Beta-testing a requirements analysis tool

Article *in* ACM SIGSOFT Software Engineering Notes · September 2014

DOI: 10.1145/2659118.2659128

3 authors:

Andrew Brooks
University College Roosevelt
43 PUBLICATIONS **983** CITATIONS
SEE PROFILE

Laura Krebs
University of Minnesota Duluth
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Brandon Paulsen
University of Minnesota Duluth
2 PUBLICATIONS **2** CITATIONS
SEE PROFILE

Beta-testing a Requirements Analysis Tool

Andrew Brooks
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812-2496, USA
abrooks@d.umn.edu

Laura Krebs
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812-2496, USA
krebs148@d.umn.edu

Brandon Paulsen
Department of Computer Science
University of Minnesota Duluth
Duluth, MN 55812-2496, USA
pauls658@d.umn.edu

ABSTRACT

Two requirements specifications were analyzed using three requirements analysis tools: a reconstruction of NASA's ARM tool, TIGER Pro 2.0, and a new tool. The new tool being beta-tested reported many more true positives than TIGER Pro 2.0, which in turn reported many more true positives than the ARM reconstruction. However, using its *default settings*, the new tool raised many false positives which resulted in unacceptably low values for precision. Feedback has been provided to the company developing the new tool. Lessons learned about beta-testing and requirements analysis tools are described.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*Tools*; D.2.5 [**Software Engineering**]: Testing and Debugging

General Terms

Performance

Keywords

beta-testing, requirements analysis

1. INTRODUCTION

Formal notations are typically understood by only a small subset of stakeholders on any development project. To facilitate communication and understanding among all stakeholders, natural language is often used to express requirements. The results of one survey found 79% of requirements documents were written in natural language [15]. (Use of formal methods is usually reserved for safety- and security-critical systems.) Ambiguities, however, can all too easily arise in natural language text. A thorough exposition of the ambiguity problem is provided in [11]. Below is the well-known, water level requirement which is ambiguous because it is unclear if the mean, median, root mean square, or minimum level is being talked about.

> Shut off the pumps if the water level remains above 100 meters for more than 4 seconds.

Several tools have been created to analyze requirements for ambiguities and other recognized weaknesses. In principle, the requirement writer can act on the comments or warnings from such tools and rewrite requirements in a better way. However, natural language is complex, and tools sometimes generate comments or warnings that would not get acted on. A tool faces rejection by the very people it is designed to help if it generates too many false positives. In a report summarizing work on the use of static analysis to find bugs in program code [12], the authors state the following:

> False positives do matter. In our experience, more than 30% easily cause problems. People ignore the tool. True bugs get lost in the false. A vicious cycle starts where low trust causes complex bugs to be labeled false positives, leading to yet lower trust.

The work reported here involved a determination of true and false positives to assess the acceptability or otherwise of the new tool being beta-tested.

An opportunity arose that allowed the authors to act as beta-testers for a new requirements analysis tool. The SWEBOK guide [3] defines beta-testing simply as providing trial use to external, potential users, before the software is released. Of course, as part of such trial use, feedback to the company or group who is developing the tool is expected. One online dictionary [1] distinguishes between open and closed beta-testing. In open beta-testing, the tool is available to the public. In closed beta-testing, the tool is available only to a select few.

To perform the task of (closed) beta-testing in as meaningful a way as possible, a decision was made to compare the output of the new tool with the output of two, free to use, existing tools, namely the reconstructed ARM tool [6,13] and TIGER Pro 2.0 [8]. Other tools that could have been made good use of in the comparison included SREE [16,17] and QuARS [5,14]. Unfortunately, it was not possible to obtain working copies of SREE or QuARS. Also excluded from the comparison were commercial tools advertised as requirements *management* tools [9]. Some of these tools might well offer requirements analysis in addition to requirements management.

For input, two requirements specifications were selected, namely WARC and New Adelaide Airport. These requirements specifications were chosen for convenience: the text of these requirements specifications is publicly available [10, 16]. They were also chosen because they had been used in other studies [13, 16, 17], which allowed for comparison.

