Toward Automating Requirements Satisfaction Assessment

E. Ashlee Holbrook
Lexmark
ashleeh@gmail.com

Jane Huffman Hayes
University of Kentucky
hayes@cs.uky.edu

Alex Dekhtyar
California Polytechnic
State University
dekhtyar@csc.calpoly.edu

Abstract

This paper introduces the automation of satisfaction assessment: the process of determining the satisfaction mapping of natural language textual requirements to natural language design elements. Satisfaction assessment is useful because it assists in discovering unsatisfied requirements early in the lifecycle when such issues can be corrected with lower cost and impact than later. We define the basic terms and concepts for this process and explore the feasibility of developing baseline methods for its automation. This paper describes the satisfaction assessment approach algorithmically and then evaluates the effectiveness of two proposed information retrieval (IR) methods in two industrial studies – one based on a large dataset including a complete requirements specification and design specification for a NASA science instrument, and one based on a smaller dataset for an open source project management dataset. We found that both approaches have merit, and that the more sophisticated approach outperformed the simpler approach in terms of overall accuracy of the results.

1. Introduction

In 2006, the National Defense Industrial Association (NDIA) examined the top software engineering issues within the Department of Defense and the Defense industry. The number one issue raised was "*the impact of system requirements upon software is not consistently quantified and managed in development or sustainment* [1]." Within this, a number of sub-issues was identified, two of which are of interest: 1) "*strengthen policies and guidance for maintaining full traceability across all levels of requirements,*" and 2) "…ensure requirements are validated, balanced and consistent [1]."

A question of interest for all software systems, but of crucial interest for safety-critical and mission-critical software, is that of whether or not the requirements have been adequately addressed or satisfied by the subsequent artifacts. Examples of questions that address satisfaction of requirements include considering whether the system requirements have been satisfied by the hardware and software requirements, or whether the high level software design satisfies all the requirements? An automated method for answering these questions would help us address items 1 and 2 above.

Projects may have many artifacts that, when paired, possess the 'is satisfied by' or 'satisfies' relationship. For example, customer requests, change requests, or enhancement requests may be examined to see if they are already satisfied by an existing requirement or feature description of a software product. There may be higher-level enterprise goals or objectives that need to be satisfied by a software project. If so, the software requirements may be examined to see if they 'satisfy' the enterprise goals.

In addition to assessing the "status" of satisfaction between "delivered" or completed artifact pairs, this technique can also be applied to evolving artifacts. For example, a requirements specification could be examined for satisfaction assessment with a number of evolving textual design solutions. This will support the comparison and evaluation of various design alternatives, by looking to see what best satisfies the requirements. Others have started to look at this issue from a formal perspective by focusing on partial goal satisfaction [2].

This paper specifically focuses on assessing whether requirements have been satisfied by lower level artifacts such as design. We propose a three-step approach to addressing this problem. First, we identify important components for each individual requirement/design element. Generally speaking, each component should represent an important facet of a requirement or a design element. We define satisfaction assessment as determination of whether each component of each requirement has been addressed in the design document. This in turn is

achieved by tracing the requirements document, broken into components (which we call "chunks") to the design document, also broken into chunks. The third step, not addressed here, is to have the human analyst vet the results of step two.

Two methods are used to determine the mapping of requirement to design element chunks. The first method is based on a simple idea of tracking and thresholding the percentage of common terms between the two chunks. This method is selected due to its simplicity and ease of implementation. The second method is vector space information retrieval using TF-IDF (term frequency - inverse document frequency) term weighting [35]. This traditional IR method is now commonly used in requirements tracing [12,13]. Vector space retrieval has previously been validated in the automated tracing domain, and was therefore chosen as an initial method to investigate for satisfaction assessment [12,13,36].

1.1 Definitions

We have narrowed our focus to the problem of assessing satisfaction of a set of textual requirements by a set of textual design elements. Definitions of terms are in order. The IEEE definition of a requirement is: "1. A condition or capability needed by a user to solve a problem or achieve an objective. 2. A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. 3. A documented representation of a condition or capability as in (1) or (2)" [41]. Requirements include three pieces of information: a) a subject: b) a modal verb, typically "shall;" and c) an object phrase describing what is required. The subject of a requirement is typically the name of the software system or a subsystem contained within it. The modal verb may be "will," "shall," or another word. Finally, the requirement must contain a phrase describing what the subject of the sentence must do or contain. The IEEE definition of design is "A software design description (SDD). An SSD is a document used to specify system architecture and application design in a software related project [41]." Textual design elements are descriptions of how a requirement will be implemented or paragraphs that describe a requirement in more detail.

Satisfaction assessment (process) is defined as determining the satisfaction mapping of portions of natural language textual requirements to natural language design elements [3]. A satisfaction mapping encodes a satisfaction decision that has been made about a set of textual requirement elements and a set of corresponding textual design elements [3]. Satisfaction assessment is the process of determining the satisfaction mapping of natural language textual requirements to natural language design elements. *A satisfaction assessment* (artifact) is defined as a set of satisfaction mappings for a given set of requirements and design elements [3]. Formal definitions and examples of the artifacts are presented in Section 3.

