KNOWLEDGE ENGINEERING: PRINCIPLES AND TECHNIQUES
Proceedings of the International Conference on Knowledge Engineering,
Principles and Techniques, KEPT2007
Cluj-Napoca (Romania), June 6–8, 2007, pp. 58–65

A TEXT ANALYSIS BASED APPROACH FOR THE COMPLIANCE BETWEEN THE SPECIFICATION AND THE SOFTWARE PRODUCT

DANA LUPŞA(1) AND ADRIANA TARŢA (2)

Abstract. Nowadays, the success or failure of a software product depends on its quality. An essential component of software quality is its functionality. In this paper we propose a new approach in evaluating the compliance between software documentation (expressed on natural language) and the final software product. We define two evaluation measures and present some case studies.

1. Introduction

The continuous development of computer science and the increased expansion of application areas of software products has raised more and more frequent the question: *What makes a good project?*

According to ISO-9126 [**?**] the factors of the software product quality are functionality, reliability, usability, efficiency, maintainability, and portability. *Functionality* is the ability to satisfy stated or implied needs. It assumes the program is correct. Correctness is strongly connected to good specifications and good design [**?**]. *Reliability* refers to the capability of software to maintain its level of performance under stated conditions for a stated period of time. *Usability* refers to the effort needed to use the software product. *Efficiency* is related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions. *Maintainability* refers to the effort needed to make specified modifications and *portability* is related to the ability of software to be transferred from one environment to another.

Software developers are interested in saving time and costs along with minimizing the risks associated with non-compliance between user needs and final program functionality. Researchers agree that, for achieving these goals, it is necessary to develop and implement an effective standard management solution that

2000 *Mathematics Subject Classification.* 68Q60.
Key words and phrases. software quality, software specification, requirements.

will result in the engineers and project staff actually using the standards and specifications [**?**].

In this paper we propose a new approach in studying the quality of a software product based on the analysis of its functionality. We will acquire knowledge about the software functionality from requirements documents. In order to evaluate the quality of the final product we propose two measures that indicates its compliance to the specification documents.

The paper is structured as follows: Section 2 gives a short presentation of the specification documents. Section 3 presents some approaches in the domain of writing good requirements specification. Section 4 proposes an original view on the automatic analyse of the compliance between the requirements and the final product. Section 5 describes some real case studies. The paper ends with conclusions and identifies some key issues for future work in this area.

2. Specification documents

To create a software application, the description of the problem and the requirements are needed, i.e. what the problem is about and what the system must do. Specification documents are the results of an investigation of the problem rather than how a solution is defined [**?**]. One important principle that must be followed when developing a complex software system is: *Think first, program later* [**?**]. The first step in applying this principle consists of defining the problem completely [**?**], [**?**]. A good software specification is the first step toward a successful software product.

The description of the problem is usually called the *problem specification*. It contains a short description of where the user needs support from the program. It is usually informal and it can be considered as a blueprint for the problem analysis.

Requirements are description of needs or desires for a product. The primary goal of the requirements phase is to identify and document what is really needed. The documents must be easily understood by the clients and the development team.

Requirements are typically classified into three categories:

- functional requirements - describe system features or what the system must do;
- non-functional requirements - describe properties the system must have (e.g. performance, availability, accessibility);
- constraints - limits the development in some way. For example, a constraint can be the operating system the system must run on, or the programming language that must be used to implement the system.

The description of functional requirements is named *functional specifications* (FS).

Essential features of requirements are:

- Necessary - contains elements that must be included and for which other system components will not be able to compensate.
- Unambiguous - susceptible to only one interpretation.
- Concise - stated in language that is brief and easy to read, yet conveys the essence of what is required.
- Consistent - does not contradict other stated requirements nor is it contradicted by other requirements. In addition, the specification must use terms that have the same meaning in all statements of the documents.
- Complete - stated entirely in one place and in a manner that does not force the reader to look at additional text to know what the requirements means.
- Reachable - a realistic capability that can be implemented for the available money, with the available resources, in the available time.
- Verifiable - must be able to determine that the requirements have been met through one of four possible methods: inspection, analysis, demonstration, or test.

During analysis review, comparison of application domain model with client's reality may result in changes to each. Specifications are most important for external interfaces that must remain stable [**?**]. One of the main problems of requirements elicitation is expressing customer requirements in a form that can be understood not only by requirements engineers but also by noncomputer professional customers and users. The usual choice for expressing elicited requirements is natural language, since it is frequently the only common language to all participants.

Our approach, presented in Section 4, analyses the correspondence between specification documents, expressed in natural language, and the visible linguistic components of the final product, which is the user interface. We propose two measures that indicate the degree of consistency between specifications and the user interface (UI). We have applied our measures to problem specifications, functional specification and user manuals (UM) and we will present a short comparison of the results. We focus on functional specification because it is the most detailed specification document.

3. Related work

The previous work in the domain of writing good requirements specification focuses on finding solutions to build the appropriate requirements based on some automated processes or some patterns. Two approaches will be presented in the following.

In [**?**], the authors present requirements templates that can improve requirements elicitation and expression. They use two categories of patterns: linguistic patterns (very used sentences in natural language requirements descriptions), and

requirements patterns (generic requirements templates that are found very often during the requirements elicitation).

In [**?**] the authors present a system that automatically verifies some desired quality properties of software requirements, for example the unambiguity and completness. Their approach is based on the representation of software requirements in XML and the usage of the XSLT language .