2. REQUIREMENTS ANALYSIS TOOLS

2.1 Reconstructed ARM

According to [13], the original Automated Requirement Measurement (ARM) tool was developed by the Software Assurance Technology Center (SATC) at the National Aeronautics and Space Administration (NASA) in the late 1990s. The ARM tool, as recently reconstructed by Carlson and Laplante, is available online [6]. [18] is the original article about the ARM tool.

ARM has several categories of quality indicators: imperatives, directives, continuances, options, weak phrases, size, and readability. Only the categories of options and weak phrases unequivocally

represent lack of quality and so output from these categories was analyzed. The following indicator words comprise the options category: can, may, and optionally. The following indicator phrases comprise the weak phrases category: adequate, as appropriate, be able to, be capable of, capability to, effective, as required, normal, provide for, timely, and easy to. Also included in the analysis were reports of the word `should` in the imperatives category as its use in a requirements specification is also unequivocally regarded as poor practice as it can suggest a desire rather than a demand.

In [13], the reconstructed ARM tool was used to analyze and report on summary statistics for the WARC requirements specification. No determination, however, was made of the rate of false positives for the options and weak phrases categories. Also, only an aggregate count for the imperatives category was made, i.e. there was no individual count regarding use of the word `should`.

2.2 TIGER Pro 2.0

TIGER Pro 2.0 runs under Windows and can be downloaded from [8]. TIGER Pro 2.0 has several categories of quality indicators: defect type 1 indicating multiple requirements in a line, defect type 2 indicating possible multiple requirements, defect type 3 indicating a not verifiable word, defect type 4 indicating use of wrong word, defect type 5 indicating a user defined poor word, defect type 6 indicating a possible design requirement, and defect type 7 indicating an incomplete sentence.

The *default settings* of TIGER Pro 2.0 were used. Additional poor phrases were not added to the existing poor phrases list (i.e. no defects of type 5 appear in the analyses of the two requirements specifications). For defect type 6, TIGER Pro 2.0 reported on the use of `via`, and, for defect type 7, TIGER Pro 2.0 reported on any missing `shall`. There are 77 phrases listed in TIGER Pro 2.0's poor phrase list, considerably more than ARM. The list, however, does not contain phrases from the options category of ARM. Only `adequate` and `effective` are matching phrases with ARM's weak phrase category. In other words, there is not a great deal of overlap between the poor phrase lists of these two tools.

TIGER Pro 2.0 also has an agent called BADGER that makes use of a requirement builder template. This feature was not examined in the work reported here.

2.3 SREE

Among non-commercial tools based on the poor phrase list approach to requirements analysis, SREE [16, 17] is one of the best. SREE has ten categories of quality indicators (called subcorpora): continuance, coordinator, directive, incomplete, optional, plural, pronoun, quantifier, vague, and weak. There are almost 60 phrases in the pronoun category (anyone, anybody, anything, etc.) and over 110 phrases in the vague category (adaptability, additionally, adequate, etc.).

As noted earlier, it was not possible to obtain a working copy of SREE. However, SREE's precision for the New Adelaide Airport requirements specification is known. In [17], the precision is reported as 68%. Also, in [1], it is clear that trying to deal with plurals was responsible for some three-quarters of the false positives. SREE's precision could be greatly improved by configuring the tool not to try to deal with plurals.

2.4 QuARS

QuARS [5, 14] is described as a tool which makes use of both a lexical and syntactical approach to requirements analysis. The MINIPAR parser was used by QuARS for parsing requirements text. As noted earlier, it was not possible to obtain a working copy of QuARS. However, the precision of QuARS for six requirements documents from a real industrial project is known. The data of Table 5 in [14] indicate that the precision varied from 61% to 89%, with an overall average of 76%.

In [14], the conclusion was reached that human inspectors of requirements expressed in natural language cannot be fully replaced by a tool like QuARS. Also, to tackle the false positive problem, tailoring dictionaries was concluded to be important.

2.5 New Requirements Analysis Tool

For reasons of commercial confidentiality, no specific details of the tool that was beta-tested can be provided other than to say the tool is substantial and does not rely solely on a poor phrase list approach.

