Requirements for Tools for Hairy Requirements or Software Engineering Tasks

Daniel M. Berry
Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada
dberry@uwaterloo.ca

***Abstract*—A *hairy* requirements or software engineering task involving natural language (NL) documents is one that is not inherently difficult for NL understanding humans on a small scale but becomes unmanageable in the large scale. A hairy task demands tool assistance. Because humans need far more help in carrying out a hairy task completely than they do in making the local yes-or-no decisions, a tool for a hairy task should have as close to 100% recall (that the tool finds *all* desired information) as possible, even at the expense of high imprecision (that not all the information that the tool finds is desired). A tool that falls short of 100% recall may even be useless, because to find the missing information, a human has to do the entire task manually anyway.**

Any such tool based on NL processing (NLP) techniques inherently fails to achieve 100% recall, because even the best parsers are no more than 91% correct. Therefore, to achieve 100% recall in a tool for a hairy task, it needs to be based on something other than traditional NLP.

The reality is that a tool's achieving exactly 100% recall, which may be impossible anyway, may not be necessary. It suffices for a human working with the tool on a task to achieve better recall than a human working on the task entirely manually.

This paper describes research whose goal is to discover and test a variety of non-traditional approaches to building tools for hairy tasks to see which, if any, allows a human working with with the tool to achieve better recall than a human working entirely manually.

***Index Terms*—abstraction finding, ambiguity finding, F-measure, false negatives, false positives, hairy task, manual task, natural language documents, precision, recall, tool-assisted task, tracing**

I. Introduction

A *hairy* requirements or software engineering (RE or SE) task involving natural language (NL) documents is one that requires NL understanding and is not difficult for humans on a small scale but is unmanageable when it needs to be done to the documents or artifacts that accompany the development of large software systems. [1]. Examples of hairy tasks include finding abstractions, ambiguities, or trace links [2].

Humans understand NL well enough that a human has the potential of achieving for the hairy task 100% *correctness*, i.e., finding all and only the desired information. While a hairy task itself is generally not overly complex, the sheer size of the documents for a real-life software development makes the task burdensome enough that tool assistance is needed for a human to do complete job [3]. Such a tool is a *tool for a hairy task*.

The rest of this paper assumes that the tool at hand is being used to carry out a hairy task for the development of a large real-life system [1].

II. Limitations of Tools for Hairy Tasks

The typical tool for a hairy task is built using NL processing (NLP), i.e., with parsers and parts-of-speech taggers [2], [3]. Even the best parsers are no more than 85–91% accurate [4]. No NLP-based tool can be better than its parser. Thus, when faced with the inherently ambiguous and anomalous NL text that is found in real-life software development documents, no NLP-based tool will achieve more than 85–91% correctness. This limitation is termed the *fundamental limitation of NLP-based tools for hairy tasks*.

III. Recall and Precision to Evaluate Tools

The measures by which a tool for a hairy task is evaluated are *recall* (the percentage of the correct answers that the tool finds) and *precision* (the percentage of the found answers that are correct). Thus, "correctness" in Section I, capturing *all and only* the desired information, is a composite of recall and precision. Often, the F-measure,

$$F = 2 \times \frac{P \times R}{P + R}, \tag{1}$$

the harmonic mean of recall, R, and precision, P, is used as a single summary measure. It captures the notion of correctness in Section I.

Because an NLP-based tool's finding a correct answer in NL text depends on parsing the text correctly, the fundamental limitation of NLP-based tools for hairy tasks means that an NLP-based tool for a hairy task cannot achieve better than 85–91% recall.

IV. Recall vs. Precision

Many a tool for a hairy task is reported in the literature as having more precision than recall [5]. Sometimes, that a tool has a precision of greater than 85% while having a recall of as low as 65% is reported as satisfactory [6]. (See Section IX for more details.) However, for a typical hairy task, finding a missing correct answer (a false negative) is at least an order of magnitude harder than rejecting as nonsense an incorrect answer (a false positive). Finding a missing correct answer generally requires examining all the input documents in detail,

while rejecting an incorrect answer generally requires understanding only the incorrect answer and the input documents at only a general level. For example, Kong, Huffman Hayes, Dekhtyar, and Holden show that humans are much better at validating tool-proposed trace links than they are at finding links in documents [7]. In this author's experience, for a typical hairy task, recall is at least an order of magnitude more difficult than precision and should be weighted accordingly.

