Assessing Tools for Defect Detection in Natural Language Requirements: Recall vs Precision

Daniel M. Berry
University of Waterloo
Waterloo, ON, Canada
dberry@uwaterloo.ca

Alessio Ferrari
ISTI-CNR
Pisa, Italy
alessio.ferrari@isti.cnr.it

Stefania Gnesi
ISTI-CNR
Pisa, Italy
stefania.gnesi@isti.cnr.it

***Abstract*—This extended abstract discusses the tradeoffs between recall and precision in assessing tools for finding defects in natural language requirements specifications.**

I. NLP Tools for Defect Detection

In the late 1980s, some authors, e.g., Fickas [1], began working on problems related to the use of natural language (NL) in requirements engineering (RE) and on using NL processing (NLP) to automate RE processes involving NL. In 1993, Ryan observed that NLP "does not now, nor will it in the foreseeable future, provide a level of understanding [of NL requirements] that could be relied upon, ... it is highly questionable that the resulting [NLP] system would be of great use in requirements engineering." [2]. Nevertheless, Ryan saw that NLP could be useful for a variety of *specific* RE tasks involving NL requirements specifications (RSs), such as tracing and entity identification.

Thus, since then, NLP has been applied to a variety of such tasks, including requirements tracing [3], [4], requirement classification [5], app review analysis [6], model synthesis [7], RS ambiguity finding [8], and its generalization, RS defect finding [9]. The rest of this short paper is couched in terms of one representative task, RS defect finding, which has received attention from the early 2000s [9] through to now [10], including from these authors. Nevertheless, much of the discussion is applicable to the other tasks.

The RE field has often adopted Information Retrieval (IR) algorithms for use in developing tools for finding defects in NL RSs [11]. Quite naturally, the RE field has adopted also IR's measures, i.e., precision, recall, and the F-measure, to assess the effectiveness of these tools. *Precision*, P, is the probability that a tool-reported possible defect is in fact a defect; *recall*, R, is the probability that a defect is *reported* by the tool; the F-measure, is the harmonic mean of P and R. There is a weighted version of the F-measure, F_β which weights R by β^2 in the harmonic mean formula.

Precision is negatively influenced by the number of defects wrongly identified, also called *false positives*. Recall is negatively influenced by the number of undetected defects, also called *false negatives*. Precision and recall can usually be traded off in an IR algorithm: when you increase recall, you tend to decreas precision, and *vice versa*.

Historically, IR, such as for search engines, has valued precision more than recall. For example, when you are searching for an Italian restaurant near you, all you require is one true positive for a recall of $\frac{1}{n}$, where n is the possibly large number of correct answers the search engine could give you. However, you are very annoyed with low precision, when you have to wade through many false positives to get to the one true positive that you seek.

However, the situation in RE for finding defects in NL RSs is different [12]. On a small scale, deciding whether a particular RS sentence has a defect is easy. However, in the context of the typical large collection of large NL RS documents accompanying the development of a computer-based-system (CBS), the task becomes unmanageable. So we are motivated to develop a tool to assist in the task, particularly in cases, such as for safety-critical CBSs, in which *all* defects must be found. On this basis, it seems clear that for such a tool, recall is going to be more important than precision, and that the value of β to use in F_β will be greater than 1. However, the annoyance factor of having to wade through the many false positives arising from low precision is real, and it could totally discourage users from using the tool.

This extended abstract adopts the Hegelian structure of thesis, antithesis, and synthesis in describing how a tool for finding defects in an RS should be assessed. The thesis, presented in Section II is that recall is more important than precision. The antithesis, presented in Section III is that precision is more important than recall. The synthesis, presented in Section IV, goes beyond the precision–recall dispute, and suggests an empirically sound way to evaluate any given tool.

II. Thesis: Arguments in Favor of Recall

The first argument that recall is more important than precision for the defect finding task is based on the costs of the task. Manually finding a true positive, an RS sentence with a defect, in the large collection of large input documents takes significantly more time than manually rejecting a false positive, an RS sentence with no defect after all, among the RS sentences that the tool claims to have defects. Furthermore, if the tool is not able to achieve 100% recall, or at least better recall than is humanly possible, the tool is effectively useless. The user will be compelled to search for defects manually in the entire set of inputs, because he or she cannot determine

from the tool's output what parts of the input do not have to be examined.

The second argument is based on the costs of undetected defects. Particularly in the context of a safety-critical CBS, the cost of an undetected, false negative defect is much higher than the cost of an incorrectly identified, false positive defect. Very high recall is required to avoid an undetected defect that can lead to a life-threatening run-time error.

