NATURAL LANGUAGE PROCESSING FOR DETECTING FORWARD REFERENCE IN A DOCUMENT

Daniel Siahaan[1], Izzatul Umami[2]

***Abstract*— Meyer's seven sins have been recognized as types of mistakes that a requirements specialist are often fallen to when specifying requirements. Such mistakes play a significant role in plunging a project into failure. Many researchers were focusing in ambiguity and contradiction type of mistakes. Other types of mistakes have been given less attentions. Those mistakes often happened in reality and may equally costly as the first two mistakes. This paper introduces an approach to detect forward reference. It traverses through a requirements document, extracts, and processes each statement. During the statement extraction, any terms that may reside in the statement is also extracted. Based on certain rules which utilize POS patterns, the statement is classified as a term definition or not. For each term definition, a term is added to a list of defined terms. At the same time, every time a new term is found in a statement, it is check against the list of defined terms. If it is not found, then the requirements statement is classified as statement with forward reference. The experimentation on 30 requirements documents from various domains of software project shows that the approach has considerably almost perfect agreement with domain expert in detecting forward reference, given 0.83 kappa index value.**

Keywords—Forward Reference; Natural Language Processing; Term

Abstrak*—*Meyer's seven sins dikenal sebagai jenis kesalahan yang sering dilakukan sistem analis ketika menspesifikasi kebutuhan. Kesalahan-kesalahan tersebut berperan besar sebagai penyebab gagalnya sebuah proyek. Banyak peneliti memfokuskan diri pada kesalahan berjenis kerancuan dan kontradiksi. Jenis kesalahan yang lain kurang mendapat perhatian. Padahal jenis kesalahan tersebut juga pada kenyataannya sama dampak finansialnya disbanding dua jenis pertama. Artikel ini menjelaskan sebuah pendekatan untuk mendeteksi forward reference. Pendekatan ini akan mengekstrak dan memproses setiap pernyataan dalam dokumen kebutuhan Selama proses ekstraksi tersebut, setiap istilah yang ditemukan juga diekstraksi. Berdasarkan aturan tertentu yang memanfaatkan pola POS, pernyataan diklasifikasikan sebagai sebuah definisi istilah atau bukan. Untuk setiap definisi tersebut, sebuah istilah akan ditambahkan ke daftar istilah terdefinisi. Pada saat yang sama, untuk setiap kali sebuah istilah baru ditemukan dalam sebuah pernyataan, pendekatan ini akan mengecek eksistensi definisinya. Jika tidak ditemukan, maka pernyataan tersebut diklasifikasikan sebaga pernyataan yang mengandung forward reference. Hasil pengujian atas 30 dokumen kebutuhan dari berbagai ranah proyek perangkat lunak menunjukkan bahwa pendekatan ini hampir dapat diandalkan sebagaimana seorang ahli dalam mendeteksi forward reference, dengan nilai kappa 0.83.

Kata Kunci—*Forward Reference, Istilah, Pemrosesan Bahasa Alamiah*

I. INTRODUCTION

Requirements specification as part of requirements engineering is mainly dealing with how to express requirements in a specific, measurable, realizable, attainable, and time-bound manner. Requirements specification should be agreed by all stakeholders. It concerned with the process to elicit, analyze, and validate/verify requirements. These processes are documented for the most part in natural language. Software Requirements Specification (SRS) is one of deliverables produced iteratively throughout software development lifecycle. It is one of the most important artefacts produced during this phase of software development. The quality of SRS document determines whether a software project may end up as a success story or just another project failure. It stands as the first entrance before and provides input for design, coding and testing phases. The report in 2009 on software project chaos from Standish Group indicates that 31.1% software projects failure rooted from requirements specification. Therefore, considerable resources, in term of man hour, are spend in order to ensure the SRS document quality This is due to the fact that the real-life SRS documents may take up to considerable amount of pages, sentences, figures, and tables.

During requirements specification, engineers focus on specifying requirements, which on most cases is written in natural language. Therefore, requirements specification inherits subjectivity of natural language. This often leads to common mistakes made by engineer when specifying requirements. These mistakes are known as Meyer's seven sins [1]. Meyer's seven sins indicate that there are seven common mistakes that are often found in requirements document, i.e. noise, silence, over-specification, contradiction, ambiguity, forward reference, and wishful thinking.