There are weighted variations of the F-measure,

$$F_\beta \;=\; (1+\beta^2) \times \frac{P \times R}{(\beta^2 \times P) + R} \quad , \tag{2}$$

called the F_β-measure. Here, β is the ratio by which it is desired to weight R more than P. Therefore, for a hairy task in which finding a true positive requires n times the time required to reject a false positive, β should be n. Note that the simple F-measure is F_1. Note also that the formula for F_β ends up multiplying P by the dominating β^2 in both the numerator and the denominator. Thus, as n grows, very quickly F_β approaches R, and P becomes irrelevant.

Why is there such an emphasis on precision? Precision is important in the information retrieval area from which are borrowed many of the algorithms used to construct the tools for hairy tasks [3], [8]. In information retrieval, users of a tool with low precision are turned off by having to reject false positives more often than they accept true positives. In some cases, only a few or even only one true positive is needed. Perhaps the force of habit drives people to evaluate the tools for hairy tasks with the same criteria that are used for information retrieval tools. Also, "precision" sounds so much more important than "recall", as in "This output is precisely right!".

V. Requirements for 100% Recall

When a hairy task is applied to the development of software with high dependability requirements, 100% recall is *essential*. If a tool for the task achieves less than 100% recall, then the task must be done manually on the whole of the documents to find the answers that the tool does not deliver. Therefore, in the last analysis, such a tool is really useless[1].

VI. True Recall Goal for a Tool for a Hairy Task

While 100% recall for a hairy task is nice in theory, there are several problems that get in the way of achieving 100% recall for a tool for the hairy task:

1) Achieving 100% recall is probably impossible, even for a human, as is finding all bugs in a program.
2) Even if achieving 100% recall were possible, we have no way to know if we have succeeded, because the only way to measure recall for a tool for is to compare the output of the tool against totally correct output, which can be made only by humans, who may miss some, for the very reason that a tool is needed. It is like knowing that the last bug has been found.

Let us call what humans can achieve when performing the task manually under the best of conditions the "humanly achievable high recall (HAHR)". So, all that is really necessary is to show that the tool for the task measurably achieves better recall than the HAHR. If this achievement can be demonstrated, then a human will trust the tool and will not feel compelled to do the tool's task manually to look for what the human feels that the tool failed to find. Thus a tool for a hairy task must be evaluated by comparing the recall of humans working with the hairy tool with the recall of humans carrying out the task manually [9], [10].

VII. New Approach Needed for Tools

Since an NLP-based tool cannot achieve better than 85–91% recall, perhaps it is time to try other approaches to design a tool for a hairy task:

1) One possible approach is to partition the task into two parts [11]:
   a) a clerical part that can be done algorithmically with 100% recall
   b) a hard part that must be done manually.

   The hope is that the remaining manual task is significantly easier than the original or that it can be done on significantly reduced documents. The tool and people cooperate to achieve higher overall recall than is possible for either working alone [12].
2) Another possible approach is to build a tool for the inverse task, which returns for removal, parts of the input that cannot possibly be relevant. The hope is that the inverse tool has 100% precision (so that only irrelevant stuff is removed) and that what is left for manual processing is significantly smaller than the original [11], [13], [14].

Part of the research is to identify other approaches.

VIII. Recent and not so Recent Progress

Leading up to this research were several discoveries often made in the process of doing other work.