The default settings of this tool were used. It should be emphasized that the tool has configuration options which were essentially unused due to lack of detailed documentation being available at the time of the beta-testing. Attempts were made trying to exploit these configuration options but were halted as they increased rather than decreased the number of reported issues. The proper use of these configuration options could dramatically reduce the rates of false positives reported here.

3. REQUIREMENTS SPECIFICATIONS

3.1 WARC

The requirements specification for WARC (Web ARChive) was derived from the PDF of the WARC tools SRS downloaded from [10]. Functional and non-functional requirements labeled with a specific SRS number were extracted. As part of this extraction process, a new numbering scheme was applied. In terms of volume, text of the derived requirements occupies some 3 pages. The aim of the WARC tools project is described as the facilitation and promotion of the adoption of the WARC file format for storing web archives. The WARC requirements specification is a software requirements specification. To help the reader gauge the content and style of WARC requirements, three of the individual requirements are given below.

> Libwarc shall provide an API describing (1) the set of data, and (2) the set of operations that can be performed on the data. The data types shall be abstract (abstract data types –ADT), to ensure independence of concrete implementations.
>
> It shall not be possible to use more than one compression schema (including non-compression) within a single WARC file. (i.e. it is not possible to mix compression schemes within a single WARC file).
>
> A command line tool shall be implemented utilising libwarc to check the consistency of WARC-records and their conformance to the WARC ISO standard.

3.2 New Adelaide Airport

The requirements specification for New Adelaide Airport was extracted from the PDF downloaded from [16]. (The original source of the requirements specification is unclear.) As part of this extraction process, a new numbering scheme was applied. In terms of volume, text of the derived requirements occupies some 2 pages. The New Adelaide Airport requirements specification is not a software requirements specification. Rather, it is a high-level system specification of an airport. To help the reader gauge the content and style of New Adelaide Airport requirements, three of the individual requirements are given below.

There shall be an arrival meeting area for international passengers.

The transit time between two departure gates shall not exceed 7 minutes.

Time taken to proceed through Security checkpoint shall be less than 10 minutes.

3.3 Other Requirements Specifications

Requirements specifications available online at NLRP Benchmark [4] were considered, but a decision was made to focus on the requirements specifications studied in [13,16,17]. A further decision was made to use only WARC and New Adelaide Airport requirements specifications due to the expense of manually categorizing comments or warnings as true or false positives. (The new tool generated so many issues for the MCSS requirements specification provided in [16] that it was estimated the evaluation of all the issues would have taken several days.)

4. ANALYSIS

Requirements text was cut and pasted as a whole into the reconstructed ARM tool. Requirements text was imported as a whole into Tiger Pro 2.0 and the new tool. To facilitate the determination and analysis of false positives, a spreadsheet was created with a row for each requirement and columns containing copies of the raised issues from the three tools. This was done separately for the WARC and New Adelaide Airport requirements specifications.

In the style of a software inspection [2], a paper copy of each requirements specification was read over by each investigator (inspector) for half an hour with each investigator making notes of issues. Then, several sit-down meetings, each of duration 1-2 hours, took place to review paper copies of the spreadsheets and make a determination as to whether or not a raised comment or warning (hereafter, issue) by a tool was a true or false positive. Determinations were mostly straightforward following a brief discussion. Occasionally, a unanimous view was not possible despite a discussion lasting several minutes. Majority determinations were recorded in such cases.

5. RESULTS

5.1 Precision

Table 1 shows the calculations of precision of the three tools for the WARC requirements specification. Table 2 shows the analysis of precision of the three tools for the New Adelaide Airport requirements specification. The reconstructed ARM tool has good precision but reports so few issues that its usefulness in a modern-day industrial context is questionable.

The new tool being beta-tested reported many more true positives than TIGER Pro 2.0, which in turn reported many more true positives than the ARM reconstruction. However, using its *default settings*, the new tool raised many false positives which resulted in unacceptably low values for precision (27% and 41%).