Researchers have been working on identifying and dealing, with such mistakes for the last two decades. Ambiguity has been receiving the most attention from researchers [2]–[6]. There are researchers from Stanford [7] who have been working on detecting contradiction between text. Nevertheless, so far less attention has been given to other type of mistakes, aside from the fact that they all are equally important.

Daniel Siahaan is with Informatics Departement, Institut Teknologi Sepuluh Nopember, Surabaya-Indonesia, 60111, Indonesia. e-mail:daniel@if.its.ac.id.

Izzatul Umami with Informatics Departement, University of Darul Ulum Jombang, Jombang-, Indonesia. E-mail: izzatul_umami@yahoo.com

Workflow support refers to the part of the system that allows multiple workers to work on multiple cases in a wide variety of organizational arrangements. The framework must support workflow functions that are managed either by assigning cases to individual **case workers** or to a pool of case workers of the same subtype. The use-case descriptions below will specify when this applies. In the former situation, the cases are assigned centrally by a line manager. In the latter situation, the organization using the framework for developing their case management system has essentially chosen to empower their case workers to partially manage their workload. These case workers will pick a case from the pool to work on next. The manager is depending on their professionalism to ensure that cases do not stay in the pool too long.

At a minimum, the framework should support:

A case work is assigned cases which are managed from start to finish.

A case worker is assigned a case by a manager to perform a specific function for that case, and then informs the manager the task is complete. The manager then determines the next appropriate step (close, assign to another case worker for the next processing step, put aside until a bring-forward is activated, etc.).

Figure 1. Snapshot of problem description p01s04.txt

This work focuses on creating an approach to detect forward reference in requirements specification document. Forward reference refers to a first appearance of a term in passage which precedes its definition. To our knowledge, there has been no previous work that focusing on forward reference detection. Our approach uses natural language processing (NLP) library for capturing terms within a document and determining whether a statement contains a definition of a term. We developed a set of rules which processes metadata of a sentence generated from natural language process to extract terms and identifying definitions.

II. FORWARD REFERENCE

Reference [8] defines forward reference as a state of an element in a document which refers a feature of a solution domain which precedes its definition. It suggests that forward reference in requirements specification document refers to a first appearance of a term in passage which precedes its definition. Let's consider one of problem descriptions from ACM's OOPSLA DesignFest® online source (http://designfest.acm.org/) shown in Figure 1.

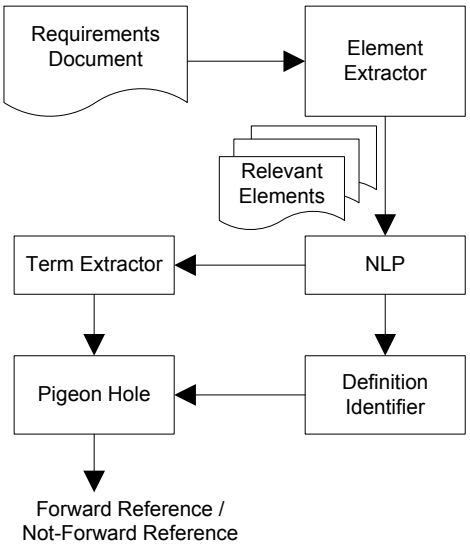

Figure 1. Snapshot of problem description p01s04.txt

We can see that a sentence in line 4 contains a term "case worker". The term "case worker" in the document refers to a role in the respective solution domain. At the point where it is first referred, the term "case worker" has not been described or defined yet. Its description can be found later in line 15. It can be concluded that the sentence in line 4 contains a forward reference.

The goal of this research is to provide an approach to assist requirements engineer in producing a high quality requirements document which is forward reference free. This approach is designed to identify the occurrence of forward reference in software requirements specification document.

