Innovations Syst Softw Eng (2014) 10:77–91
DOI 10.1007/s11334-013-0225-8

ORIGINAL PAPER

The NASA automated requirements measurement tool: a reconstruction

Nathan Carlson · Phil Laplante

Received: 5 July 2013 / Accepted: 6 September 2013 / Published online: 17 September 2013

Abstract In the late 1990s the National Aeronautics and Space Administration (NASA) Software Assurance Technology Center (SATC) developed a tool to automatically analyze a requirements document and produce a detailed quality report. The report was based on statistical analysis of word frequencies at various structural levels of the document. The Automated Requirements Measurement (ARM) tool was further enhanced to include additional functionality such as custom definitions of quality indicators inputs for document analysis. By 2011 work on the ARM tool was discontinued. This paper describes the reverse-engineering and reproduction of the functionality of ARM. Recreating the functionality of this tool yielded valuable insight into certain quality metrics and provides a benchmark tool for future research. In addition to recreating and working with the ARM tool, this paper explores both existing and potential definitions of quality metrics in requirements specifications. Automated requirements analysis is a convergence of various fields of research, including text mining, quality analysis, and natural language processing. Informed by tangential areas of research in document understanding and data mining, recommendations are made for future areas of research and development in automated requirements analysis.

Keywords Requirements engineering · Text processing · Software quality

1 Introduction

High on the list of the most common factors associated with software project failure is "badly defined system requirements" [1]. This problem afflicts projects of all sizes, though with the increased cost of larger software projects the failures can be more catastrophic. Studies indicate that failures tend to occur most often in large-scale projects. These are three to five times more likely to fail than small-scale projects, due to the increased complexity and scope of work to be managed [1]. Many of the most significant software development failures can be attributed to requirements issues, ranging from missed requirements to excessive scope creep. Some of the more public software failures, such as the cancelled US Internal Revenue Service tax modernization effort, and the UK Inland Revenue tax credit overpayment scandal, end up costing billions of dollars to project sponsors [1].

Requirements describe what behaviors a system should exhibit, without specifying how they should be accomplished. The requirements specification document constitutes a key artifact early in the software development lifecycle that can help to control complexity. Effectively managing software requirements can also contribute to overall project success throughout the lifecycle. Code that is written and tests that are based on well-maintained software requirements will in turn be more likely to meet customer and user expectations.

1.1 Software requirements specifications

The requirements engineering process, focused on the production and refinement of a software requirements specification (SRS) or system-specific group of requirements, plays a critical role in the development of software. Hay likens the SRS to the score for a great opera. Without a score there can be only chaos; on the other hand, a quality score ensures that the activities and contributions of the various players in the opera are coordinated and integrated together [8]. Similarly, without an SRS a software development project faces

N. Carlson · P. Laplante (✉)
Penn State, Malvern, PA, USA
e-mail: plaplante@psu.edu

a significant amount of risk, particularly as the development process progresses.

Hay also describes a number of phases of requirements elicitation that illustrate the central role of the SRS in coordinating the various aspects of the software development lifecycle. This process begins with defining scope, using information from the strategic phase of project planning. The analysis effort itself is then planned in terms of specific tasks and resources. Information must be elicited from system owners and other stakeholders at this point. This includes information on the system to be developed, the enterprise in which the system will function, and current systems with which it will interact. Finally, he cites the need for transition planning based on careful evaluation of the organizational changes that will be required to implement the system. The SRS captures the output of these processes and guides system development as it progresses [8].

Leffingwell and Widrig [14] cite additional studies that underscore the centrality and importance of an SRS to the overall success of software development projects. Among these are a Standish study published in 1994 in which the three most commonly cited factors for projects being late or not meeting expectations were lack of user input, incomplete requirements, and changing requirements. They also cite a survey by the European Software Process Improvement Training Initiative (ESPITI) conducted in 1995 in which the two largest software problems in industry were requirements specifications and managing customer requirements.

1.2 The need for quality requirements

Given the central role of requirements in the lifecycle of a software project and in determining software success or failure, it follows that the overall quality of software is directly related to the quality of the requirements. The requirements individually and collectively must be of the highest possible quality to avoid risks to quality of downstream software products such as code and test cases.

Ferguson and Lami cite research showing the significance of quality in requirements to the incidence of defects at later stages of the software development lifecycle. Figure 1 illustrates this point using information collected by James Martin, indicating that over half of all defects are attributed to problems with requirements. The importance of requirements is even more striking when considering Fig. 2. This work demonstrates that over 80 % of rework effort can be attributed to defects related to requirements.

Wiegers [16] also discusses the need for quality requirements to minimize overall project cost and mitigate risk in later stages of the software development lifecycle. He illustrates the relative cost to correct a defect in requirements at different stages of the development process (Fig. 3).

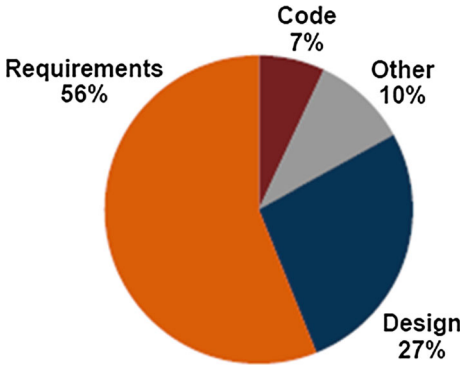

Fig. 1 Distribution of defects in software projects [7]

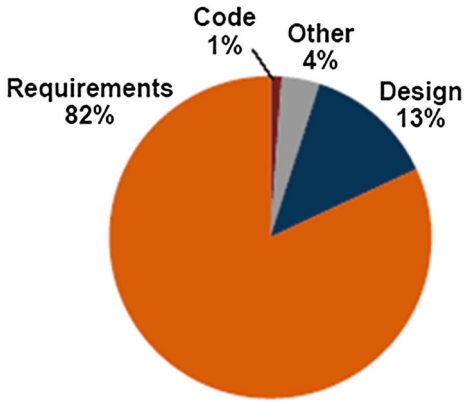

Fig. 2 Distribution of effort to repair defects [7]

Wiegers [16] suggests that requirements problems can increase development costs by up to 50 % and can contribute up to 80 % to rework costs due to errors discovered after delivery. If quality is enhanced by identifying and correcting defects at the requirements stage rather than later in the development of a software project, cost will be kept low and success will be more probable.

1.3 Automated requirements measurement (ARM) at NASA

The need to ensure quality requirements and SRS documents at NASA was a major driver for the development of the ARM tools. These included both the ARM tool and a follow-on eSMART tool that incorporated some additional features, such as the ability to customize some of the tool inputs and outputs. These tools were originally developed by the Software Assurance Technology Center (SATC) at NASA's Goddard Space Flight Center.

