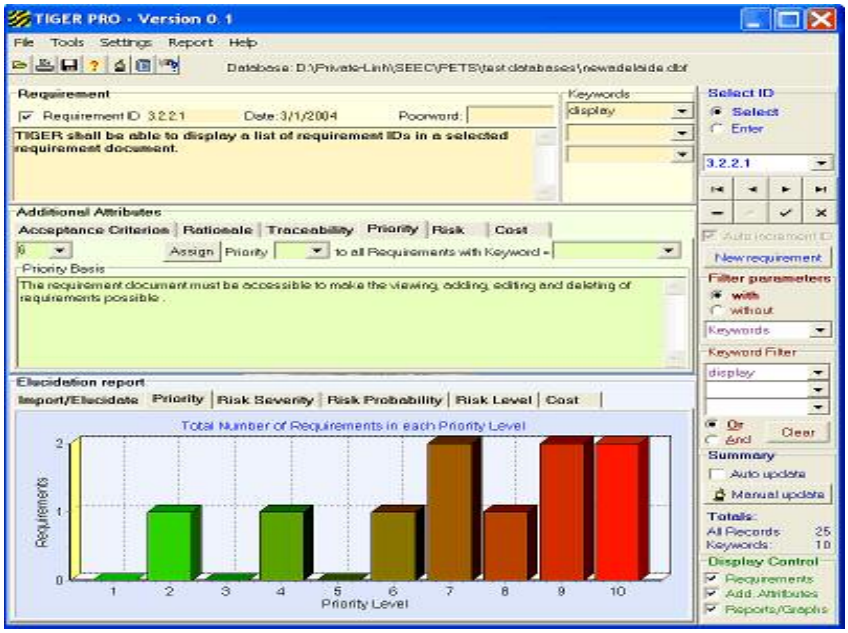

Figure 1. Typical TIGER PRO main display

4. THE REQUIREMENTS WORKSHOP

In mid May 2005, TIGER PRO was used in a Requirements Workshop. There were 85 participants who were professionals in the Defence Industry involved in writing or using requirements. Some participants were there involuntary which may affect the results discussed herein. The workshop comprised lectures about the difference between good and defective requirements, the effect of defective requirements in the industry and concluded with a practical using TIGER PRO to elucidate requirements. Pre- and post-workshop questionnaires were handed out to participants. The questionnaires comprised a set of 37 good and defective requirements. For the pre-workshop assessment,

participants were required to read each requirement and decide if it was good or bad. With the post-workshop assessment, the same set of requirements was given but the order was randomly mixed. This time, not only were the participants required to read each requirement and decide if it was good or bad, they were also required to state the type of defect they found in the requirements.

4.1 Early Results

Comparing the number of correct decisions from the post-workshop questionnaire with the pre-workshop questionnaire shows an improvement of 71%. It was also noticed that some requirements proved to be difficult for most participants and some participants were confused when identifying different types of defect.

To assess whether the level of reading difficulty in the requirements given in the workshop had any affect on the participants' responses, both the pre- and post-workshop correct answers are plotted against the Flesch-Kincaid Grade Level readability score provided in Microsoft Word as shown in Figure 2 and Figure 3..

Flesch-Kincaid Grade Level score rates text on a U.S. grade-school level. For example, a score of 8.0 means that an eighth grader can understand the document.

The formula for the Flesch-Kincaid Grade Level score is:

$$(.39 \times ASL) + (11.8 \times ASW) - 15.59$$

where:

ASL = average sentence length (the number of words divided by the number of sentences)

ASW = average number of syllables per word (the number of syllables divided by the number of words)