III. FORWARD REFERENCE DETECTION

The approach is designed in a number of processes, as shown in Figure 2. First, the requirements specification document is processed using element extractor module to extract relevant elements. Second, a natural language processing module processes each extracted element to generate metadata of each element, such as part of speech, sentence structure, and word dependency. Third, given the metadata, a term identifier module identifies any term resides in an element. Fourth, using the same metadata, a definition identifier module classifies whether an element is a definition and identifies what specific term the element defined. Finally, a pigeon-hole module direct the term found by term identifier module to a list of defined term or a list of undefined term respectively.

A. Element Extractor

A document is composed by one or more set of elements. Each set of element has certain type. In software requirements specification document, type of element may be one of the following, title, section, sub-section, paragraph, table, figure, sub-title, table header, cell, header, footer, and page number. There are a number of document element types which are not considered in forward reference detection. Title, section and sub-section are examples of document element

which often expressed as term or contain one or more terms. The term or terms in this document element are not relevant to forward reference. Our approach considers only paragraph, sub-title, and cell elements for the forward reference detection process. These elements represent the description about software requirements of the solution domain. Aside from the three elements, figure is also describing about software requirements. Nevertheless, our current approach does not consider any term resides in a figure due to the fact that the treatment should be similar to cell element in a table. A module which extract text element from graphical component is necessary to be added.

A paragraph element is a type of element that contains a set of sentence elements. Sentence element is a set of words that compose a sentence. In requirements document, a sentence usually take a form of a statement. One of the sentence elements should be the main idea of the respective paragraph. Each paragraph element is decomposed into sentence elements. A sub-title element is a type of element that indicates what a figure is describing about. The following is an example of sub-titles.

"Fig. 1 Architecture design of rotary lock."

We can see that phrase marked in bold represents term being referred in respective elements. The sub-title contains a term but does not contain a term definition. Cell element is a text that resides in a cell of a table. This may apply to any document element. Element extractor is designed to extract each relevant element in a document and its respective element type. The process ignores irrelevant element, such as titles and sections. Sequentially, these relevant elements are fed to the next process, i.e. Natural Language Processor.

B. Natural Language Processor

Each relevant element is processed using a natural language processor (NLP). This module traverses through the list of elements and generates metadata from each given element. This module uses OpenNLP to produce part of speech (POS) tags, terms, and word-dependencies. For example, consider the following document element e1.

e_1: "*The program's input is a stream of characters whose end is signaled with a special end of text character, ET.*" (source http://www.designfest.org)

Document element e_1 is a sentence element. The NLP uses en-pos-maxent model to generate POS tags out of e_1. The following are the POS tags generated for document element e_1.

The/DT program/NN 's/POS input/NN is/VBZ a/DT stream/NN of/IN characters/NNS whose/WP$ end/NN is/VBZ signaled/VBN with/IN a/DT special/JJ end/NN of/IN text/NN character/NN ,/, ET/NNP ./.

C. Term Identifier

The term identifier chunks the given tagged sentence. It chunks the given tagged sentence into a set of tagged phrases. The following is part of chunking result of e_1.

[NP The program's input/NNP] [VP is/VBZ] [NP a/DT stream/NN of/IN characters/NNS] [NP whose/WP$] [NP end/NN] [VP is/VBZ signaled/VBN] [PP with/IN] [NP a/DT special/JJ end/NN of/IN text/NN character/NN] [NP ET/NNP] [./.]

Each chunk is a candidate term. As already mention, this work only considers chunk with *NP* tag. Therefore, given e_1, the chunking process returns the following terms (after removing any determinant or cardinal): program's input, stream of characters, special end of text character, and *ET*. At the end, NLP removes any commonly known terms using Wikipedia. This last part removes the first two terms and left one term as a result, i.e. program's input and *ET*.

D. Definition Identifier

Like term identifier, definition identifier also consumes tagged sentence produced by NLP. Parallel to term identifier, the definition identifier identifies any definition of a term resides in a document element and decide whether a document element contains a definition of a term. A definition is a clause that explains, formulates, or describes a term. The process determines a clause as a definition base on a set of rules. A rule is a pattern that comprises of a word dependency tree with its given POS tags. The pattern is generated by analyzing a sentence corpus of term definition. We managed to generate 7 patterns for a sentence that contains a term definition. The following is the list of rules to identify a term definition.