Goldin built AbstFinder, a tool to help identify abstractions in NL text for use in the initial stages of RE [9]. The two lessons learned from this work that impact the proposed research are:

1) As Goldin had worked in signal processing in the past, she used, not an NLP approach, but a signal processing approach and treated each sentence, not as a sequence of space-character-separated words, but as a sequence of characters, some of which happen to be spaces. She proved, in effect, that an NLP approach was not absolutely essential for such a tool.
2) Goldin evaluated AbstFinder by comparing her operating AbstFinder for 8 hours on an industrial RFP she

[1]Of course, one can argue that such a tool is useful as a defense against a human's less-than-100% recall when the tool is run as a double check after the human has done the tool's task manually. However, it seems to this author, that if the human *knows* that the tool will be run, he or she might be lazy in carrying out the manual task and not do as well as possible. Empirical studies are needed to see if this effect is is real, and if so, how destructive it is of the human's recall.

had not seen before to three domain expert requirements analysts' (RAs') examining the same RFP over one month. She found all abstractions that the three RAs had found and then some; i.e., her recall in 8 person hours was better than that of the three domain expert RAs in 3 person months, about 65 times longer. Of course, there is no way to know the absolute recall.

Tjong built SREE, a tool to help find ambiguities in NL text, for use on requirements specifications in the analysis stages of RE [11]. The three lessons learned from this work that impact this proposed research are:

1) Tjong divided the ambiguity identification task into two parts:
   a) a clerical part that can be done algorithmically with 100% recall, but probably with a lot less than 100% precision, with SREE, a lexical-analysis-based tool. SREE finds 100% of all ambiguities that cannot be present without the occurrence of specific keywords, e.g., the "only" ambiguity can occur only when the word "only" is used. A tool that finds every sentence with "only" finds 100% of the "only" ambiguities, but has high imprecision, because not every sentence with "only" suffers the "only" ambiguity. These kinds of ambiguities are said to be *in SREE's scope*.
   b) a hard part that must be done manually by a human RA who is attuned to spotting ambiguities in NL text. The RA is left to find manually all instance of only those ambiguities, e.g., the plural ambiguity, that are not in the scope of the lexical-analysis-based tool.
2) She realized that manually rejecting as a false positive a tool-found potential ambiguity — necessarily in SREE's scope — that is not actually ambiguous is an order of magnitude faster than manually finding any true positive ambiguity outside SREE's scope. Moreover, a tool-found potential ambiguity can be manually rejected as a false positive with more certainty than can a true positive ambiguity be manually found. Therefore, if a tool for a hairy task has 100% recall, it is OK that it has also low precision.
3) Her method to evaluate SREE was to to do a post-hoc analysis of how a SREE-assisted search for ambiguities in an industrial SRS (software requirements specification) compares with a totally manual search for ambiguities in the same SRS. She determined that the time to use SREE to find ambiguities in its scope, including rejecting false positives plus the time to manually find the remaining ambiguities outside SREE's scope is less than the time to manually find all ambiguities.

Berry, Gacitua, Sawyer, and Tjong recognized the gist of the argument (but using a slightly different vocabulary) given in Sections I–VII of this paper and asked that RE or SE tool developers report to the authors any tools that they thought met the goals stated in the publication [3]. The last paragraph of Section IX lists these reported tools.

Lan empirically compared the recall of six WordNet-based and two context-based algorithms for finding synonyms in requirements documents to determine that word co-occurrence-based methods came the closest, but not all the way, to achieving 100% recall [15].

IX. Literature Review and Ideas for Approaches

Some of the relevant literature is that cited in Sections I–VIII.

A. Tracing and other RE Tools

Many developers of tracing tools *have* recognized that recall is more important for a tracing tool than precision [16]–[20]. In fact, Cleland-Huang, Czauderna, Gibiec, and Emenecker, who use a machine-learning rather than an NLP approach to tracing, have stated explicitly that recall is more important than precision for tracing by their use of F_2 rather than F_1 for evaluation of their approach. However, there are still some developers of tracing tools and similar tools who appear to regard precision as equally or more important than recall, at least part of the time, or that use an F-measure that equates precision and recall [6], [10], [21]. In addition, developers of tools for other hairy tasks such as a NL abstraction finder, a NL requirements finder, a NL requirements categorizer, a NL sentiments finder, a NL term finder appear to be favoring precision over recall, use an F-measure that equates precision and recall, or describe a recall less than 95% as acceptable [5], [22]–[30]. Gleich, Creighton, and Kof say that their goal was 100% recall, but they accepted much lower recall as satisfactory, arguing correctly that humans do about the same [31].