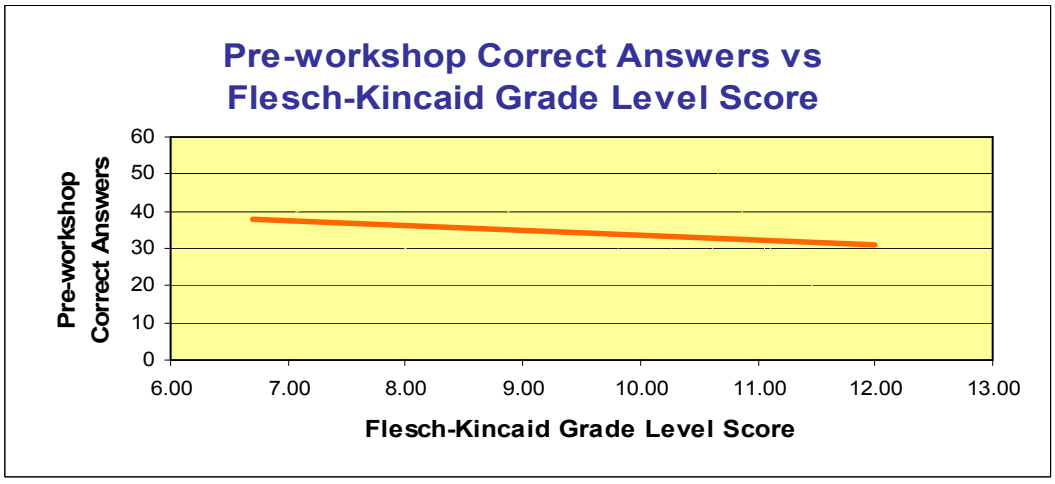

Figure 2. Pre-workshop correct answers vs Flesch-Kincaid Grade Level Score

From Figure 2 and Figure 3, the following deductions can be made:

- The linear trend line in Figure 2 - Pre-workshop correct answers shows that as the reading grade level of the requirements increase , the number of correct answers decrease.

- The linear trend line in Figure 3 - Post-workshop correct answers shows that as the reading grade level of the requirements increase, the number of correct answers increase.

- These trendlines suggest that participants recorded more correct answers after training with TIGER PRO. One explanation for this outcome is that the students paid more attention into the more complex requirements, hence producing better results. However, for the easier requirements, they did not perform as well. There is no concrete evidence to suggest the reason for this outcome.

- This outcome indicates that the post-workshop results are encouraging. This provides an attractive platform for future research and development, since most engineering requirements texts tend to have high reading grades.

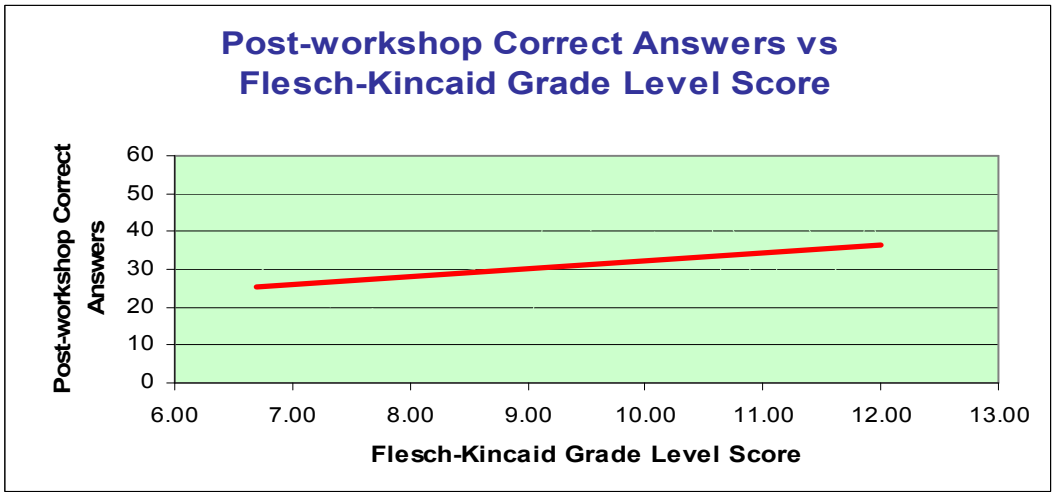

Figure 3. Post-workshop correct answers vs Flesch-Kincaid Grade Level Score

4.2 Future Plans

With the positive results obtained from the workshop, future plans for TIGER PRO and its implementation include a focus on:

- Correlating requirements with different levels of difficulty based on the four readability statistics provided by Microsoft Word, namely the Flesch Reading Ease score, the Flesch-Kincaid Grade Level index, the Coleman-Liau Grade Level index, and the Bormuth Grade Level index.