There are at least three versions of the original tool. An early version, ARM21, was a command-line tool that could be fed an input document in text format, and would produce a detailed quality report. The ARM95 version was released in 1996, and added a wizard user interface to the original

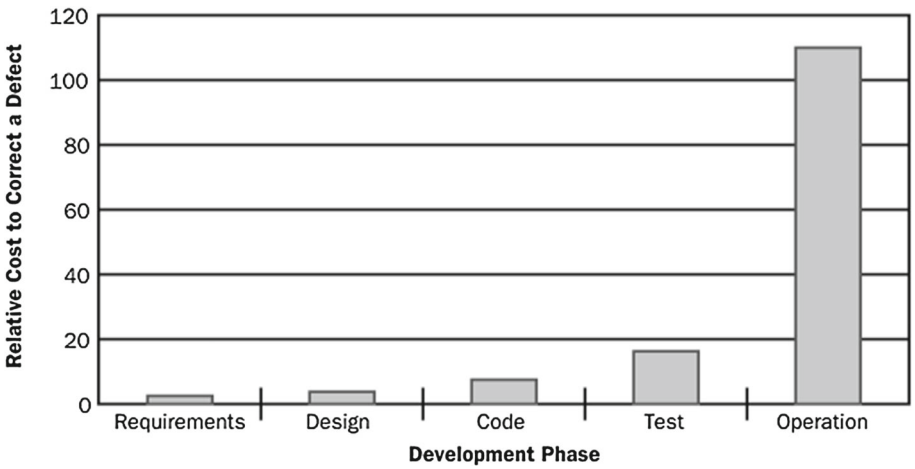

Fig. 3 Cost to correct requirements defects [16]

tool. The SATC developed an updated version of the tool called eSMART that included additional features, the most notable of which was an updated user interface, the ability to read from input specifications in Microsoft™ Word format, and the ability to specify custom word lists for the quality indicators used in the analysis process.

The ARM tool, in its various incarnations, was used across NASA organizations and by universities, government entities and private firms in several countries for several years until work on the tool ceased and the SATC was disbanded.

It is not entirely clear what happened to the tools after the SATC went offline. The SATC is still listed in the Goddard Space Flight Center organizations and projects[1], but the SATC web site is no longer active. Wyatt, et al, indicate that the work of identifying quality metrics for requirements, and perhaps also responsibility for maintenance of the existing tools, were assumed by the Independent Verification and Validation group in NASA [19].

2 Requirements quality

Most definitions of quality assume that the satisfaction of the user is of paramount importance. Included in many definitions is the absence of large numbers of defects. Almost all definitions, either explicitly or implicitly, hinge quality on the system requirements. The quality of the SRS, defined in terms of user acceptance and usefulness, is thus of highest importance to ensuring the overall quality of the software being produced. In addition, at a very high level, these definitions emphasize the need for user input, completeness and consistency in requirements specifications, which will reduce the number of defects in all phases of the software development lifecycle.

[1] http://www.nasa.gov/centers/goddard/about/organizations/org2.html

A pejorative view of software quality involves the risks to project success that manifest in requirements statements and SRS documents. Ambiguity in requirements is one of the most significant risks to project success, as the understanding of the developer, tester, and end user may vary widely if the requirements specification leaves room for interpretation. In addition to ambiguity, Wiegers [16] cites a number of other risks, including poor or missing user involvement, scope creep, gold plating, lack of detail, overlooked classes of users, and inaccurate planning. All of these risks can lead to poor quality user requirements specifications. Conversely, good requirements contribute to faster development, fewer miscommunications, more accurate system testing, and reduced overall project chaos.

2.1 Quality metrics for requirements

Wiegers suggests seven essential qualities of good requirements statements. Requirements statements must be complete, that is, they must fully describe the functionality to be delivered. Requirements must also be correct, or accurate in describing the functionality to be built. They must be feasible in terms of the known limitations of the system and its environment. They must be necessary either in terms of customer needs or external requirements or standards. Each requirement should be prioritized to enable the project manager to negotiate schedule, budget, resourcing and other constraints during the development of the system. In order to ensure that requirements are accurately interpreted, they must also be unambiguous-simple, straightforward and comprehensible statements of system attributes. Finally, good requirements statements must be verifiable, that is, there satisfaction must be provable in an objective way (Wiegers). Of these qualities, completeness is a particularly important but elusive [13].

Quality requirements can also be assessed in terms of the requirements specification document. For example, the IEEE Standard 830 suggests that a high quality SRS docu-

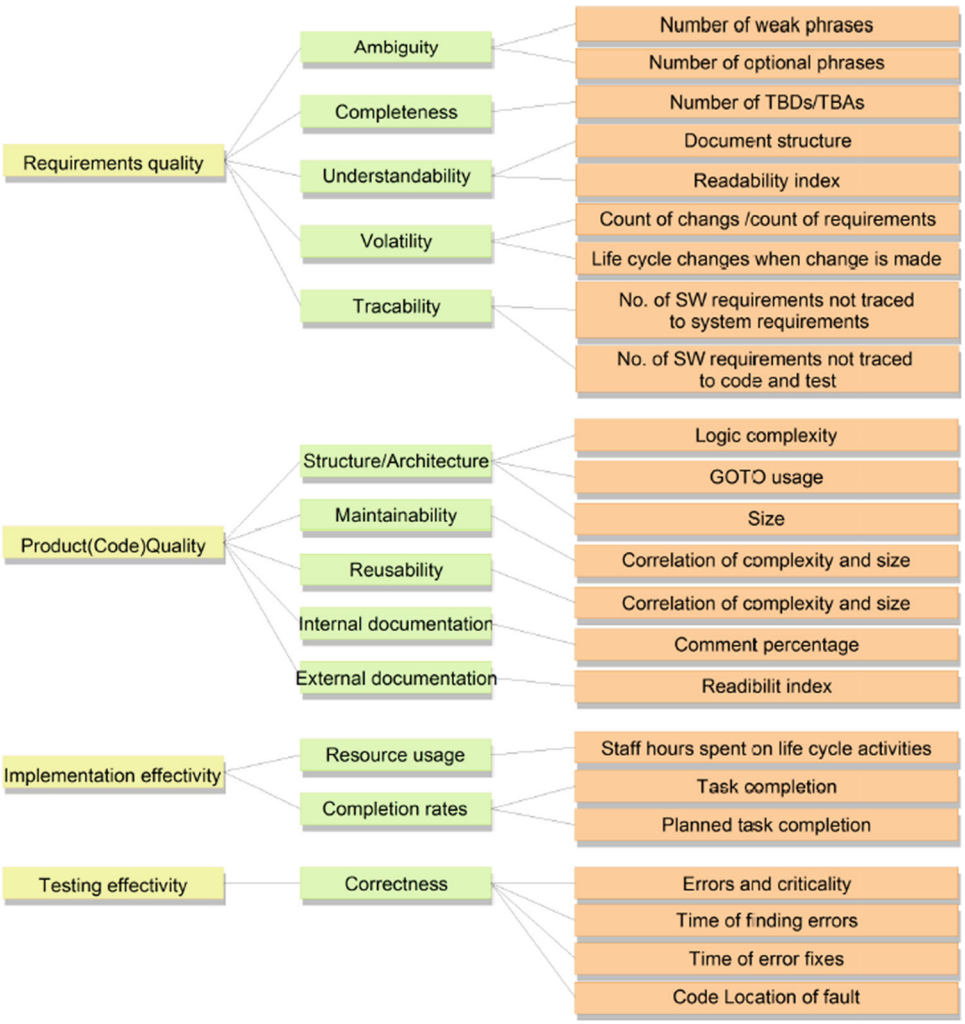

Fig. 4 NASA SATC quality model [19]

ment should be correct, complete, consistent, unambiguous, modifiable, traceable and ranked for stability and importance [10].