1.2 Contribution

This paper presents our initial work on the problem of satisfaction assessment. Our goal was to examine the feasibility of its automation and to investigate the appropriateness of some common techniques as baselines for its evaluation. Specifically, we defined a three-step process for satisfaction assessment. This paper directly addresses the first two steps. We present two methods that apply information retrieval techniques to assess the satisfaction of requirements by design elements. We discuss a study evaluating the accuracy of the methods by examining two industrial datasets. We found that both methods, despite their simplicity, produce reasonably accurate satisfaction assessments. When comparing the performance of the methods, we found that the more sophisticated method (TF-IDF*)* had higher overall accuracy.

1.3 Paper Organization

The paper is organized as follows. Section 2 presents related work. Section 3 discusses satisfaction assessment, explaining methods in detail. Section 4 presents the study design and threats to validity. Section 5 presents the results and analysis. Section 6 concludes and discusses future work.

2. Related Work

Satisfaction assessment may be thought of as a way to validate whether requirements have been fully addressed by design elements and provides a way to measure the quality of a software project. The majority of previous automation work in the requirements traceability community focused on candidate link generation, not on satisfaction assessment. This work, however, has paved the way for our study and is briefly described here.

Previous work on requirements validation has focused on formally specifying requirements [28,22,10], optimizing natural language processing (NLP) approaches to requirements analysis, and discovering potential ambiguities [16,9,7].

Durán et al. used XSLT and requirements in XML to automatically verify requirement qualities such as completeness and lack of ambiguity [6]. Analysts have used requirement defect detection techniques [20] to discover requirements that cannot be satisfied (i.e., inconsistent and omitted) and inconsistencies between requirements and design. Reading methods such as scenario-based [27] and perspective-based reading [2, 26] have also been used to increase the quality of requirement specifications.

Automation of requirements tracing has received extensive attention in recent years. For a detailed survey, we refer the reader to [13]. Tracing examines the creation of a requirements traceability matrix (RTM) that relates requirements to design to code and beyond. Recent work concentrated on applying IR methods to tracing. Antoniol et al. [36] and Marcus and Maletic [17] applied IR methods to the problem of tracing design to code. Cleland-Huang [4] used IR to trace non-functional requirements. Hayes et al. investigated the process of tracing and built a special-purpose requirements tracing tool called RETRO (REquirements TRacing On-target) [12, 13]. Spanoudakis et. al. created a system to automatically generate traceability information based on tracing rules [33].

Additionally, several researchers have examined requirement quality through design and requirement analysis. Diallo et. al. used ScenarioML to create mappings between requirement-level scenarios and system architecture [29]. Alspaugh and Antón examined automation of requirement scenario analysis to determine requirement quality, looking at four primary traits: well-definedness, coverage, minimality, and coherence [30]. Robinson looked at rule-based requirements monitors to dynamically analyze requirements as a system is designed [31]. Letier and van Lamsweerde created a system to analyze partial goal satisfaction to help quantify the impact of partially met requirements due to design constraints [32].

Algorithmic techniques that are useful for both assessing requirement satisfaction and tracing include keyword extraction methods [14] and the vector space model for information retrieval [24]. Vector space models represent documents as vectors by extracting terms, weighting these by relevance and document location, and ranking the document as a whole based on a given query.

3. Satisfaction Assessment

This section proposes approaches to satisfaction assessment as well as measures for assessing the validity of the approaches. Each approach was implemented in a tool called REquirements SATisfaction (RESAT). The tool is written for Windows in Visual C# and is approximately 10,130 lines of code excluding external libraries.

3.1 Satisfaction Assessment Defined

In this paper, satisfaction assessment is defined as the process of determining the satisfaction mapping of natural language textual requirements to natural language design elements. Given a set of requirements decomposed into terms ($R = \{t_{r1}, t_{r2}, ...\}$) or phrases ($R = \{p_{r1}, p_{r2}, ...\}$) and a set of design element terms ($D = \{t_{d1}, t_{d2}, ...\}$) or phrases ($D = \{p_{d1}, p_{d2}, ...\}$), a satisfaction mapping is a set of pairs of terms (t_{rn}, t_{dm}) where t_{rn} is a term in a set of requirements and t_{dm} is a term in the set of design elements where t_{rn} is directly correlated to t_{dm}[1]. A satisfaction mapping may also occur at the phrase level. In this case, the mapping will consist of a series of phrase pairs (p_{rn}, p_{dm}) with one phrase, p_{rn}, being a phrase in a requirement, and p_{dm} being a phrase in a corresponding design element where p_{dm} directly addresses p_{rn}. In this study, we have broken down the requirement and design element text into phrases based on parts of speech processing. We refer to each of these phrases as *requirement chunks* or *design element chunks*. All chunks have unique identifiers that we use to build candidate satisfaction mappings.