Though the new tool reported many more true positives, the investigators noticed many instances of comments or warnings that appeared to target the same issue. This might have been as a result of multiple detectors arriving at a similar conclusion about a piece of text. Also, it was suspected that some output might have been aggregated reports of individually reported issues. Upwards of a third of the raw data reported in Tables 1 and 2 might not represent distinct issues. Rather than make the observation from the raw data that the new tool detected around five times as many true positives, a more conservative estimation suggests that the new tool detected around three times as many true positives.

Table 1: Tool Precision for WARC

	ARM	TIGER Pro 2.0	New Tool
Issues	16	108	867
True Positives	11	50	230
Precision	69%	46%	27%

Table 2: Tool Precision for New Adelaide Airport

	ARM	TIGER Pro 2.0	New Tool
Issues	2	71	442
True Positives	2	34	183
Precision	100%	48%	41%

A further analysis of issues that were deemed to be false positives revealed a handful of well-populated categories. It is suspected many false positives in these categories could be addressed by making good use of the new tool's configuration options (either by supplying numerous dictionary entries or by turning off a detector that is not a good match for the requirements specification being analyzed). Two of the categories involved what appeared to be code words. When queried, the company advised that these two categories were still under development. Across the categories, around 50 false positives strongly suggested that numbers in the requirements specifications (either the requirement numbering itself or numbers actually within the text of a requirement) were sometimes confusing the parsing performed by the new tool. When queried, the company suggested the use of an alternative, spreadsheet-based import format. This fixed the problem with the processing of requirement numbers, but not the problem with the processing of numbers within the text of requirements.

With development fixes and good use of configuration options, around 300 and 150 false positives could possibly be eliminated for WARC and New Adelaide Airport requirements specifications respectively. Precisions would then improve to 41% and 63% respectively.

5.2 Inspectors' Issues

As noted above, each investigator acted as an inspector and separately recorded issues found during their readings of the WARC and New Adelaide Airport requirements specifications. A sit-down meeting of the investigators was held to consolidate those issues not detected by any of the three tools. Issues were broken down into two categories. The first category comprised spelling, grammar, and style issues that seem likely to be detectable by an experienced editor or state-of-the art checker tool (e.g. StyleWriter [7]). The second category comprises other issues that might easily allude detection by either a checker tool or a requirements analysis tool.

Table 3 shows that the majority of issues were concerned with

Table 3: Issues Raised by Inspectors

requirements	spelling, grammar, & style	other
WARC	13	5
New Adelaide Airport	21	10

spelling, grammar, and style. This was a disappointing finding. Documents with spelling, grammar, and style issues would typically not meet the entry criteria to allow a software inspection process to begin. In retrospect, the WARC and New Adelaide Airport requirements specifications should have been first sanitized before analyzing them using the three requirements analysis tools.

There now follows a description of the issues labeled `other` in Table 3.

For WARC, to maintain consistency with two other occurrences, the following text should read `peer C class` rather than simply `peer class`. In the context of the WARC requirements specification, this defect is low priority as the two other occurrences are spatially close and the reader would not misunderstand the requirement.

> Each peer class shall expose a set...

For WARC, the following sentence of a requirement was deemed unnecessary. It goes without saying that if software is not written correctly then the system being built will likely not work correctly either.

> In order for the memory management system to work properly, the developer must use the libwarc API correctly.

For WARC, the following part of a requirement was deemed non-verifiable due to the phrase `significant properties` being used. What properties are contained? Which of these are regarded as significant?

> ... and finally to characterise the file by extracting and displaying significant properties contained in the file.

For WARC, the phrase `in the spirit of` in the following requirement is deemed a colloquialism. While this defect is low priority, it is usually best to avoid colloquialisms in requirements specifications.

> Command line tools incorporating libwarc shall be atomic - i.e. each tool performing a single function, performing it perfectly, and in the spirit of Unix command lines tools. These may be combined using pipes and redirection and scripting, to create more high level commands.

For WARC, the following sentence of a requirement was deemed unnecessary. The sentence does not express a requirement on the system.

> Developers may then adapt the build configuration files for best performance on their target machines.

For New Adelaide Airport, the following requirement was deemed non-verifiable as the nature of the breakdown is unclear. Is the breakdown by number of flights or volume of passengers?