An expanded list of qualities exhibited by a good SRS can be found in Davis et al. [3]. Davis also summarizes the contributions of two decades of prior research into requirements quality metrics. While Davis' work predates the IEEE standard 830, the 24 attributes discussed in this seminal paper may be grouped under the subset of attributes given by the IEEE standard [10].

2.2 Quality models for measurement

Sets of quality goals and attributes have been proposed and refined, and together with product metrics comprise "quality models." Quality can be assessed manually by analyzing SRS documents using a quality model as a guideline [2]. However, because this is a manual process, it is both time-consuming and error-prone. The quality model used by the NASA SATC (Fig. 4) were developed by the SATC [9].

The volatility metrics, measured over time as the specification evolves, are not addressed in the ARM tool. The traceability metrics are also not measured, since these are integrated into the broader view of requirements management within an overarching software development life-cycle (SDLC) process. The ambiguity, completeness, and understandability quality attributes are measured by employing basic statistical techniques and document structure analysis.

3 Recreating an automated requirements measurement tool

Elcock and Laplante [6] outlined an approach to testing without requirements in which a requirements specification is reverse engineered using available development artifacts.

Table 1 Artifacts used in reverse-engineering ARM tool

Name	Description
ARM21 tool	The original command-line version of the ARM Tool
ARM95 tool (user interface)	An enhanced version of the command-line tool, including a wizard-style user interface
eSMART tool (user interface)	An enhanced version of the ARM95 tool including an updated user interface and additional features
Report output	The report produced when running automated requirements measurement tools against input requirements specifications
Partial Source Code (BASIC)	Partial (incomplete) source code for the ARM95 tool, including some of the document analysis processing
Hyatt and Rosenberg [9]	An article describing the requirements quality model used at NASA SATC, which forms the foundation for the ARM tool
Wilson et al. [17]	An article detailing the quality indicators and other metrics utilized in the development of the original ARM tool, and explaining the implementation of the tool at a high level

This approach was used to reverse engineer a requirements specification for the NASA ARM tool. The tool was then reconstructed from the derived requirements specification. The artifacts used in reverse engineering ARM are noted in Table 1.

The artifacts include working versions of the tool and published works.

3.1 Overall approach to reverse engineering

The overall method employed in reverse engineering the ARM tool was behavioral analysis using iterative high-fidelity prototyping. A number of iterative prototypes were developed in the PERL programming language. PERL is a natural candidate for prototyping textual processing tools, as it was originally designed for text extraction and reporting applications. Dominus [4] makes a good case for using PERL for advanced parsing as well, noting in particular the power of PERL regular expressions for lexical analysis.

Stripping away the user interface and usability aspects of the ARM tool, it fits a standard parser/generator paradigm. The tool needed to read an input requirements specification in some format, perform some analysis of the contents, and produce a summary report measuring the defined quality attributes of the specification. This evolved into the following sequence:

1. Read the input specification and flatten the document if necessary. Word documents may contain images, tables and figures-these were removed or stripped down to text format where necessary using the OpenOffice Word parser to produce a text-formatted file.
2. Reduce the input text to a sequence of lines of text, eliminating whitespace and consolidating lines where possible.
3. Parse each line, looking for paragraph numbers, used as both position indicators within the SRS, and to determine document depth.
4. Search each line of text for the key quality indicator words: imperatives, continuances, directives, options, weak phrases and incompletes.
5. For each matched word, note the line, word matched, count of word matches (in case of multiple words matching in a single line of text), line number, document section number, and document depth.
6. Once the entire document has been parsed, generate a standard report with the information noted, grouped by quality indicator. This report follows the format used in the eSMART tool, which is nearly identical to that used in the ARM21 and ARM95 tool releases.

Once this sequence was implemented and working in prototype form, the tool was used to generate reports for comparison with reports generated using the eSMART tool (the latest version of the ARM tools available). The outputs provided a basis for evaluating the accuracy of the eSMART and ARM tools, and assessing the overall effectiveness of the ARM tool approach.

To determine the effectiveness of the reconstructed tool reports from the same SRS document were generated from both the eSMART and tools. These were then compared using a grep-like tool. The differences between the eSMART and reconstituted ARM tool were manually verified against the input specification and the reasons for the differences between the two tools catalogued.

3.2 Improvements in reconstructed ARM tool

When the reconstructed tool report output was compared with the eSMART report output, some interesting differences emerged. For certain input documents the counts were found to be different between the two systems, as the approach to initial parsing and flattening of input requirements specifications was different between the two tools. In a number of cases, SRS documents had requirements embedded in tab-

Table 2 ARM tool quality indicator categories [17]

Quality indicators relative to individual requirement statements	Quality indicators for SRS as a whole
Imperatives	Size
Directives	Readability
Continuances	Specification depth
Options	Text structure
Weak phrases	

ular structures within a Word document. The reconstructed tool parsed out document sections paragraph by paragraph using an OLE automation library. The original tool appeared to convert Word documents to text directly, losing some of the requirements text embedded in tables. For this reason the detailed counts and reporting in the updated tool were slightly more accurate in the reconstructed ARM than in the original tool.

In addition, some of the calculated indicators were different between the two tools. Manual analysis of the source code for the original tool revealed some flaws in the identification of subjects, and thus counting their number. Paragraph identification, and thus document depth identification, was also suspect in the original ARM tools. Manual analysis of both source code artifacts and detailed comparison of the output of both tools revealed some flaws in both the calculation of document depth and the resultant computed metrics.

It is likely that some of these issues emerged due to the requirements specifications that were used as input to the tools. The original quality indicators were developed from manually analyzing a large subset of NASA SRS documents [17]. The reengineered ARM tool was tested using a variety of SRS documents from both academic and industry sources. While these documents conformed to the recommendations in the IEEE standards, the structure and language between these documents was less consistent than would have been the case with a subset of documents from a single industry and limited number of practitioners, as was the case at NASA.

3.3 ARM quality indicators

Central to the ARM tool quality measurement is the definition of quality indicators included in the tool. There are nine quality indicator categories identified by Wilson et al. that were included in the ARM tool and report output, listed in Table 2—ARM tool quality indicator categories. These are discussed in detail in subsequent sections.

The ARM suite of tools includes configurable indicator lists, that is, word lists for the different quality indicator categories in the left-hand column of Table 2—ARM tool quality indicator categories. The tools include a set of indicator words and phrases as described in Wilson et al. [17].

3.3.1 Imperatives

Imperatives are command words, indicating something that is of absolute necessity. The ARM tools include a list of imperatives in order from strongest to weakest. These are described in Table 3—imperative quality indicators.