Delater and Paech propose and implement in a prototype tool a new tracing approach in which data from a developer's current task are mined to build a link between the requirements and the code involved in the task [32]. An empirical evaluation of the approach and the tool achieves 90–94% recall compared to human-determined gold links, not quite the 100% recall that they had aimed for. Hübner, also working with Paech, is building on the work of Delater and Paech. He proposes two additional sources for link-building data:

1) to link a pair of specific artifacts, existing links in the context of the artifacts are examined, and
2) to discover artifacts that are likely to need a link between them, each developer's interaction logs are examined to discover the artifacts that he or she touched during a task [33].

Moreover, he suggests that links be proposed to developers as they are doing a task, for immediate vetting. Immediate vetting improves both recall and precision, as vetting occurs right when the task suggesting the link is being done. Also, it makes the links available for use during the development by the very developers who are causing them, thus increasing the motivation of the developers to do the link vetting immediately [34]. Hübner is only in the early stages of his research and

has not yet evaluated the recall and precision of his proposed tool. However, the approach he is taking is new, and it looks promising for improving on Delater and Paech's 90–94% recall.

B. Tools used by Humans

Cuddeback, Dekhtyar, Hayes [35] have developed several tools for the hairy task of identifying trace links between utterances in the set of artifacts, many of which are NL documents, for the development of a software system. Typically, such a tool uses NLP methods to find different utterances in one or the set of artifacts that appear to be talking about the same thing. For each such pair of utterances, a human RA is shown the utterances and is asked to vet the pair as being worthy of a link. Each such tool has a recall, the percentage of all links that it finds, and a precision, the percentage of the pairs it found that were vetted to be links. The precision of a typical of these tools is higher than its recall, which is contrary to the goal of the HAHR.

Cuddeback *et al.* have recently begun to consider the role of the user of these tools and how the user's perceptions, knowledge, and behavior affect recall and precision. When comparing two tools whose recall differs, they have noticed that the RAs vetting the poorer of the two tools did a better job and achieved an overall higher recall than the RAs vetting the better of the tools. It is as though the RAs using the poorer tool can sense the poor quality of the tool and rise to the occasion. So perhaps a good approach would be to run a high-recall tool to get a list of proposed links, divide the list randomly into two equal-sized lists, that will necessarily look as though they came from a low-recall tool, let one group of RAs vet one list, let another group vet the other list, and then combine their vetted links. This method is an attempt to get the best of both worlds.

Along the same line, Zeni, Kiyavitskaya, Mich, Cordy, and Mylopoulos [10] have shown in an experiment that a high-precision, low-recall tool for annotating laws helps novices achieve 96% recall relative to legal experts' so-called gold annotations, which establishes, in essence, the HAHR. Perhaps the high precision helped the novices learn what is right, so that each could use his or her intelligence correctly to achieve high recall.

More and more builders of tools for hairy tasks are evaluating their tools by comparing humans working with the tool on artifacts with humans working manually on the same artifacts [26], [36], [37].

C. Nocuous Ambiguity Finding

Yang, De Roeck, Gervasi, Willis, and Nuseibeh experiment with a machine-learning (ML) approach to recognizing nocuous[2] anaphoric ambiguity [38]. They had as one of their goals the achievement of 100% recall for this kind of ambiguity, "even at the expense of some imprecision" [38]. Thus, their algorithm is designed to maximize recognition of nocuous instances of anaphoric ambiguity even at the cost of delivering more innocuous instances. In addition, to evaluate their approach, they use F_2, in which recall is weighted twice what precision is, as their F_β measure. Their ML algorithm achieves 99.37% recall, very close to the 100% goal, while paying 82.01% precision. Their algorithm achieves this recall in part by by having two thresholds, playing off against each other, rather than only one. Perhaps, ML should be considered as basis for constructing tools with the HAHR for hairy tasks. Experience with ML has shown that it is very successful for some tasks and very unsuccessful for others [39].