Satisfaction assessment, as performed in this study, consists of a variety of processing techniques. Requirements and design elements are taken as input in their natural form. There are no formatting or language rules imposed to avoid placing a burden on those who specify the requirements and design. No models are required, nor does the method attempt to examine possible underlying abstract models. The sole requirement for input was that all documents used for this study were in English text. An RTM for each dataset was also used as input to limit the search space for satisfaction mappings.

Performing satisfaction assessment is a three step process. The first step is to parse the requirement and design elements into chunks. This step is described in Section 3.2.

Step two is to map the design chunks to the requirement chunks, resulting in a candidate satisfaction assessment mapping. This step determines if each pair of requirement and design

[1] That is to say that these items trace to each other and the traceability relationships are documented in the RTM.

chunks are similar. We view this as a tracing problem, which can be defined as an information retrieval (IR) problem: given a document collection and a query, determine those documents from the collection that are relevant to the query. Our prior work, and that of others, has shown that requirement and design similarity can be modeled, or at least approximated, by the document relevance notions on which different IR algorithms rely [4,12,13,14,17,36]. The methods that we applied to the tracing step are described in Section 3.3.

Finally, the results from step 2 are shown to the analyst for confirmation/approval. The analyst work is step three. This step is not in the scope of the current paper, but it should be noted that the output of the step is referred to as the final satisfaction assessment mapping.

3.2 Processing

Requirements and design elements were subjected to a series of processing steps before a candidate satisfaction assessment could be determined. First, each textual requirement and design element was preprocessed. Second, we constructed and applied a domain specific thesaurus, which contains a set of synonym pairs for domain-specific vocabulary. The thesaurus was generated by analyzing a subset of roughly 25% of the requirements and design elements for each dataset domain. Thesaurus entries for this work are in the form "term synonym1 synonym2… synonymN" where term is synonymous with synonyms 1 through N. For example, the terms "error" and "problem" could be included as synonyms in the thesaurus. Next, each requirement and design element was tokenized into chunks based on parts of speech.

Each chunk is a phrase in a sentence. Chunking takes place by parsing sentences, with each phrase of the sentence identified uniquely as a chunk. Incremental processing of text is shown in Figure 1 and a sample satisfaction mapping is shown in Figure 2. Note that the text in Figures 1 and 2 has been formatted to show the matches visually (e.g., Figure 2 tells us that requirement chunk 19, "the original error code," maps to design chunk 32 in design element 1, "an error code." To make it easier to see that they are related, each is double-underlined. Similar formatting changes (bold, italic and underline combinations) indicate other potential mappings between requirement element chunks and design element chunks. Finally, similarity measures are calculated between chunks of requirements and the design elements they are tied to in the project RTM.

The similarity measures for this study are determined using either method described next.

3.2.1. Satisfaction assessment using TF-IDF. TF-IDF is a statistical measurement of the importance of a term within a document. ***Term frequency (TF)*** is the (possibly normalized) number of times a term appears within a document. ***Inverse document frequency (IDF)*** of a term is the logarithm of the ratio of the total number of documents in a collection to the number of documents that contain the term. The less frequent a term is, the more discriminating

Chunked Requirement Text: RE1	<1>The DPU-CCM</1> <2>shall be able</2> <3>***to count***</3> <4>***a consecutively reported error*** </4>. <5>When<5> <6>**the count**</6> <7>for</7> <8>***a particular error ID***</8>, <9>**exceeds**</9> <10>*250*</10> <11>for</11> <12>a particular reporting period</12>, <13>***the error code***</13> <14>will be replaced</14> <15>with</15> <16>*an error code sequence*</16> <17>which</17> <18>shall include</18> <19>the original error code</19> <20>and</20> <21>**the number of times**</21> <22>***the error***</22> <23>**was reported** </23>.
Chunked Design Element Text: DE1	<24>The ccmErrEnq() function</24> <25>tracks</25> <26>the last error reported</26> <27>and<27> <28>its<29> <30>frequency of occurrence</30>. <31>Once</31> <32>an error code</32><33>***has been reported***</33> <34>it</34> <35>becomes<35> <36> ***the previously reported error code***</36>…
Chunked Design Element Text: DE2	<100>In</100> <101>order</101> <102>to insure</102> <103>that</103> <104>**error counts**</104> <105>are</105> <106>not</106> <107>lost</107>…

Figure 1. Sample Requirement and Design Element Satisfaction Assessment.