3.3.2 Directives

Directives point to information within the requirements document that illustrates or strengthens the specification's statements. These indicator words are often used to make requirements more understandable. A high ratio of the total count

Table 3 Imperative quality indicators [17]

Imperative	Description
Shall	Dictates the provision of functional capability
Must	Establishes performance requirements or constraints
Is required to	Used as an imperative in specifications statements written in the passive voice. Rosenberg notes that this should not be used in requirements specifications [15]
Are applicable	Used to include standards or other documentation as an addition to the requirements being specified
Are to	Referenced in Rosenberg [15], but not Wilson et al. [17]; includedin the eSMART indicators as closely linked to use of the "will" imperative
Responsible for	Used in requirements documents written for systems whose architectures are predefined
Will	Indicates that something will be provided from outside the capability being specified
Should	Not frequently used in requirement specification statements, and always "very weak." Rosenberg notes that this should not be used in requirements specifications [15].

of directive statements to the total lines in the requirements specifications is a good indicator of how precisely requirements are specified. The ARM tools include the following directives:

- e.g.
- i.e.
- For example
- Figure
- Table
- Note:

3.3.3 Continuances

Continuances are words or phrases that follow imperative words and phrases in a requirement statement, and introduce more detailed specification. The extent to which continuances appeared in NASA requirements documents was found to be a good indicator of document structure and organization. However, frequent use of continuances could also be an indicator of requirements complexity and excessive detail. The ARM tools include the following continuances:

- below:
- as follows:
- following:
- listed:
- in particular:
- support:
- and
- :

3.3.4 Options

Options loosen the specification by allowing the developer latitude in implementing a requirement. This introduces risks to schedule and cost, as the final product is not under as much control. The ARM tools include the following option indicator words:

- can
- may
- optionally

3.3.5 Weak phrases

Weak phrases include words and phrases that introduce uncertainty into requirements statements. These indicators leave room for multiple interpretations, either indicating the requirements are defined in detail elsewhere or leaving them open to subjective interpretation. The total count of weak phrases is thus indicative of ambiguity and incompleteness of a requirements specification. The ARM tools include the following weak phrases:

- adequate
- as appropriate
- be able to
- be capable of
- capability of
- capability to
- effective
- as required
- normal
- provide for
- timely
- easy to

3.3.6 Size

Size in the context of the ARM tools includes counts of three indicators: total lines of text, total number of imperative words and phrases (as defined previously), and the total number of subjects of specification statements. These counts are tallied as a requirements specification is processed. The total number of subjects is determined by capturing all the unique combinations and permutations of words preceding imperatives in the specification. Ratios of imperatives to subjects and lines of text to imperatives provide indicators of how detailed the specification is, and how concise it is in specifying the requirements [17].

3.3.7 Readability

Readability is mentioned as a quality indicator in Wilsonet al. [17], but the specific metrics included in the paper are not calculated by the ARM tools. Readability statistics measure the ease with which an adult reader can comprehend a written document. These include the following:

- Flesch reading ease index: a measure of the average number of syllables per word and the average number of words per sentence.
- Flesch-Kincaid grade level index: similar to the preceding, but indicating a grade level for comprehension of a document.
- Coleman-Liau grade level index: a measure that uses word length in characters and sentence length to determine grade level for comprehension of a document.
- Bormuth grade level index: another metric using word and sentence length to determine a grade level for comprehension of a document.

3.3.8 Specification depth

Specification depth is a measure of the number of imperative statements found at each level of the document's text structure. It indicates how concise the document is in specifying

requirements, as well as the amount and location of background or introductory information included in the document [17].

3.3.9 Text structure

Text structure is a measure of the number of quality indicators found at each hierarchical level of the document. Documents including indicators down to as many as nine levels of hierarchy at NASA were found to be the most detailed, while high-level documents rarely included indicators deeper than four levels [17]. Well organized and consistently detailed documents were found to have a pyramid structure with fewer indicators at higher levels, and more indicators at deeper structural levels. Documents with a large amount of introductory and administrative information tended to exhibit an hourglass structure, with many indicators at the highest and lowest structural levels. Documents that addressed subjects at different levels of detail exhibited a diamond shape, with most of the indicators present in the middle structural levels.

3.4 Evaluation of ARM quality indicators

As previously noted the quality indicators included with the ARM tools are based largely on analysis of requirements specifications at NASA. This approach has the potential to skew the metrics calculated by the ARM tools to the requirements culture at NASA, though in analysis of documents from various academic and industry sources this did not appear to be the case. This does appear to contribute, however, to some assumptions built in to the measurement process. For example, continuances are evaluated independent of imperative statements. These normally follow imperatives but the tool does not check whether this is the case. The assumption is that they will always follow imperatives [17].

Some of the ARM tool quality indicator categories are insufficient indicators of the attributes they are intended to measure. For example, the weak phrases category includes a small set of weak phrases encountered while analyzing specification documents at NASA. It is conceivable that a requirements document could be highly ambiguous but lack a high count in this category. For instance, documents that make frequent use of certain adverbs tend to be more ambiguous than documents that avoid this language. All of the categories are dependent on a certain selection of grammar. Since they are meant to measure the strength or weakness of word choices in different grammatical categories, these metrics could be strengthened by employing natural language processing to identify a majority of instances of word usage in the different categories.

These same quality indicator categories are used to calculate metrics for the ARM quality report. For example, the total count of imperatives in the specification is used to calculate specification structure, which is in turn used to calculate ratios that can be used to determine how concise the specification is, and the level of detail in the specification as a whole. However, because the imperative indicators include weaker words and phrases such as "are to," "are applicable," and "responsible for," these ratios are not always accurately calculated. These measures, while seemingly objective, still require subjective interpretation in order to understand their implications. The tool also depends on the absence of these phrases in unrelated contexts. For instance, the phrase "are to" is not normally associated with the grammatical category of "imperatives."

Ambiguity in language is about more than word choice. The word "possibility" might be an indicator or weakness of expression, but context is the ultimate arbiter of meaning. If this word appeared in an explanatory phrase, it might add clarity to a statement. For example, "because of the possibility of system failure, the system shall…" could introduce a descriptive requirement with some explanation of its derivation. "The system shall eliminate the possibility of…" would be an example of the same word in a weaker requirements context.

The calculation of some of the quality indicators by the ARM tool was found to be suspect due to such issues of the ambiguity of natural language and grammar. The words and phrases included in the quality indicator categories had their origin in the analysis of a large number of requirements specifications at NASA. The writing culture of NASA requirements engineers may have been assumed in the determination of these indicators. For example, the number of unique subjects was determined by capturing the number of unique combinations of words preceding imperative statements in the document. Depending on whether the writer of a given requirement was paying attention to word order and use of imperative language, the identification of subjects in this way could be problematic. The NASA ARM tools do not produce a list of these unique subjects in the report, so it is difficult to verify their accuracy. This list was included in the reconstructed tool. Comparing the output of this tool to the count produced by the ARM tools, as well as a manual analysis of some requirements specifications, showed a small increase in accuracy of identifying subjects by limiting the way in which words were matched. An example of these analyses is shown in Table 4—subject identification in sample SRS. This is based on reports generated by the eSMART ARM tool, the reconstructed tool, and a manual analysis of the SRS including subject identification.

Table 4 Subject identification in sample SRS

eSMART subjects	Reconstructed tool subjects	Manually-identified subjects
(none discretely identified)	Application code	Application code
	Default GUI	A separate user interface specification for the system's default GUI
	All external libraries including their respective licenses	All external libraries including their respective licenses
	Application code and comments	Application code and comments
	Readers	Readers
	Java platform	Java platform
	Modularity	Modularity
	The JSR-API	The JSR-API
	Developers	Developers
	Vyasa is an open source project and	Vyasa
	Internationalized analysis, search and display	Internationalized analysis, search and display
	Methods and variables	Methods and variables
	Methods and classes	Methods and classes
	Product branding	Product branding
	Management systems	Document management systems
16 Total	15 Total	15 Total

By performing word frequency counts across a document, the ARM tools do not distinguish between introductory or explanatory text and actual requirements statements. This approach can skew the calculated metrics, as invalid matches are made and counted in the final results. Consider as an example the following phrase:

"Company X is developing a system which will enable authorized persons to vote electronically."