- Ensuring that the workshops are designed in a way that statistics obtained are based on both students and on requirement difficulty levels.

- Spend more time on explaining the different types of defects and their interpretation while providing more examples.

5. CONCLUSION

With the simple goal of providing a simple yet effective tool to assist users to perform early diagnosis on requirements to identify potential defects, TIGER PRO has focused user's attention on, and raised awareness about potentially defective requirements.

Needless to say, the tool should only be used as a mechanical lever to point out that there may be defect(s) in the text of the requirement. It does not replace the experience and judgement of the users. In the end, it is the requirements engineer who makes the final decision on whether the defects are there and how best to correct them.

6. ACKNOWLEDGMENTS

The authors would like to thank the following people in for their contribution in the data gathering, organising and proof reading process: Mr Bill Daniels, Mr Regis Queffelec, Miss Natalie Tran and Miss Emmatien Tran.

7. REFERENCES

Alvarez, J., N. Castell, et al. (1996). Combining Knowledge and Metrics to Control Software Quality Factors. Third International conference on Achieving Quality in Software, Florence, Italy.

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

Carson, R. S. (2001). Keeping the Focus During Requirements Analysis. Proceedings of the 11th International Symposium of the International Council on Systems Engineering (INCOSE), Melbourne, Australia.

Goldsmith, R. F. (2004). Discovering Real Business Requirements for Software Project Success. Massachusetts, Artech House Inc.

Hooks, I. (1993). Writing Good Requirements. Proceedings of the 3rd NCOSE International Symposium.

Jacobs, S. (1999). Introducing Measurable Quality Requirements: A Case Study. Proceedings of the IEEE International Symposium on Requirements Engineering, Limerick, Ireland.

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

Kar, P. and M. Bailey (1996). "Characteristics of good requirements." 6th International Symposium of the International Council on Systems Engineering (INCOSE).

Kasser, J. E. (1995). Applying Total Quality Management to Systems Engineering. Boston, Artech House.

Kasser, J. E. (2000). A Framework for Requirements Engineering in a Digital Integrated Environment (FREDIE). Proceedings of the Systems Engineering, Test and Evaluation Conference, Brisbane, Australia.

Kasser, J. E. (2002). A Prototype Tool for Improving the Wording of Requirements. Proceedings of the 12th International Symposium of the INCOSE, Las Vegas, NV.

Kasser, J. E. (2004). "The First Requirements Elucidator Demonstration (FRED) Tool." Systems Engineering.

Kasser, J. E. and R. Schermerhorn (1994). Determining Metrics for Systems Engineering. The 4th Annual International Symposium of the NCOSE, San Jose, CA.

Kasser, J. E., X.-L. Tran, et al. (2003). Prototype Educational Tools for Systems and Software (PETS) Engineering. Proceedings of the AAEE Conference.

Kasser, J. E. and V. R. Williams (1998). What Do You Mean You Can't Tell Me If My Project Is in Trouble? The First European Conference on Software Metrics (FESMA 98), Antwerp, Belgium.

Kenett, R. S. (1996). Software Specification Metrics: A Quantitative Approach to Assess the Quality of Documents. Nineteenth Convention of Electrical and Electronics Engineers, Israel.

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

Standish. (1995). "The Chaos Report." Retrieved March 19, 1998, from http://www.standishgroup.com/chaos.html.

Wilson, W. M., L. H. Rosenberg, et al. (1997). Automated Analysis of Requirements Specifications. Boston, MA, IEEE International Conference on Software Engineering.

ST DADS Requirements Analysis Document (FAC STR-22), Rev. C, August 1992, as modified by the following CCR's:- 139, 146, 147C, 150 and 151B.