Satisfaction Assessment:
1,2,5,7,14,15,17,20 – No Satisfaction Mapping
3 - 43 (***bold underline italic***)
4 - 33, 36, 45, 49, 66, 79, 115 (***bold italic***)
6 - 40, 64, 104, 123, 132 (**bold**)
8 - 33, 36, 45, 49, 66, 79, 115 (***bold italic***)
9 - 118, 124 (**bold underline**)
10 - 119, 125 (*underline italic*)
11 - 126
12 - 59, 121, 127 (underline)
13 - 33, 36, 45, 49, 66, 79, 115 (***bold italic***)
16 - 61, 130 (*italic*)
19 - 32, 51 (double underline)
21 - 40, 64, 104, 123, 132 (**bold**)
22 - 33, 36, 45, 49, 66, 79, 115 (***bold italic***)
23 - 52 (bold double underline)

Figure 2. Sample Requirement and Design Element Satisfaction Assessment (cont.).

power it has, and thus the higher the IDF. The weight of each keyword in a document is the product of TF and IDF [35].

Each requirement and design element chunk is considered an individual document within the document collection. TF-IDF similarity scores[2] are calculated between pairs of chunks *(cr1, cd2),* where *cr1* is a requirement chunk in requirement 1 and *cd2* is a design element chunk that is in design element 2, and where requirement 1 is mapped to design element 2 in the RTM for the dataset. All such pairs from the RTM are considered to be *potential* matches. If the pair *(cr1, cd2)* has a similarity score above a given threshold value (we used a set of 18 threshold or filter values – 9 values starting at 0.01 and incrementing by 0.01 to 0.09, and 9 values starting at 0.1 and incrementing by 0.1 until 0.9), then the two are considered a satisfaction match and the *pair (cr1, cd2)* is included in the *candidate* satisfaction assessment mapping produced by this method. The entire set of satisfaction match pairs that have similarity scores greater than the threshold value is considered to be a *candidate satisfaction assessment* for a given dataset. We call our final output a candidate mapping because we acknowledge that an analyst should examine the tool's output and confirm the *final* mapping [40]

3.2.2 Naïve satisfaction assessment. The naïve satisfaction approach examines textual similarity only. If terms within a requirement chunk and design chunk in the dataset contain the same root or the root of a synonym, then the terms are considered a match. The overall percentage of matching terms in a chunk, excluding stop words, is the weighted similarity value for a requirement and design element chunk. Threshold values from 0.01 to 0.9 (described in Section 3.3.1) are used to filter chunks, chunks with similarity values below the threshold do not appear in the candidate satisfaction assessment mapping.

3,2,3, RESAT Tool. The software implementation of the satisfaction assessment procedures is called *RESAT (REquirements SATisfaction)*. From within the RESAT tool, users can load a set of requirements and a set of design documents, a domain thesaurus, set threshold values as described for the methods below, and perform automated satisfaction assessment using the method described below. Batch mode processing is also available within the tool to process datasets at multiple threshold values.

3.3 Measures

Our primary goal is assessment of accuracy of the methods in determining the associations (links) between the requirement and design element chunks. In our evaluation, we used three traditional information retrieval measures: *precision, recall,* and *f-measure.* Given a list of candidate satisfaction pairs, the *precision* of the list is the percentage of the retrieved pairs that are correct. *Recall* of the list is the percentage of correct pairs that were retrieved.

High *precision* means low incidence of type I errors (including incorrect pairs). Higher precision indicates that analysts will have fewer incorrect results to remove in order to obtain a true satisfaction assessment. High *recall* means low incidence of type II errors (omitting true pairs). High recall indicates that a majority of the true matches were returned, meaning an analyst will have to search less for satisfaction mappings that have been omitted from a candidate satisfaction assessment.

Given a list of candidate satisfaction pairs, the pair *(precision, recall)* provides a good description of the list's accuracy. However, when the accuracy of different lists needs to be compared, it is much more convenient to combine *precision* and *recall* into a single measure. This is typically done via *f-measure* [35], the harmonic mean of *recall* and *precision.* In our study, we use a variant of *f-measure* called *f2,* computed as follows:

$$f2 = \frac{3 \cdot precision \cdot recall}{(2 \cdot precision) + recall}.$$

[2] In situations when chunks compared to each other are rather short, *tf-idf* weights become essentially pure *idf* weights, as all term occurrences in the chunks are unique.

The *f2* measure is a weighted harmonic mean of *recall* and *precision* which favors *recall*. We choose to favor *recall* in our evaluation because traditionally the cost of repairing type II errors (errors of omission) is higher than the cost of repairing type I errors (errors of commission). Thus, when comparing two lists of candidate satisfaction pairs with the same number of total errors, we give preference to the list with fewer type II errors, i.e., with *higher recall.*

4. Evaluation

In order to validate the satisfaction assessment methods, we undertook a study using two datasets. The study design and threats to validity are presented below.

4.1 Study Design

The study was conducted on data from two industry projects. Each dataset consisted of textual requirements (high-level elements) and textual design documents (low-level elements). The first dataset, **NASA CM-1** [38], consists of the entire requirement specification and the entire design document for a NASA scientific instrument. There are 235 requirements and 220 design documents. After the dataset was chunked based on grammatical structure, there were a total of 2,780 requirement chunks and 10,490 design chunks. The RTM for this dataset contains 362 links between requirements and design elements, with a density of 1.54 design elements per requirement. The RTM is sparse, meaning that not all requirements in the dataset have corresponding design elements. Using the RTM, there were 205,696 requirement-design element pairs to be analyzed. Without the RTM, considering every possible requirement-design element pair, there would have been 29,162,200 comparisons.