[2] "Nocuous" is the opposite of "innocuous".

D. Finding Privacy Violating Disclosures

Finding unauthorized disclosures of private information in the code of mobile applications, i.e., in apps, is a hairy task. Moreover, for someone whose job is to find and prevent all such disclosures, this task must be done with 100% recall. The traditional approach, taint analysis, searches the code for an app for transmissions to the outside world. If a transmission point is transmitting information derived from internal information that is known to be privacy sensitive, the transmission is flagged as illegitimate.

Tripp and Rubin observe that taint analysis yields false positives since a derivative of sensitive information is not necessarily sensitive itself [40]. Moreover, release of sensitive information is often essential for an app's functionality [41]. The result of this observation is that taint analysis yields many false positives that require manual inspection and thus has low precision. In addition, taint analysis may not have good recall, having achieved only 51% when applied to some real-life apps. Tripp and Rubin use the F-measure, combining recall and precision, as the measure of accuracy; thus, taint analysis has low accuracy. Finally, taint analysis is expensive to run. Thus, over all, Tripp and Rubin consider taint analysis not to be very useful.

To solve this low-accuracy problem, Tripp and Rubin propose and test a Bayesian (machine-learning) approach to learn which transmissions are sending truly sensitive information. This approach appears to be faster than taint analysis and ends up achieving high recall and high precision, and thus high accuracy. When they compared taint analysis with their Bayesian approach on the 65 most popular apps from Google Play, they found that

- taint analysis achieved 100% precision and 51% recall, and thus an accuracy of 67%, and
- the Bayesian approach achieved 96% precision and 100% recall, and thus an accuracy of 98%,

a significant improvement in recall and accuracy with a minor drop in precision.

However, this author believes that manually finding a true positive sensitive transmission takes significantly more time than manually inspecting and rejecting a false positive. His guess is that manually finding a true positive requires about 5 times the time that manually rejecting a false positive. If this guess is true, then the F_β-measure that should be used to evaluate the taint analysis and the Bayesian approach is F_5,

which is very close to the recall value. With F_5, the Bayesian approach fares even better than taint analysis than with F_1.

E. Reports of Tools with Close to 100% Recall

As a result of the publication by Berry, Gacitua, Sawyer, and Tjong [3], people e.g., Rene Meis [13] and Garm Lucassen [42], have begun reporting to the author that work that they had done is what the publication is calling for.

X. Research Objective

The long-term research objective is to find and validate the effectiveness of new approaches for building tools for a variety of hairy RE or SE tasks, tools that achieve better than the HAHR.

The near-term objective is to build a working prototype tool for *one* hairy RE or SE task, identified by the research, and to empirically validate that the tool achieves better than the HAHR. It will be necessary to show that humans using the tool to do the task achieve a higher overall recall than do humans doing the task completely manually.

Achieving the near-term objective is a reasonable first step for achieving the long-term objective.

XI. Research Plan

The research plan consists of four steps.

1) A systematic literature survey will be conducted to find
   a) recall-tested tools for hairy RE or SE tasks and
   b) alternative tool and task architectures for hairy RE or SE tasks that have the potential of achieving the recall objectives.

   Section IX of this paper is a very preliminary pilot for the systematic survey.
2) Based on what is learned in that survey, a hairy RE or SE task will be chosen for further investigation, namely that judged most likely to be doable with a tool with better than the HAHR.
3) Then, prototype tools will be developed for the identified hairy task. These prototypes will need to be adjusted until they meet the recall requirement.
4) Finally, humans conducting the task manually will be compared empirically with humans conducting the same task with the help of the prototype tools.

XII. Conclusions

This paper has described the concept of hairy RE or SE tasks and has argued that tools for hairy tasks need to have as close to 100% recall as is humanly possible, what is called the HAHR. It has observed that an NLP-based tool, being based on an imperfect parser, can never achieve better recall than its parser's accuracy, which is at best 85–91%. Therefore, to achieve the HAHR for a tool for a hairy task, we need to consider other bases for the tool. The paper describes a research program to find these bases.