If this phrase were included in the introductory text of a document, the imperative count would include an imperative ("will") that does not appear in the context of a requirement statement. This sentence is descriptive not proscriptive. Furthermore, according to the rules stated for inclusion of a phrase in the total count of subjects, the subjects would include the phrase "Company X is developing a system which," not an actual subject of the requirements. The total lines counted for the specification could be skewed dramatically if a large body of introductory material were included.

This issue is exacerbated by contemporary requirements writing practice, in which requirements specifications are produced that include detailed use cases followed by derived requirements. The calculated metrics would more accurately indicate quality were they confined to the discrete requirements in the specification. But since words are counted regardless of their relative position in the document, or sectional context, for these documents detailed use case text will incorrectly be included in the calculation of quality metrics and will skew the results.

3.5 Comparison of eSMART and reconstructed ARM tools

Following is a comparison of the eSMART and ARM tools, based on the generated quality report statistics for four open-source software requirements specifications. While the reconstructed tool was tested and refined using a number of additional requirements specifications, these four provide a representative sample of some of the differences encountered between the original ARM suite of tools and the reconstructed tool.

Software requirements specifications were obtained from the web sites of the following open-source software projects:

- Warc-tools: a legacy software project consisting of tools to facilitate manipulation and management of web archive, or WARC files (http://code.google.com/p/warc-tools).
- Vyasa: a digital asset and document management system (http://vyasa.sourceforge.net).
- PeaZip: an open source archive utility (http://peazip.sourceforge.net).
- JHotDraw: a Java GUI framework for technical and structured graphics (http://www.jhotdraw.org).

The summary statistics are compared for two main indicators from the quality reports generated by the eSMART and reconstructed tools. First the quality indicator categories, such as imperatives and continuances, are compared between the two tools. Then the numbering and specification structure between the two tools are compared. These statistics

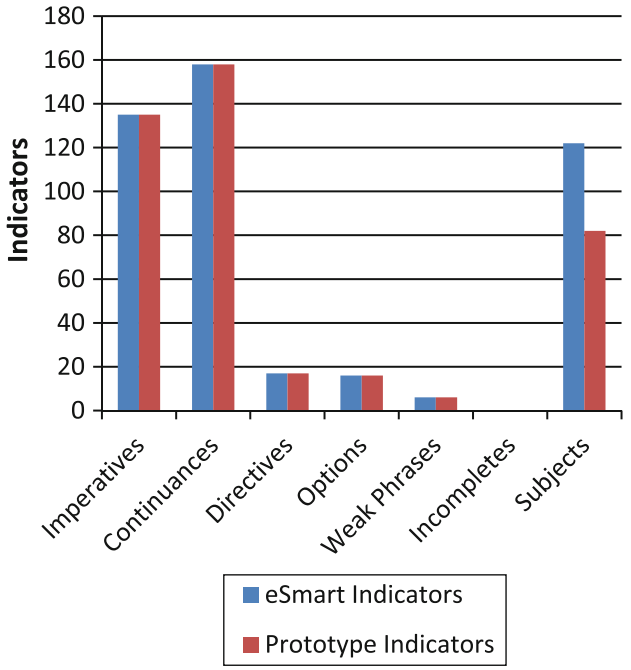

Fig. 5 WARC quality indicator comparison

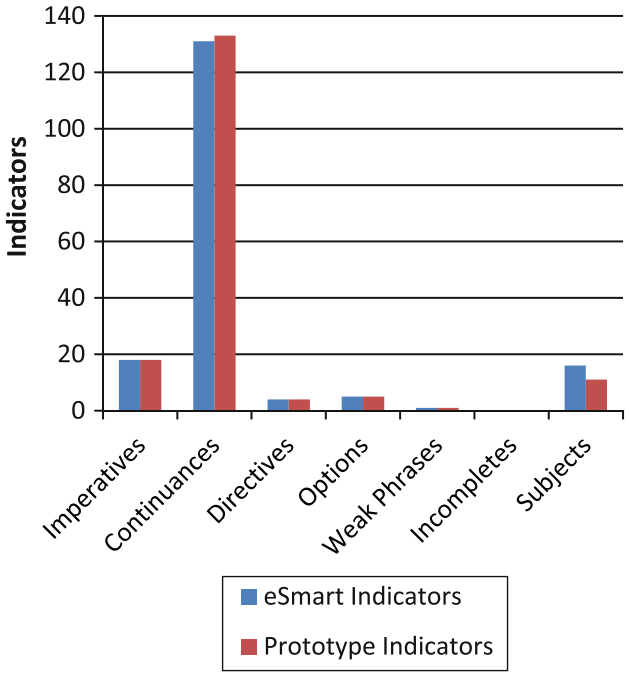

Fig. 6 Vyasa quality indicator comparison

form the foundation on which the rest of the quality report is generated, and are a good indicator of improvements in the reconstructed tool.

3.5.1 Quality indicator category comparison

Figure 5—WARC Quality Indicator Comparison shows the quality indicator counts generated for the WARC SRS in both the eSMART and reconstructed tools. The only difference in the generated indicator counts was the number of identified subjects. The regular expression pattern match used for identifying subjects is an improvement over the eSMART tool, which did not use regular expression pattern matching. The new approach was manually vetted using various sample specifications. Also the accuracy of subject identification can be better assessed using the reconstructed tool since a list of the subjects identified in the specification is included at the end of the report.

Figures 6, 7 and 8 show the differences between reported quality indicators for the Vyasa, PeaZip and JHotDraw specifications, respectively. In all cases where differences exist, the reconstructed tool reports slightly increased counts in indicators. These were assessed manually and found to be more accurate in the reconstructed tool than the counts generated by the eSMART tool. In all of these cases, while subject identification still suffers from a lack of natural language understanding, the reconstructed tool is more accurate and thus produces fewer subjects.

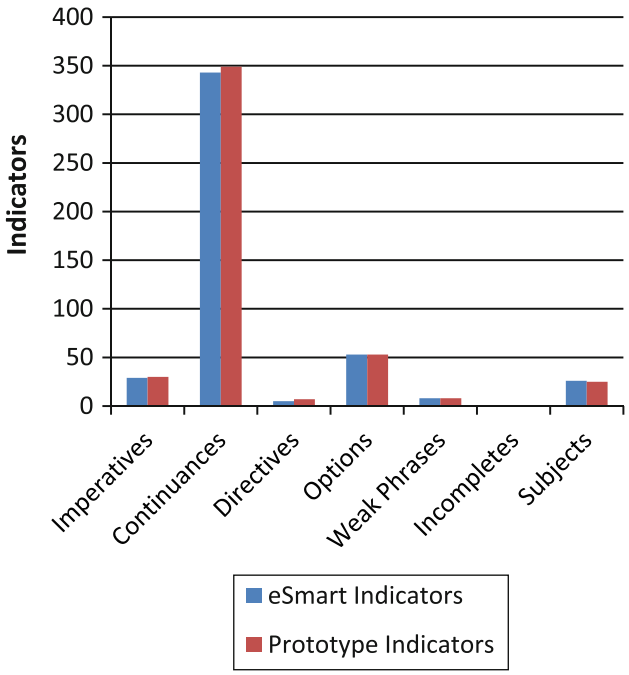

Fig. 7 PeaZip quality indicator comparison

Further tests by personnel at Transport for New South Wales, Australia (regular user of ARM tools for many years) on 13 standards for various railway, telecommunications, signaling and related systems were conducted using the ARM and reconstructed tool. Their findings were similar to the authors'—the reconstructed tool tended to have slightly higher counts for indicators, particularly