The second dataset is based on an open source program called **GanttProject** used to create Gantt charts and perform basic project management [39]. There are 17 requirement elements and 78 design elements. After chunking, there were 312 requirement chunks and 632 design chunks. The RTM for the GanttProject dataset contains 68 links. An average of 4.0 design elements link to each requirement. From these, using the RTM, there were 15,430 requirement-design element pairs to be analyzed. Without the RTM, considering every possible requirement-design element pair, there would have been 275,064 comparisons to be made.

In order to validate the accuracy of our methods, we built "golden" answer sets for the datasets. Two analysts (not among the authors) built the answer sets from the chunked text for each dataset. One analyst constructed the initial answer sets, while the second analyst reviewed and offered suggestions as necessary. As only one analyst built the answer set, inter-rater reliability statistics could not be applied. The analysts met and reviewed the suggestions to produce final answer sets. The final satisfaction answer set for CM-1 has a total of 959 satisfaction mappings, with an average density of 0.09 design element chunk mappings per requirement chunk. For GanttProject, there are 307 links between requirement and design elements with a density of 0.983974359 design element chunk mappings per requirement chunk. It took the analysts a combined total of 15 hours to create the initial Gantt answer set, and another four hours for verification.

It took analyst 1 120 hours to create the initial CM-1 answer set, and it took analyst 2 another 40 hours for verification.

In [13], the authors introduce a taxonomy of project sizes for the purpose of performing traceability tasks. According to this taxonomy, the **GanttProject** dataset is a ***small project*****,** while the ***CM-1*** dataset is a ***medium-size project.*** In our expertise, the **CM-1** dataset is rather typical of software artifacts generated for various NASA instruments.

Each method from Section 3.2 was applied to each dataset. The results are discussed in Section 5.

4.2 Threats to Validity

There are external threats to validity that may impact the generalizability of our results. Our methods were applied to only two systems. The second system, GanttProject, was fairly small (though it was a complete project). Also, the analysts that built the answer sets were not subject matter experts on the systems. It is possible that a different group of analysts may yield different "golden" answer sets.

We attempted to mitigate reliability threats to validity. Our process is outlined and repeatable. Our datasets are available upon request and we plan to make the RESAT tool available upon request, pending University approval. We believe the study is repeatable.

We mitigate construct validity in this study by using real world requirements and design elements that were not specifically created for this work. The datasets chosen were not reorganized or modified, the preprocessing steps were constant between methods, and the same methods were applied to both datasets in an effort to mitigate threats to internal validity.

5. Results and Analysis

Our evaluation was conducted as follows. Both the naïve method and TF-IDF retrieval method were applied to the chunked requirements and design elements of each dataset. The lists of candidate satisfaction pairs were constructed for threshold parameters with values 0.01, 0.02,...,0.09, 0.1, 0.2, ...,0.9. *Precision, recall* and the *f2* measure were then computed for each threshold value.

The complete results of our study are shown in Tables 2 through 5 at the end of the paper. Each table documents the *recall, precision,* and the *f2 measure* values for each threshold value considered by the method. The key results of the study are summarized in Table 1 above. In it, for each dataset and method, we report the *largest value of the threshold* at which the best value of the *f2* measure was achieved, listing also the *precision, recall* and *f2* values for that threshold. For example, we can see that the largest threshold value with the best value of *f2* occurred at threshold 0.2 for TF-IDF for the Gantt dataset with *recall* of 0.664, *precision* of 0.486, and *f2* measure of 0.619. Additionally, we include the cases when significantly higher *recall* (albeit at lower *f2* value) was achieved.

Gantt dataset. We first examine the performance of the methods on the Gantt dataset. The *recall* values are almost identical for TF-IDF and naïve. *Precision* is somewhat higher for TF-IDF at 0.486 as compared to 0.41. The *f2* measure for TF-IDF is better than for naïve at 0.619 compared to 0.539. Overall, the TF-IDF method outperforms the naïve method for the Gantt dataset.

That aside, it is worth noting that both methods fared reasonably well. The *recall* values for each method are acceptable (above 65%) and the *precision* values are rather high (above 40%). The *f2* measure of each method exceeds 0.5. Note also that TF-IDF was able to achieve the higher *f2* value at a much higher threshold value (0.2 versus 0.04 for the naïve method).

CM-1 dataset. Next, we examine the results for the CM-1 dataset. The highest *f2* value, 0.315, for the naïve method is at the threshold value of 0.09, with the *recall* value of 0.525 and *precision* of 0.12. The TF-IDF method has the highest *f2* value, 0.528, at a threshold of 0.2 with *recall* at 0.742 and *precision* of 0.245. In this case, the *recall* is much higher for TF-IDF than for the naïve method (0.742 as compared to 0.525), the *precision* is much higher at 0.245 as compared to 0.12, and the *f2* value is much higher at 0.528 as compared to 0.315. Thus, on the CM-1 dataset, TF-IDF clearly outperformed the naïve method.