> The breakdown between domestic and international is 50:50.

For New Adelaide Airport, the following requirement was deemed non-verifiable as the arrival location is not specified. Is the arrival at baggage claim or the arrivals hall after collecting baggage and clearing any customs and immigration?

> The time from landing to arrival shall be less than 90 minutes for domestic and 120 minutes for international passengers.

For New Adelaide Airport, the following requirement was deemed a non-requirement. A feasibility and risk assessment should address the opposition by the business community expressed in this non-requirement, not the requirements specification.

> The business community (potential investors in the project) like the current airport location and have voiced their opposition to building the new airport 100km north of the city.

For New Adelaide Airport, the following requirement was deemed non-verifiable due to the word `aesthetic` being used. Aesthetic judgements are usually regarded as subjective rather than objective.

> The airport shall have aesthetic departure and transit lounges.

For New Adelaide Airport, the following requirement was deemed non-verifiable due to the word `banking` being used. What kind(s) of banking (ATM, counter, currency exchange, etc.) are actually required?

> Passenger amenities - cafeteria, banking, post communications (phone, mobile, and internet) and frequent flyer lounges shall be provided.

For New Adelaide Airport, the following requirement occurred twice. The requirement writer might have used copy and paste instead of cut and paste.

> All buildings shall be in line with the International Building Code (2000).

For New Adelaide Airport, the following is a non-verifiable requirement due to the acronym `UPS` being used without definition. (UPS in this context probably stands for Uninterruptible Power Supply.)

> Power including UPS and lighting.

For New Adelaide Airport, the following requirement was deemed non-verifiable due to the use of the phrase `built-up areas`. Are the built-up areas in question defined by city limits, densities of population, or speed limits?

> The noise level in built-up areas shall meet EPA specifications.

For New Adelaide Airport, the following requirement was deemed non-verifiable due to the use of the expression `10-9`. While it is likely the requirement writer had intended 10.0E-9 or 10^{-9}, the fact that some kind of mistake has been made means there is doubt about the number to be used in the safety risk analysis.

> An aircraft safety risk analysis of take-off and landing infrastructure (covering runway, apron, control tower and taxi design) shall demonstrate a probability of aircraft accident of less than 10-9.

For New Adelaide Airport, the following requirement was deemed non-verifiable due to the word `growth` being used. Does growth refer to hanger, runway, or terminal facilities? What volume of growth is to be catered for and over what period?

> A modular design shall be adopted to facilitate growth.

6. FEEDBACK

As noted earlier, it is expected of beta-testers that they report their findings to the company or group developing the tool. Below is a summary of the feedback given to the company.

- The new tool, using its *default settings*, detected around three times as many true positives as the reconstructed ARM tool and Tiger Pro 2.0 (also using its default settings).
- Upwards of a third of comments or warnings from the new tool, using its *default settings*, might not have represented distinct issues. This might have been as a result of multiple detectors arriving at a similar conclusion about a piece of text. Also, it was suspected that some output might have been aggregated reports of individually reported issues.
- The new tool's precision, using its *default settings*, was measured at 27% (for the WARC requirements specification) and 41% (for the New Adelaide Airport requirements specification).
- To reduce the rate of false positives, detailed documentation or other guidance is needed on how to make best use of configuration options. Attempts at configuration were unsuccessful.
- There is a strong suggestion that numbers within the text of requirements sometimes confused the parsing performed by the new tool.
- With development fixes and good use of configuration options, it is estimated the measured precisions could improve to 41% and 63% respectively, the latter being similar to the reported precision of the SREE tool for the New Adelaide Airport requirements specification.

7. CONCLUSIONS

7.1 Beta-testing

The new tool was perhaps not quite ready for beta-testing. Two categories of comments or warnings were still under development. Also, there is a strong suggestion that numbers within the text of requirements sometimes confused the parsing performed by the new tool.

The investigators were unable to make good use of the new tool's configuration options due to a lack of detailed documentation. In retrospect, the company should have been alerted to this configuration difficulty earlier to provide ample opportunity for detailed documentation or other guidance to be provided.