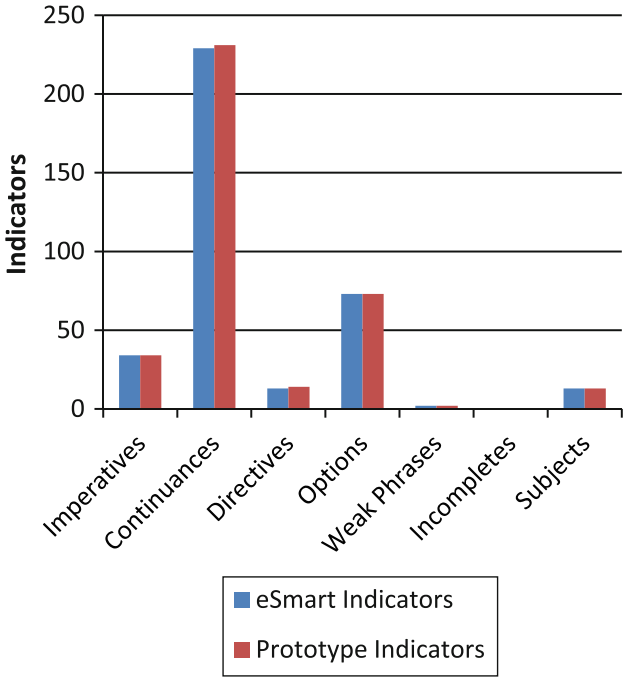

Fig. 8 JHotDraw quality indicator comparison

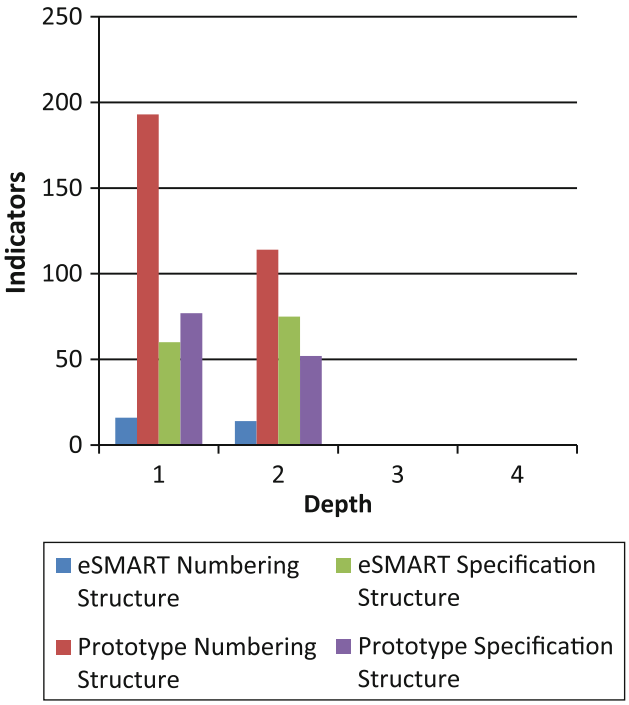

Fig. 9 WARC numbering and specification structure comparison

in continuances and directives. The documents analyzed by the Transport for New South Wales personnel can be found at http://www.asa.transport.nsw.gov.au/ts/railcorp-30_June_2013-snapshot/telecommunications-standards.

3.5.2 Numbering and specification structure comparison

The differences between the eSMART and reconstructed tool statistics are more striking when analyzing the numbering and specification structure counts. The numbering structure counts at the different depth levels of the WARC specification were very low in the eSMART report compared to the reconstructed tool output. The reason for the difference was immediately apparent when examining the output of both tools. The eSMART tool assigns a requirement statement to a particular level of a document only when it is preceded by an outline number. The reconstructed tool tracks the level of the document as it is parsing the input lines, so whether a statement is preceded by an outline or paragraph number is irrelevant to the depth at which it is placed in the document. While not as striking in the other three specification comparisons, the numbers for numbering structure are consistently higher in the reconstructed tool output compared to the eSMART tool output. This is due to the failure of the eSMART tool to correctly identify the depth at which requirements statements appear in certain documents. The eSMART tool is more sensitive to document formatting than the reconstructed tool.

Figures 9, 10 and 11 summarize the differences between the numbering and specification structure sections of the

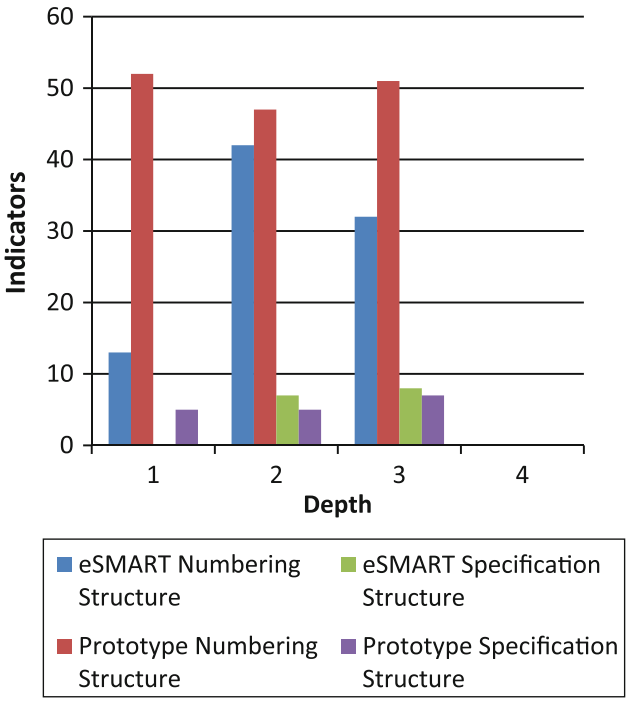

Fig. 10 Vyasa numbering and specification structure comparison

quality reports for WARC, Vyasa and PeaZip in both the ARM and reconstructed tools. These differences can be attributed to the improved accuracy of the reconstructed tool