It should be noted that *the best recall* value achieved by the naïve method for CM-1 did outperform the *recall* for TF-IDF, as shown in the table. At threshold of 0.03, the naïve method had *recall* of 0.781 (a little bit better than TF-IDF), but this high recall came with a very low *precision* of 0.063 (almost four times worse than for the best TF-IDF case), and yielded a value of just 0.238 for *f2*. Note that the *f2* value is much lower than the best value achieved for the naïve method (0.315).

Analysis. The methods did not fare as well on CM-1 as they did on Gantt. Though *recall* values of 0.74 and above are achieved by each method, it is with low *precision* (0.245 and 0.063). The *f2* measure exceeds 0.5 for the TF-IDF method, but not for the naïve method. On both datasets, TF-IDF had reached higher values of *f2*, thus outperforming the naïve method, despite showing somewhat lower (but very comparable) recall.

What sets the TF-IDF method apart is that it achieves the best results at a significantly higher threshold (0.2 vs. 0.04), which means that the TF-IDF method allows us to be more selective about which similarity values to treat as a match.

6. Conclusions and Future Work

In conclusion, both methods performed well on the Gantt dataset with *recall* values above 65%, *precision* values above 40%, and *f2* values above 0.5. The TF-IDF method showed acceptable performance on the CM-1 dataset, with the naïve method falling a bit short.

We believe that we have successfully established that both the naïve and TF-IDF methods can serve as viable baseline methods for assessing performance of automated satisfaction assessment techniques. The naïve method, in particular, is very simple and delivers decent performance with relatively little effort expended.

To a large degree, the performance of the naïve method establishes a measuring stick for us. Any automated methods that fall short of its performance shall be deemed unacceptable for dealing with satisfaction assessment. Methods that exhibit similar performance shall be considered candidates for further improvement, but if no further improvement is achieved, Occam's razor dictates that we abandon them in favor of the naïve method as well.

We believe there are a number of areas for future work. As a long term goal, we would like to evaluate additional datasets as well as establish methods to

Table 1. Summary of the evaluation.

Dataset:	Gantt				CM-1			
	threshold	recall	precision	**f2**	threshold	recall	precision	**f2**
Naïve	0.04	0.67	0.41	**0.539**	0.09 0.03	0.525 0.781	0.12 0.063	**0.315** **0.238**
TF-IDF	0.2	0.664	0.486	**0.619**	0.2	0.742	0.245	**0.528**

automatically determine the optimal threshold values for each method. Optimal threshold values will likely vary by dataset size (number of requirements, design elements, and RTM density) and domain. In addition, future studies include developing measures of requirement quality. The two datasets we used in this work are but a drop in the ocean of possible types of software engineering artifacts that may undergo satisfaction assessment. Because of this, we are not in a position to draw any strong general conclusions about the applicability of specific methods to different types of documents. The only observation we feel comfortable making in this respect is that when requirements are written using proper grammar, punctuation, spelling etc., tagging and natural language processing algorithms often perform better.

We hope to develop additional satisfaction assessment methods. We plan to further examine the thresholds used. We plan to examine potential new methods, including an approach that generates chunk matches based on a set of predefined rules on content and grammatical structure of the chunks. In addition, validation needs to be expanded. We plan to apply our work to additional datasets in other domains. We also plan to examine the applicability of our methods to different artifact pairs.

7. Acknowledgments

This work is funded in part by the National Science Foundation under NSF grant CCF-0811140. This work was partially sponsored by NASA under grant NNG05GQ58G. We thank Wenbin Li, Hakim Sultanov, and Bill Kidwell for building the golden answer sets. Thanks to Stephanie Ferguson, Marcus Fisher, Ken McGill, Tim Menzies, and everyone at the NASA IV&V facility. Thanks also to fellow graduate students Jody Larsen, Senthil Sundaram, Liming Zhao, and Sravanthi Vadlamudi.

8. References

[1] National Defense Industrial Association, Systems Engineering Division Task Group Report, "Top Software Engineering Issues within Department of Defense and Defense Industry," September 2006, Version 5a, 9/26/06.

[2] Letier, E. and van Lamsweerde, A. 2004. Reasoning about Partial Goal Satisfaction for Requirements and Design Engineering. SIGSOFT Softw. Eng. Notes 29, 6 (Nov. 2004), 53-62.

[3] Holbrook, E. Ashlee, "Assessing Satisfaction of Requirements by Design Elements," in Proceedings of the 2006 IEEE Requirements Engineering (RE) Doctoral Symposium.

[3] B.W. Boehm, Software Engineering, IEEE Trans. On Computers, 25(12):1226-1241, 19.

[4] J. Cleland-Huang, et.al. "Goal-Centric Traceability for Managing Non-Functional Requirements", Int. Conference on Software Engineering, Saint Louis, May 2003.