III. Antithesis: Arguments in Favor of Precision

The first argument that precision is more important than recall for the defect finding task is based on a paradox. A tool that gives 100% recall could be the tool that simply reports *every* input sentence as having a defect. This tool saves the user no work at all, because he or she will have to inspect the entire input manually to weed out all the false positives. Therefore, recall cannot be the only measure to assess the performance of a defect finding tool, and precision must be taken into account.

The second argument concerns the perception that users have of an automated tool. An automated tool is not expected to fail, and, if it fails too often, the user will cease to use it. A false positive defect — an RS sentence reported as defective, but actually not defective, as the user can plainly see — is regarded as a failure by the user and as a reason to distrust the tool. It is therefore important that a tool give some value to precision.

IV. Synthesis: A New Method to Evaluate Tools

Notwithstanding the need to favor recall over precision, the thesis and antithesis show that that both measures need to be considered when assessing a defect-finding tool. Thus, the question is "What are the measures' relative importance, i.e., what is the correct value of β to use in F_β?" Nevertheless, we have actually ignored the underlying question, "How can we know whether it is better to use a defect finding tool, or to find the defects entirely manually?" The answers to both of these questions can come only empirically, e.g., by adapting IR's cost-based evaluation measures [13] to the RE context.

For determining the relative importance and thus the value of β, we will need to determine empirically [12] :

1) the average time that an average human needs to manually find a defective RS sentence among all RS sentences, and
2) the average time that an average human needs to manually vet any RS sentence marked as defective by the tool, which is less than or equal to the average time that an average human needs to manually determine whether or not any RS sentence is defective.

The value of β is the ratio of the first to the second.

For determining whether to find defects manually, with a tool, or with some mixture of both, we need to determine empirically [12]:

- the criticality of achieving 100% recall on the task, by estimating the cost of an undetected defect,
- the criticality of high precision, by estimating the tool-use deterrence created by each false positive reported by the tool,
- the average recall that the tool achieves on the task, and
- the average recall that humans achieve when they do the task manually.

Some of all of these data can be computed by keeping track of (1) time and (2) individual results (1) during a multi-person construction of a gold standard with which to evaluate any tool constructed to find RS defects and (2) during the test of the tool at hand against the gold standard. These data allow us to answer the second, underlying question.

V. Conclusion

This paper has briefly reviewed the use of NLP in RE. It has focused on the task of finding defects in RSs in order to discuss the use of recall and precision to assess tools developed for NLP in RE. The conclusion is that for any NL RE task, empirical studies are needed to allow recall and precision to be valued correctly for the task. We hope that this paper will trigger deeper discussions of way the RE field assesses its NLP tools.

References

[1] S. Fickas, "Automating the analysis process," in *International Workshop on Software Specification and Design (IWSSD)*, 1987, pp. 58–67.

[2] K. Ryan, "The role of natural language in requirements engineering," in *International Symposium on Requirements Engineering (ISRE)*, 1993, pp. 240–242.

[3] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, and G. Antoniol, "The quest for ubiquity: A roadmap for software and systems traceability research," in *International Requirements Engineering Conference (RE)*, 2012, pp. 71–80.

[4] V. Gervasi and D. Zowghi, "Supporting traceability through affinity mining," in *International Requirements Engineering Conference (RE)*, 2014, pp. 143–152.

[5] A. Casamayor, D. Godoy, and M. Campo, "Functional grouping of natural language requirements for assistance in architectural software design," *Knowledge-Based Systems*, vol. 30, pp. 78–86, 2012.

[6] W. Maalej, Z. Kurtanović, H. Nabil, and C. Stanik, "On the automatic classification of app reviews," *Requirements Engineering*, vol. 21, no. 3, pp. 311–331, 2016.

[7] M. Robeer, G. Lucassen, J. M. E. van der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via NLP," in *International Requirements Engineering Conference (RE)*, 2016, pp. 196–205.

[8] S. F. Tjong and D. M. Berry, "The design of SREE: a prototype potential ambiguity finder for requirements specifications and lessons learned," in *Requirements Engineering: Foundations of Software Quality (REFSQ)*, 2013, pp. 80–95.

[9] F. Fabbrini, M. Fusani, S. Gnesi, and G. Lami, "An automatic quality evaluation for natural language requirements," in *Requirements Engineering: Foundations of Software Quality (REFSQ)*, 2001, pp. 1–18.

[10] H. Femmer, D. M. Fernández, S. Wagner, and S. Eder, "Rapid quality assurance with requirements smells," *Journal of Systems and Software*, vol. 123, pp. 190–213, 2017.

[11] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 71–98.

[12] D. Berry, "Evaluation of tools for hairy requirements engineering and software engineering tasks," School of Computer Science, University of Waterloo, Tech. Rep., 2017, https://cs.uwaterloo.ca/~dberry/FTP_SITE/tech.reports/EvalPaper.pdf.

[13] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.