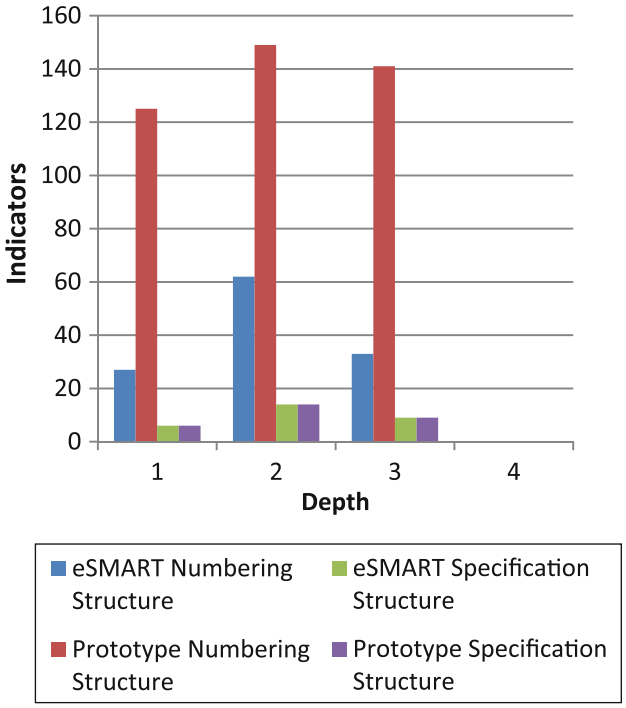

Fig. 11 PeaZip numbering and specification structure comparison

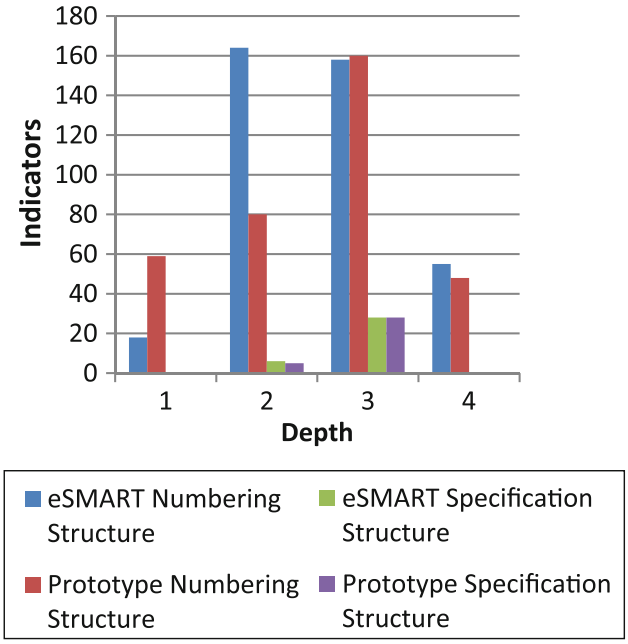

Fig. 12 JHotDraw numbering and specification structure comparison

in identifying the depth at which requirements statements are found.

The JHotDraw specification shows the opposite trend of the previous three specifications. This difference is due to the structure of the document and the unique numbering of the individual sections. There are a number of use cases with derived requirements interspersed throughout each section. This unique formatting contributes to the skewed report results. The next section will discuss some possible avenues for improvement of the reconstructed tool that could help ensure greater accuracy of the reported quality statistics (Fig. 12).

4 Future directions for ARM

4.1 Overview

No tool can independently measure quality of requirements specifications apart from review by a qualified analyst. For instance, completeness of a specification can be measured only in a limited way. The automated measurement tool might indicate that a document has a high degree of ambiguous language, but it cannot determine whether all the situations encountered by the system under analysis are covered in the specification. An automated requirements measurement tool can, however, quickly provide valuable feedback to the requirements practitioner.

4.2 Requirements input improvements

It was noted in analyzing the validity of the quality indicators used in measurement of requirements specifications that their accuracy was skewed by the presence of non-requirements text in a specification document. The ARM tool could be improved by eliminating introductory and administrative text from the automated analysis process. If the measurement process were focused on the requirements alone, instead of the entire document inclusive of non-requirements text, the calculated metrics would be more valid. This would be particularly noticeable when comparing different requirements documents across one or more organizations, as the focus would be on the requirements themselves and not skewed by differences between organizational practices in documentation.

Closely related to context sensitivity is the use of requirements management tools. Significant portions of the ARM tool are devoted to correctly identifying requirements statements, and distinguishing these from other text in a requirements specification. The accuracy of the quality report is highly dependent on this identification process. One way of ensuring a more accurate assessment of the quality of a set of requirements would be integrating the ARM measurement tool into specific requirements management systems. Given an API that allows access to requirements statements independent of other aspects of the requirements specification such as use case text, and introductory material, much of the uncertainty and ambiguity of the metrics could be removed.

One potential downside of integration with a requirements management system would be the loss of the quality indicators with respect to the entire SRS, and relationships between individual requirements. This shortcoming could be mitigated by using groupings built in to the requirements management system. The calculated ratios could be made more meaningful by comparing quality indicator counts to total requirements, rather than total lines of text in the requirements document. Measures such as text structure would not be relevant in an RM-integrated tool, but could be replaced by other measures. For example, instead of measuring imperatives at different structural levels of a document, the tool could calculate number of requirements in specific functional groups.

4.3 Parsing improvements

4.3.1 Missing metrics

Notable in their absence from the ARM tool output are the readability metric calculations. One option for improving the usefulness of the reconstructed ARM tool would be the inclusion of these metrics as measures of the overall readability of the specification document. According to Wilson et al. this might not be as useful in the context of NASA requirements, which are of necessity written at a very high comprehension level due to the complexity of the requirements being documented.

Wyatt et al. [19] have identified additional measures in a reevaluation of the original ARM metrics. These measures were manually collected, and included the number of actions a requirement must perform, the number of data sources or interfaces the action may need to be performed against, and whether the requirement needs to address multiple conditions. These additional metrics are moving away from automated measurement to a more manual and interactive method of evaluating requirements quality.

4.3.2 Requirements patterns

Withall has identified a series of requirements patterns, or "reusable approaches to specifying particular types of requirements." He notes after analyzing a large collection of requirements specifications that "a significant proportion of the requirements fall into a relatively small number of types" [18]. He defines a limited number of requirement types, based on situations most commonly encountered in specifying software system requirements. By focusing on specific types of requirements, patterns are identified and refined, and then reapplied when specifying similar types of requirements. This helps to assure quality without having to analyze each individual requirement in an SRS. The same process can be applied to developing and cataloguing new requirements patterns. Once the pattern is vetted for quality it can be reapplied to requirements of the same general type.

Patterns-based requirements development is an emerging field of research. Using patterns for specifying requirements can reduce errors in automated measurement by ensuring a greater level of consistency. Issa and AlAli have done work on patterns-based requirements engineering, though with a focus on use case patterns [11]. There is little attention paid in this research to the quality of requirements. Instead the focus is on code generation and reverse engineering from code to use cases. Presumably the assessment of quality would emerge in the transitions between these different representations of the use cases.

Eckroth and Amoussou [5] describe one common approach to improving the quality of requirements by limiting the use of natural language in specifying requirements. The goal of this process is to simplify requirements and eliminate ambiguity and misrepresentation by limiting the use of natural language in expressing software requirements. Instead, the requirements are expressed in a simple framework format: "actions over object" using active voice and transitive verbs. Quality attributes of the system, which would normally be represented by non-functional requirements, can then be tied to these functional "primitives," which the authors contend will help establish a clear link between software requirements and their impact on software quality. The requirements themselves are made less ambiguous by employing standard patterns for definition, thus reducing the number of unique forms in which the requirements are specified.

4.3.3 Natural language processing

One of the most promising enhancements to an automated requirements measurement tool is in the area of natural language processing. Most of the issues encountered with the eSMART and reconstructed tools were due to the failure to correctly distinguish between different document sections, different types of statements (e.g., requirements versus explanatory text) or context in which quality indicator words appeared. Many of the quality indicator categories are incomplete or unclear because they are grammatical categories. "Imperatives," for example, are a larger category of verbs. The important terms classed in this category are modal auxiliary verbs. As words in common usage, some evaluation of their context is required to determine whether they appear in a requirements context or some other less critical context. Some of the phrases included in this category in the eSMART indicators do not fit either the category of imperatives or modal verbs. An example of this

is the inclusion of "are to" and "responsible for"—phrases which may have meaning in the context of NASA requirements engineering practices but may introduce more ambiguity in the context of general cross-industry requirements practices.