NP(NN | NNP) VBZ + VBN+IN
NP(NN | NNP) VBZ + DT+NN
NP(NN | NNP) VBZ
NP(NN | NNP) VBZ + IN
NP(NN | NNP) VBZ + DT+NN+IN+WHNP
VP (VBZ + VBN+IN + NP(NN / NNP))

For example, given the document element e_1, we can see its sentence structure as shown in Figure 3. We can determine that e_1 matches the rule: *NP (NN) VBZ + DT + NN*. Therefore, it can be concluded that document element e_1 is an element that contains a term definition, where the term is the NP-tree ("The program's input").

E. Pigeon Hole

Both previous modules provides input for pigeon hole process. First, for a given sentence element, if it contains a term definition, it adds the respective term into the list of defined terms if and only if it is not listed in the defined term list. Second, for each term found in a document element, it marks the respective element as forward referencing if and only if the term is not listed in the defined term list.

For example, let's assume a list of defined term dt{program's input}. Given the document element e_1, the approach determines that e1 is forward referencing. It is because the term "*ET*" does not exist in dt, which means that it

IV. DISCUSSION

For experimentation purpose, this research collects 30 requirements document from various sources. They are part of different kinds of projects, such as student projects, web-based applications, information system, eBill, games, and embedded system. A non-IT person who has academic background in linguistic was asked to

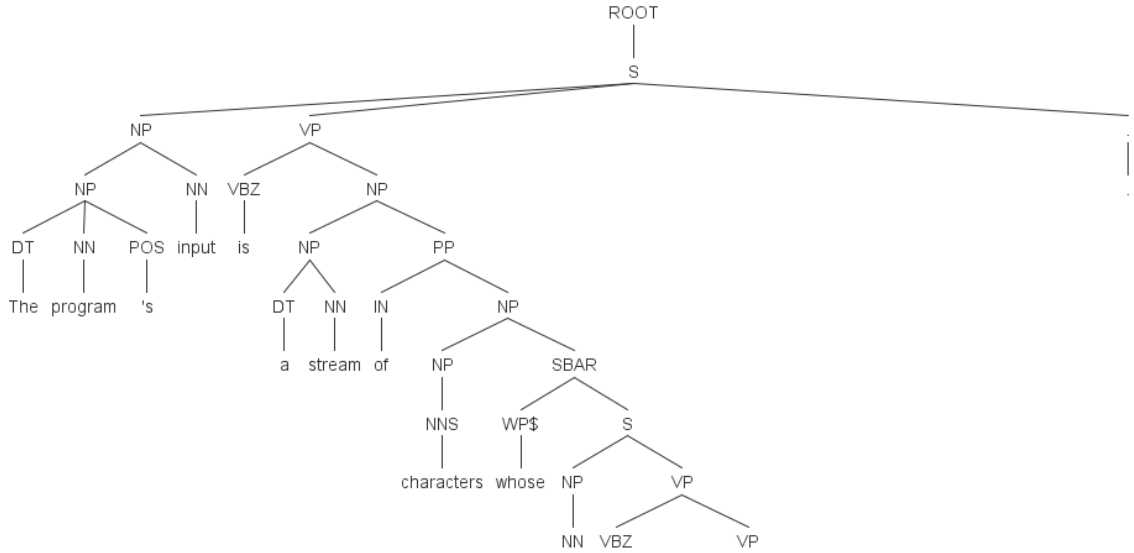

Figure 1. Snapshot of problem description p01s04.txt

TABLE 1.
DEGREE OF AGREEMENT BETWEEN PROPOSED METHOD AND EXPERT.