The lesson learned is that it is important for beta-testers to maintain frequent communication with the company or group developing the software.

7.2 Requirements Analysis Tools

A lesson learned is that spelling, grammar, and style should be corrected before requirements text is input into a requirements analysis tool. Some false positives found during beta-testing might well have arisen because the requirements text had not been sanitized beforehand.

Another lesson learned is that human inspectors of requirements expressed in natural language cannot be fully replaced by tools. This is in agreement with the conclusions drawn in [14].

8. ACKNOWLEDGMENTS

The authors would like to thank the support of the Department of Computer Science and the Swenson College of Science and Engineering at the University of Minnesota Duluth, and the Alworth endowment fund. The authors would also like to thank the company involved for providing access to their tool and allowing the authors to act as beta-testers.

9. REFERENCES

[1] Computer hope online dictionary. `http://www.computerhope.com/jargon/b/beta.htm`. Accessed July 16 2014.

[2] Guide to the Software Engineering Body of Knowledge (SWEBOK), CHAPTER 11 SOFTWARE QUALITY. `http://www.computer.org/portal/web/swebok/html/ch11`. Accessed July 16 2014.

[3] Guide to the Software Engineering Body of Knowledge (SWEBOK), CHAPTER 5 SOFTWARE TESTING. `http://www.computer.org/portal/web/swebok/html/ch5`. Accessed July 16 2014.

[4] The nlrp benchmark homepage. `http://nlrp.ipd.kit.edu/index.php/Main_Page`. Accessed July 16 2014.

[5] QuARS Quality Analyzer for Requirement Specifications Version 4.1. `http://quars.isti.cnr.it/index.html`. Accessed July 16 2014.

[6] Reconstructed ARM tool. `http://www.kuwatche.com/ARM.html`. Accessed July 16 2014.

[7] Stylewriter version 4. `http://www.stylewriter-usa.com/`. Accessed July 16 2014.

[8] Tiger Pro:- Tool to Ingest and Elucidate Requirements. `http://www.therightrequirement.com/TigerPro/TigerPro.html`. Accessed July 16 2014.

[9] Volere's list of requirements management tools. `http://www.volere.co.uk/tools.htm`. Accessed July 16 2014.

[10] warc-tools, Legacy code for handling ISO WARC files. `https://code.google.com/p/warc-tools/`. Accessed July 16 2014.

[11] D. M. Berry, E. Kamsties, and M. M. Krieger. From contract drafting to software specification: Linguistic sources of ambiguity. `https://cs.uwaterloo.ca/~dberry/`

handbook/ambiguityHandbook.pdf, 2003. Accessed July 16 2014.

[12] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler. A few billion lines of code later: Using static analysis to find bugs in the real world. *Commun. ACM*, 53(2):66–75, Feb. 2010.

[13] N. Carlson and P. Laplante. The NASA automated requirements measurement tool: a reconstruction. *Innovations in Systems and Software Engineering*, 10(2):77–91, 2014.

[14] G. Lami and R. W. Ferguson. An empirical study on the impact of automation on the requirements analysis process. *Journal of Computer Science and Technology*, 22(3):338–347, 2007.

[15] L. Mich, M. Franch, and P. Novi Inverardi. Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(2):151–151, 2004.

[16] S. F. Tjong. *Natural language patterns for requirements specifications*. PhD thesis, Faculty of Engineering & Computer Science, University of Nottingham, Malaysia Campus, 2006. https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/TjongTR-02_2006.pdf. Accessed July 16 2014.

[17] S. F. Tjong and D. M. Berry. The Design of SREE: A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. In *Proceedings of the 19th International Conference on Requirements Engineering: Foundation for Software Quality*, REFSQ'13, pages 80–95, Berlin, Heidelberg, 2013. Springer-Verlag.

[18] W. M. Wilson, L. H. Rosenberg, and L. E. Hyatt. Automated analysis of requirement specifications. In *Proceedings of the 19th International Conference on Software Engineering*, ICSE '97, pages 161–171, New York, NY, USA, 1997. ACM.