[5] "CM1 DataSet," Metrics Data Program Website, CM-1 Project, http://mdp.ivv.nasa.gov/mdp_glossary.html#CM-1.

[6] A. Durán, A Ruiz, M. Toro, "An Automated Approach for Verification of Software Requirements", Jornadas de Ingeniería de Requisitos Aplicada, Seville, Spain, 2001.

[7] C. Denger, D.M. Berry, and E. Kamsties, "Higher Quality Requirements Specifications through Natural Language Patterns," IEEE Software-Science, Technology & Engineering (SwSTE'03), pp. 80-89, Israel, Nov. 2003.

[8] C. Fox, "A Stop List for General Text." SIGIR Forum 24, 1-2 (Sep. 1989), 19-21.

[9] L. Goldin, and Berry, D.M., ``AbstFinder, A Prototype Natural Language Text Abstraction Finder for Use in Requirements Elicitation'' Automated Software Eng., 4(4), 375-412, Oct., 1997.

[10] S. Greenspan, J. Mylopoulos, A. Borgida, "On Formal Requirements Modeling Languages: RML Revisited", Proc. 16th International Conference on Software Engineering, p.135-147, May 16-21, 1994, Sorrento, Italy.

[11] K. Hacioglu, S. Pradhan, W. Ward, J. H. Martin, and D. Jurafsky. "Semantic Role Labeling by Tagging Syntactic Chunks." In Proceedings of CoNLL 2004 Shared Task.

[12] J.H. Hayes, A. Dekhtyar, J. Osbourne, "Improving Requirements Tracing via Information Retrieval", Int. Conf. on Requirements Engineering, Monterey, California, Sept. 2003,pp. 138 – 148.

[13] J.H. Hayes, A. Dekhtyar, and S. Sundaram, "Advancing Requirements Tracing: The Study of

Methods", IEEE Trans. on Software Engineering, 32(1), Jan. 2006, pp. 4 -19.
[14] J.H. Hayes, A. Dekhtyar, S. Sundaram, "Advances in Dynamic Generation of Traceability Links", Tech Report, February 2006, (TR 451-06).

[15] ISO 9000 (2000). Quality Management Systems – Fundamentals and Vocabulary. International Organization for Standardization.

[16] R. Lecceuche, "Finding Comparatively Important Concepts between Texts." Automated Software Engineering (ASE'00). Washington, DC, 55.

[17] A.Marcus, J.Maletic, "Recovering Documentation-to-Source Code Traceability Links using Latent Semantic Indexing", Software Engineering, Portland, May 2003, pp. 125–135.

[18] "OpenNLP," Open Natural Language Processing project. Available for download at: http://opennlp.sourceforge.net/about.html.

[19] M.F. Porter, 1980, "An Algorithm for Suffix Stripping," Program, 14(3) pp 130−137.

[20] A. Porter and L. Votta, "Comparing Detection Methods For Software Requirements Inspections" Empirical Software Engineering, 3(4), 1998, 355 – 379.

[21] P. Rayson, R. Garside, and P. Sawyer," Recovering Legacy Requirements." Requirements Engineering: Foundations of Software Quality, June 14-15 1999, Heidelberg, Germany, pp. 49-54.

[22] W.N. Robinson, S. Pawlowski, "Managing Requirements Inconsistency with Development Goal Monitors," IEEE Trans. on Software Eng., Nov/Dec 1999.

[23] K. Ryan, "The Role of Natural Language in Requirements Engineering." Requirements Engineering (RE'93), San Diego, pp. 80-82, 1993.

[24] G. Salton, Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[25] J. Sayyad Shirabad, and Menzies, T.J. (2005) The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada. Available: http://promise.site.uottawa.ca/SERepository.

[26] F. Shull, I. Rus, and V.R. Basili, "How Perspective-Based Reading Can Improve Requirements Inspections," IEEE Computer, 33(7), pp. 73-79, July 2000.

[27] A. Sutcliffe, "Scenario-Based Requirement Analysis," Requirements Engineering Journal 3(1), 1998, 48−65.

[28] J.M. Spivey 28, "Understanding Z," Cambridge, 88.

[29] Mamadou H. Diallo, Leila Naslavsky, Hadar Ziv, Thomas A. Alspaugh, and Debra J. Richardson. "Evaluating Software Architectures Against Requirements-level Scenarios," Third International Workshop on the Role of Software Architecture for Testing and Analysis (ROSATEA'07), Boston, MA, July 2007.
[30] Scenario Support for Effective Requirements Thomas A. Alspaugh and Annie I. Antón. Information and Software Technology **50**(3) pp. 198-220. February 2008.

[31] W. N. Robinson, "Implementing Rule-Based Monitors within a Framework for Continuous Requirements Monitoring," Hawaii International Conference on System Sciences (HICCS '05). Big Island, HI, pp. 188, 2005.