#	Doc.	Expert		Proposed Approach		Index Kappa
		FR	PR	FR	PR	
1	Doc_01	14	39	15	38	0.952
2	Doc_02	6	36	8	34	0.829
3	Doc_03	4	42	6	40	0.776
4	Doc_04	12	54	14	52	0.904
5	Doc_05	5	23	8	20	0.704
6	Doc_06	4	25	5	24	0.868
7	Doc_07	9	34	10	33	0.932
8	Doc_08	5	27	6	26	0.890
9	Doc_09	13	57	14	56	0.954
10	Doc_10	11	27	12	26	0.937
11	Doc_11	4	28	7	25	0.675
12	Doc_12	4	23	6	21	0.756
13	Doc_13	3	28	3	28	1
14	Doc_14	4	29	6	27	0.765
15	Doc_15	7	20	10	17	0.746
16	Doc_16	5	32	8	29	0.723
17	Doc_17	8	39	9	38	0.928
18	Doc_18	5	23	8	20	0.704
19	Doc_19	8	59	14	53	0.678
20	Doc_20	7	50	11	46	0.645
21	Doc_21	11	47	16	42	0.761
22	Doc_22	4	65	5	64	0.881
23	Doc_23	10	41	12	39	0.884
24	Doc_24	6	54	11	49	0.662
25	Doc_25	2	5	2	5	1
26	Doc_26	5	46	6	45	0.898
27	Doc_27	10	54	14	50	0.796
28	Doc_28	1	25	1	25	1
29	Doc_29	12	46	13	45	0.949
30	Doc_30	8	27	12	23	0.724

identify document elements that contains terms, which are forward referenced (FR) or predefined (PR). To measure the performance of the proposed approach, kappa statistics is chosen [9]. This method is chosen because it can measure how reliable the approach to perform as an expert, in this context is a person who has non-IT background. Table 1 shows the result of our experimentation. It can be calculated that the overall kappa value for 30 documents is 0.828. It can be interpreted base on [10] that there is almost perfect agreement between proposed method and expert in determining which term is forward reference and which term is predefined.

VI. CONCLUSION

Requirements engineers have the responsibility to produce a high quality requirements specification document. The effort to maintain the quality of a requirements specification document manually is relatively big and may take significant resources of software development project. Forward reference is one of the elements that reduce the quality. This research aims to provide an approach to detect any instance of forward reference within a document using natural language processing. The experiment on 30 requirements documents from various domains reveals of software project indicates that the proposed approach has considerably almost perfect agreement with domain expert in detecting forward reference in software requirements document, given 0.83 kappa index value.

REFERENCES

[1] Meyer, B. 1985. On Formalism in Specifications. **IEEE Software**, 2(1), January 1985, 6–26.

[2] Muliawan, I. W. Muliawan and Siahaan, D.O. 2012. Software Requirements Ambiguity Analysis based on SMART Requirements (Analisis Ambiguitas Kebutuhan Perangkat Lunak Berdasarkan Acuan SMART Requirements). In **Manajemen Teknologi Informasi**, *SEMNAS XIV*, Surabaya, Indonesia, 2012.

[3] Hussain, I., Ormandjieva, O., and Kosseim, L. 2007. Automatic Quality Assessment of SRS Text by Means of a Decision-Tree-Based Text Classifier. In **Proceeding of 7th International Conference on Quality Software**, Portland USA, p.209-218.

[4] Gnesi, S., Fabbrini, F. Fusani, M., and Trentanni, G. 2005. An automatic tool for the analysis of natural language requirements. **International Journal of Computer Systems Science & Engineering**, vol. 20(1), pp. 53–62.

[5] Kamsties, E., Berry, D. M., and Paech, B. 2001., Detecting Ambiguities in Requirements Documents Using Inspections. in **Proceedings of the First Workshop on Inspection in Software Engineering (WISE'01)**, pp. 68–80.

[6] Denger, C., Berry, D. M., and Kamsties, E. 2003. Higher quality requirements specifications through natural language patterns. In **Proc. of the IEEE Int. Conf. on Software – Sci. Tech. and Eng**, pp. 80–91.

[7] Marneffe, M.D., Rafferty, A.N., and Manning, C. D. 2008. Finding contradictions in text. In **ACL 2008**.

[8] Siahaan, D. 2012. **Software Requirements Analysis (Analisa Kebutuhan Dalam Rekayasa Perangkat Lunak**. Penerbit Andi.

[9] Cohen, J. 1960. A Coefficient of Agreement for Nominal Scales. **I. Educ. Psycho!**, Meat. 20:37-46.

[10] J. R. Landis, J.R. and Koch, G.G. 1977. The Measurement of Observer Agreement for Categorical Data. **Biometrics**, vol. 33(1), pp. 159–174.