Ferguson and Lami [7] describe a tool called QuARS, for quality analysis for requirements specifications. This tool performs both lexical analysis and syntactical analysis against natural language requirements specifications. Lexical analysis is used to identify words and phrases that are vague or subjective, and can assess readability of specifications. Syntactical analysis can help identify weak phrases or verbs, implicit expressions, and under-specification. This approach is in some ways fundamentally different from the approach used by the eSMART and the reconstructed tools. The measurement of fixed quality indicators is replaced by the assessment of ambiguity inherent in the grammatical construction and word choice.

For example, consider the following statement: "The system shall calculate totals depending on market conditions." Using the eSMART and reconstructed tool approach, this sentence would be included in the count of imperative quality indicators due to the presence of the word "shall." However, even a cursory grammatical analysis would indicate that the sentence is ambiguous. The word "totals" is vague, but understanding this would require some domain knowledge, or without context recognition that the word "totals" is inherently ambiguous. The QuARS tool would identify this as a defective sentence because of the presence of the words "depending on" [7].

4.4 Other approaches to requirements understanding

Kof [12] suggests some ways in which NLP automation and manual decision-making by an expert can be combined in requirements engineering. This proposed process centers on a working ontology, a taxonomy of terms and "is-a" relationships that provides a common vocabulary for stakeholders. This ontology feeds back into the requirements specification, allowing corrections to be made dynamically, resulting in consistent terminology throughout the specification. Kof makes the point that the manual intervention of a requirements practitioner is desirable to ensure correctness, as complete automation may lead to incorrect results or inconsistencies. This observation underscores a significant contributor to ambiguity in requirements specifications, that is, inconsistency in terminology throughout a specification document. By adding some form of terminology extraction the reconstructed tool could be enhanced to assist the requirements engineer in identifying inconsistencies in terminology throughout the requirements specification.

4.5 Output options and alternatives

4.5.1 Rich text reporting

The ARM tool produced a quality report as output in a plain text format. Many of the metrics included in this report would be clearer if included in line with the text of the original report. Many companies use the change tracking and collaboration features of word processing programs such as Microsoft Word. The feedback provided via these means can be hidden or removed to produce a final document, or individually addressed. If the output of the ARM tool were incorporated into the original specification as additional metadata, much of the work to relate the quality report to the original specification would be eliminated.

4.5.2 Context-aware output

The ARM tool produced a static quality report with specific metrics for each quality indicator category included. It attempts to provide some contextual information by printing each line of text where a quality identifier was identified. This is sometimes difficult to relate back to the original specification. This is especially tedious when the format of the original specification is a Microsoft Word document, as the tool has flattened the file and thus modified the total number of lines in the file and relative locations of requirements sections. As tools have evolved, static analysis has been supplemented by context-sensitive and near real-time analysis such as spell-checking and grammar-checking such that when issues are identified, feedback is immediately provided to the writer immediately.

Some of the ARM metrics would be useful applied to such a "live editing" scenario. If the tool were integrated into an application like Microsoft Word, or a requirements management tool, it could notify the writer immediately if ambiguous language was detected, and suggest alternatives. This approach could be integrated with automated suggestion of requirements patterns to improve quality throughout the requirements authoring process [18].

5 Conclusion

The NASA automated requirements management tools were an important part of many organizations' requirements practices, and versions of the tool are still being used by various organizations even at this writing. While the tool was abandoned by NASA, it is hoped that the reconstructed tool can provide a platform for future research and will revive and extend the ARM toolset to satisfy a persistent need in the requirements engineering community. The reconstructed

tool is available for experimentation at http://test.scripts.psu.edu/users/p/a/pal11/cgi-bin/ARM.html.

Acknowledgments The authors wish to thank Manoj Keswani and Eric Li of Transport for New South Wales, Australia, for various discussions related to their use of the ARM Tools over the years and for experimenting with the reconstructed tool and providing feedback, and to Christopher Laplante for various discussions and prototyping related to the reconstructed tool.

References

1. Charette R (2005) Why software fails. Spectr IEEE 42(9):42–49
2. Ciolkowski M, Soto M, Deprez J (2007) Measurement Requirements Specifications. Specification of Goals for the QualOSS Quality Model. Technical Report, QualOSS Consortium
3. Davis A, Overmyer S, Jordan K, Caruso J, Dandashi F, Dinh A, Kincaid G, Ledeboer G, Reynolds P, Sitaram P, Ta A, Theofanos M (1993) Identifying and measuring quality in a software requirements specification. In: Proceedings of First International Software Metrics Symposium, pp 141–152, 21–22 May 1993
4. Dominus M (2005) Higher-order perl: transforming programs with programs. Morgan-Kaufmann, Burlington
5. Eckroth J, Amoussou G (2008) Improving aoftware quality from the requirements specification. In: SOD '07 Science of Design Symposium, Humboldt State University
6. Elcock A, Laplante P (2006) Testing software without requirements: using development artifacts to develop test cases. Innov Syst Softw Eng 2(3–4):137–145
7. Ferguson B, Lami G (2005) Automated Natural Language Analysis of Requirements. PowerPoint Presentation, Carnegie Mellon Software Engineering Institute. Available at www.incose.org/delvalley/data/INCOSE-preview-QuARS_21June05.ppt, last Accessed 30 June 2013
8. Hay D (2003) Requirements analysis: from business views to architecture. Prentice Hall PTR, Upper Saddle River
9. Hyatt L, Rosenberg L (1996) A software quality model and metrics for risk assessment. In: Proceedings of Product Assurance Symposium and Software Product Assurance Workshop, 19–21 March, 1996 (ESA SP-377)
10. IEEE Std 830–1998 (1998) IEEE Recommended Practice for Software Requirements Specifications, Los Alamitos, CA, IEEE Computer Society Press
11. Issa AA, Al Ali AI (2011) Automated requirements engineering: use case patterns-driven approach. Softw IET 5(3):287–303
12. Kof L (2005) Natural language processing: mature enough for requirements documents analysis? In: Natural Language Processing and, Information Systems, pp 91–102
13. Laplante PA (2009) Requirements engineering for software and systems. CRC, Boca Raton
14. Leffingwell D, Widrig D (2003) Managing software requirements: a use-case approach, 2nd edn. Addison Wesley Longman, Reading
15. Rosenberg L (1999) A methodology for writing high-quality requirements specifications and for evaluating existing ones. NASA Goddard Space Flight Center, Software Assurance Technology Center
16. Wiegers K (2003) Software requirements, 2nd edn. Microsoft Press, Redmond
17. Wilson W, Rosenberg L, Hyatt L (1996) Automated quality analysis of natural language requirement specifications
18. Withall S (2007) Software requirement patterns. Microsoft Press, Redmond
19. Wyatt V, DiStefano J, Chapan M, Aycoth E (2003) A metrics based approach for identifying requirements risks. In: Proceedings of 28th Annual NASA Goddard Software Engineering Workshop