[32] Letier, E. and van Lamsweerde, A. 2004. Reasoning about partial goal satisfaction for requirements and design engineering. In Proceedings of the 12th ACM SIGSOFT Twelfth international Symposium on Foundations of Software Engineering (Newport Beach, CA, USA, October 31 - November 06, 2004). SIGSOFT '04/FSE-12. ACM, New York, NY, 53-62.

[33] G. Spanoudakis, A. d'Avila Garcez, A. Zisman. "Revising Rules to Capture Requirements Traceability Relations: A Machine Learning Approach." Software Engineering & Knowledge Engineering (SEKE'03). San Francisco, CA, pp 570.

[34] Santorini, B. 1990. Part-of-speech tagging guidelines for the Penn Treebank Project. Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania.

[35] R. Baeza-Yates, and B. Ribeiro-Neto, Modern Information Retrieval. Addison-Wesley, 1999.

[36] G. Antoniol, et. al. "Recovering Traceability Links between Code and Documentation", IEEE Trans. on Software Engineering, Volume 28, No. 10, 2002, 970-983.

[37] V.R. Basili,, et. al. "The Empirical Investigation of Perspective-Based Reading." Empirical Software Engineering, 1(2), 1996, 133-164.

[38] Predictor Models in Software Engineering (Promise) Software Engineering Repository. http://promise.site.uottawa.ca/SERepository.

[39] GanttProject, http://ganttproject.biz/

[40] Hayes, J. H. and Dekhtyar, A. 2005. Humans in the traceability loop: can't live with 'em, can't live without 'em. In Proceedings of the 3rd International Workshop on Traceability in Emerging Forms of Software Engineering (Long Beach, California, November 08 - 08, 2005). TEFSE '05. ACM, New York, NY, 20-23.

[41] IEEE Standard Glossary of Software Eng. Terminology, Std 610.12 Std 610.12-1990(R2002), 1990.

APPENDIX

Table 2. Gantt dataset naïve method results.

Filter Value	Naïve - Overall Recall	Naïve - Overall Precision	Naïve - F2-Measure
0.01	0.678	0.294	0.537
0.02	0.678	0.294	0.537
0.03	0.678	0.295	0.538
0.04	**0.674**	**0.299**	**0.539**
0.05	0.645	0.312	0.531
0.06	0.635	0.312	0.526
0.07	0.557	0.325	0.487
0.08	0.554	0.325	0.485
0.09	0.463	0.36	0.438
0.1	0.433	0.363	0.417
0.2	0.078	0.32	0.092
0.3	0	0	0

Table 3. Gantt dataset TF-IDF method results.

Filter Value	TF-IDF - Overall Recall	TF-IDF - Overall Precision	TF-IDF - F2-Measure
0.01	0.664	0.486	0.619
0.02	0.664	0.485	0.619
0.03	0.664	0.485	0.619
0.04	0.664	0.485	0.619
0.05	0.664	0.485	0.619
0.06	0.664	0.485	0.619
0.07	0.664	0.485	0.619
0.08	0.664	0.485	0.619
0.09	0.664	0.486	0.619
0.1	0.664	0.486	0.619
0.2	**0.664**	**0.486**	**0.619**
0.3	0.58	0.492	0.56
0.4	0.554	0.528	0.548
0.5	0.521	0.565	0.529
0.6	0.423	0.647	0.455
0.7	0.391	0.764	0.433
0.8	0.339	0.819	0.384
0.9	0.313	0.835	0.357

Table 4 CM-1 dataset naïve method results.

Filter Value	Naïve - Overall Recall	Naïve - Overall Precision	Naïve - F2-Measure
0.01	0.783	0.063	0.237
0.02	0.783	0.063	0.237
0.03	0.781	0.063	0.238
0.04	0.759	0.064	0.239
0.05	0.69	0.068	0.245
0.06	0.681	0.07	0.248
0.07	0.635	0.087	0.281
0.08	0.611	0.085	0.273
0.09	**0.525**	**0.121**	**0.315**
0.1	0.449	0.113	0.282
0.2	0.1	0.075	0.094
0.3	0	0	0

Table 5. CM-1 dataset TF-IDF method results.

Filter Value	TF-IDF - Overall Recall	TF-IDF - Overall Precision	TF-IDF - F2-Measure
0.01	0.746	0.235	0.52
0.02	0.746	0.235	0.52
0.03	0.746	0.235	0.52
0.04	0.746	0.235	0.52
0.05	0.746	0.235	0.52
0.06	0.746	0.235	0.52
0.07	0.746	0.235	0.52
0.08	0.746	0.235	0.52
0.09	0.746	0.235	0.52
0.1	0.746	0.235	0.52
0.2	**0.742**	**0.245**	**0.528**
0.3	0.694	0.264	0.524
0.4	0.606	0.287	0.495
0.5	0.531	0.309	0.464
0.6	0.445	0.328	0.416
0.7	0.368	0.36	0.366
0.8	0.307	0.376	0.318
0.9	0.26	0.352	0.274