If the research is successful, a number of benefits will arise:

1) We will have available new methods with which to build tools for hairy RE or SE tasks that provide greater recall than current tools and which, when used properly, help the human user to do a demonstrably better job at a hairy RE or SE task than he or she can by doing the task manually.
2) Researchers building tools for hairy RE or SE tasks will be more likely to evaluate their tools with the more relevant recall metric than with the less relevant precision metric.
3) RE or SE practitioners will have better tools to use in their daily work, which consists of many hairy tasks.

Acknowledgment

Daniel Berry's work was supported in part by a Canadian NSERC grant NSERC-RGPIN227055-00.

References

[1] L. Northrop, B. Pollak, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidt, K. Sullivan, and K. Wallnau, *Ultra-Large-Scale Systems, The Software Challenge of the Future*. Pittsburg, PA, USA: Software Engineering Institute at Carnegie Mellon University, 2006. [Online]. Available: http://www.sei.cmu.edu/library/assets/ULS_Book20062.pdf

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

[3] D. M. Berry, R. Gacitua, P. Sawyer, and S. F. Tjong, "The case for dumb requirements engineering tools," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2012, pp. 211–217.

[4] R. Socher, J. Bauer, C. D. Manning, and A. Y. Ng, "Parsing with compositional vector grammars," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, vol. 1: Long Papers, 2013, pp. 455–465.

[5] A. Dwarakanath, R. R. Ramnani, and S. Sengupta, "Automatic extraction of glossary terms from natural language requirements," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2013, pp. 314–319.

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

[7] W. Kong, J. H. Hayes, A. Dekhtyar, and J. Holden, "How do we trace requirements: An initial study of analyst behavior in trace validation tasks," in *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 2011, pp. 32–39.

[8] 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.

[9] L. Goldin and D. M. Berry, "AbstFinder: A prototype abstraction finder for natural language text for use in requirements elicitation," *Automated Software Engineering*, vol. 4, pp. 375–412, 1997.

[10] N. Zeni, N. Kiyavitskaya, L. Mich, J. R. Cordy, and J. Mylopoulos, "Gaiust: Supporting the extraction of rights and obligations for regulatory compliance," *Requirements Engineering Journal*, vol. 20, no. 1, pp. 1–22, 2015.

[11] 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 International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2013, pp. 80–95.

[12] T. Limoncelli, "Automation should be like iron man, not ultron," *Communications of the ACM*, vol. 59, no. 3, pp. 58–61, 2016.

[13] A. Alebrahim, S. Faßbender, M. Heisel, and R. Meis, "Problem-based requirements interaction analysis," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2014, pp. 200–215.

[14] K. Herzig and N. Nagappan, "Empirically detecting false test alarms using association rules," in *Proceedings of the International Conference on Software Engineering (ICSE)*, 2015, pp. 39–48.

[15] X. Lan, "An experimental study towards achieving 100% recall of synonym in software requirement document with selected methods," Master's thesis, University of Waterloo, Waterloo, ON, Canada, 2015. [Online]. Available: https://cs.uwaterloo.ca/~dberry/FTP_SITE/students.theses/XiaoYeLan.thesis/XiaoyeLanThesis.pdf

[16] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram, "Advancing candidate link generation for requirements tracing: The study of methods," *IEEE Transactions on Software Engineering*, vol. 32, no. 1, pp. 4–19, 2006.

[17] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker, "A machine learning approach for tracing regulatory codes to product specific requirements," in *Proceedings of the International Conference on Software Engineering ICSE)*, 2010, pp. 155–164.

[18] J. Cleland-Huang, O. Gotel, and A. Zisman, Eds., *Software and Systems Traceability*. London, UK: Springer, 2012.

[19] C. Ingram and S. Riddle, "Cost-benefits of traceability," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 23–42.

[20] A. Mahmoud and N. Niu, "Supporting requirements to code traceability through refactoring," *Requirements Engineering Journal*, vol. 19, no. 3, pp. 309–329, 2014.

[21] T. D. Breaux and D. G. Gordon, "Regulatory requirements traceability and analysis using semi-formal specifications," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2013, pp. 141–157.

[22] F. Chantree, B. Nuseibeh, A. de Roeck, and A. Willis, "Identifying nocuous ambiguities in natural language requirements," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2006, pp. 56–65.

[23] A. P. Nikora, J. H. Hayes, and E. A. Holbrook, "Experiments in automated identification of ambiguous natural-language requirements," in *Proceedings of the International Symposium on Software Reliability Engineering*, 2010, pp. 229–238.

[24] T. D. Breaux and F. Schaub, "Scaling requirements extraction to the crowd: Experiments with privacy policies," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 163–172.

[25] E. Guzman and W. Maalej, "How do users like this feature? A fine grained sentiment analysis of app reviews," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 153–162.

[26] E. Knauss and D. Ott, "(semi-) automatic categorization of natural language requirements," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2014, pp. 39–54.

[27] M. Riaz, J. T. King, J. Slankas, and L. A. Williams, "Hidden in plain sight: Automatically identifying security requirements from natural language artifacts," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 183–192.

[28] M. Rahimi, M. Mirakhorli, and J. Cleland-Huang, "Automated extraction and visualization of quality concerns from requirements specifications," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 253–262.

[29] A. Sutcliffe, P. Rayson, C. N. Bull, and P. Sawyer, "Discovering affect-laden requirements to achieve system acceptance," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 173–182.

[30] F. Pittke, H. Leopold, and J. Mendling, "Automatic detection and resolution of lexical ambiguity in process models," *IEEE Transactions on Software Engineering*, vol. 41, no. 6, pp. 526–544, 2015.

[31] B. Gleich, O. Creighton, and L. Kof, "Ambiguity detection: Towards a tool explaining ambiguity sources," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2010, pp. 218–232.

[32] A. Delater and B. Paech, "Tracing requirements and source code during software development: An empirical study," in *Proceedings of the International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2013, pp. 25–34.

[33] P. Hübner, "Quality improvements for trace links between source code and requirements," in *Joint Proceedings of the REFSQ 2016 Co-Located Events: the REFSQ 2016 Doctoral Symposium*, 2016, pp. 1–7. [Online]. Available: http://ceur-ws.org/Vol-1564/paper29.pdf

[34] P. Arkley and S. Riddle, "Overcoming the traceability benefit problem," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2005, pp. 385–389.

[35] D. Cuddeback, A. Dekhtyar, and J. H. Hayes, "Automated requirements traceability: The study of human analysts," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2010, pp. 231–240.

[36] P. R. Anish and S. Ghaisas, "Product knowledge configurator for requirements gap analysis and customizations," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2014, pp. 437–443.

[37] A. Ferrari, F. Dell'Orletta, G. O. Spagnolo, and S. Gnesi, "Measuring and improving the completeness of natural language requirements," in *Proceedings of the International Working Conference on Requirements Engineering: Foundation of Software Quality (REFSQ)*, 2014, pp. 23–38.

[38] H. Yang, A. N. D. Roeck, V. Gervasi, A. Willis, and B. Nuseibeh, "Analysing anaphoric ambiguity in natural language requirements," *Requirements Engineering Journal*, vol. 16, no. 3, pp. 163–189, 2011.

[39] V. Gervasi, "Private communication," 2016.

[40] O. Tripp and J. Rubin, "A Bayesian approach to privacy enforcement in smartphones," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014, pp. 175–190.

[41] J. Rubin, M. I. Gordon, N. Nguyen, and M. Rinard, "Covert communication in mobile applications," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2015, pp. 647–657.

[42] G. Lucassen, F. Dalpiaz, J. M. E. M. van der Werf, and S. Brinkkemper, "Forging high-quality user stories: Towards a discipline for agile requirements," in *Proceedings of the IEEE International Requirements Engineering Conference (RE)*, 2015, pp. 126–135.