As far as we know, there is no approach that investigates the correspondance between software specification documents and the software product, based on text processing.

4. Our approach

We propose a method to evaluate the compliance between requirements specification and the final software product. Our approach deals with specifications in natural language.

A good functional requirements specification is a key step towards a successful software product. Obviously, this is true only if the software product follows the specification. We propose two measures to evaluate the correlation between the specification in natural language and the "natural language" part of a software product, which is the user interface. They are useful to verify the degree to which a software product respects specifications (in the case of a good quality specification) or to evaluate the completeness of a specification (in the case of a program that meets the quality standards).

According to [**?**], a functional specification is "a formal document used to describe in detail for software developers a product's intended capabilities, appearance, and interactions with users". This means that the words chosen to appear in the user interface (UI) illustrate the concepts described in functional specification. The compliance between the specifications and the UI grows with the number of common words. Based on this idea, we propose two measures: $\mathcal{CW}$ and $\mathcal{CWT}$, that are based on the number of words that appear both in the UI and in the specification. Their values are scaled in order to obtain real values from the interval $[0, 1]$. They reach the maximum (value 1) when all the words from the UI are present in the specification document . This is also the ideal case.

The first measure, denoted by $\mathcal{CW}$, is based on the number of *Common Words* from FS and UI.

$$\mathcal{CW} = \frac{common - words}{words - in - the - user - interface}$$

The second measure, denoted by $\mathcal{CWT}$, is based on counting the *Common Words Truncated to first k letters*:

$$\mathcal{CWT} = \frac{first - k - letter - from - common - words}{first - k - letter - from - words - in - the - user - interface}$$

From all the specifications, the functional specification contains the most detailed description of the application functionalities. It is the most related to the user interface. That is why we consider that the two measures are appropriate to evaluate the compliance between FS and UI.

The problem specification makes a short description of user needs. Parts of the problem specification should be reflected on the UI, if the implementation is compliant with the problem specification. The concepts presented in the problem specification are expected to be present in the UI. We expect that, in this case, the values of $\mathcal{CW}$ and $\mathcal{CWT}$ will be lower than in the case of the functional specifications.

The user manual (UM) describes and explains all the items in the UI. We consider that the $\mathcal{CW}$ and $\mathcal{CWT}$ measures should also be a good indicator of the quality of the UM document.

In the next section a detailed study of some functional specifications is described. A comparative view of the correspondance between problem specification, functional requirements and user manual, on one side, and the user interface, on the other side, is also presented.

5. The Experiments

We have studied the $\mathcal{CW}$ and $\mathcal{CWT}$ measures for four cases of applications developed for the points based jobs evaluation method. The main functionalities of the applications are: job management, job evaluation sessions management, job evaluation factors configuration and job evaluation results management. The applications were developed using Borland Delphi 7 environment.

The words from the user interface were automatically extracted. A Java application was developed for this purpose. It parses the **.dfm* files from the project and extracts the words associated to the widgets displayed on the user interface. The output generated by this application is a file containing the words extracted from the user interface.

For those applications, we have one problem specification and four triplets of functional specification, user interface and user manual. All documents were written in Romanian.

Truncation is usually used for languages for which a stemmer is missing. The number of characters the word is truncated to is the truncation parameter. A common practice for choosing the truncation parameter is to use a value smaller, but close to, the average length of the words in that language. In our case, we have extracted the Romanian words from the freely available Romanian-English dictionary from [**?**]. The average length of the words from this dictionary is 7.65. We choose the value 6 for the truncation parameter.

FS	$\mathcal{CW}$	$\mathcal{CWT}$
FS1	0.31	0.47
FS2	0.73	0.86
FS3	0.77	0.82
FS4	0.77	0.80

TABLE 1. Values of $\mathcal{CW}$ and $\mathcal{CWT}$

5.1. **FS and UI Compliance.** We have computed the $\mathcal{CW}$ and $\mathcal{CWT}$ measures on four good quality functional specifications. We have chosen k=6 for evaluating $\mathcal{CWT}$. The results are presented in Table 1.

We expect that $\mathcal{CWT}$ makes a more accurate estimation of the common concepts because it approximates better the number of common word roots. In order to determine the origin of the differences, we focus on the functional specification that has the lowest $\mathcal{CW}$ and $\mathcal{CWT}$ scores. We have considered the words that appear in the UI but are missing from the functional specification. We classified them in two sets as presented in Table 2. The words in the first set can be characterized as being specific for working with computers and they do not describe the functional logic of the application. From this point of view, their missing is insignificant. The second set contains more meaningful words. If we take a look at the functional specification, we can see that, for most of them, different words derived from the same word root appear. For example, we can not find the words like *expert (expert)*, *evaluatori (evaluators)*, but words like *expertii (the experts)*, *evaluatorii (the evaluators)* are present. This issue is solved by using the second proposed measure, $\mathcal{CWT}$.

Set	**Words**
Set 1	*accepta (accept), adauga (add), adaugare (addition),* *iesire (exit), intra (login/enter), salvare (save), selectate (selected),* *selectati (select), sterge (delete), stergere (deletion), ...*
Set 2	*expert (expert), evaluatori (evaluators), ...*

TABLE 2. Some words that appear in the UI and do not appear in FS

The values computed for $\mathcal{CWT}$ measure increase because the number of differences decreases. For example, in this case, for two separate words present in UI: *sterge (delete)* and *stergere (deletion)*, $\mathcal{CWT}$ (with $k = 6$) "sees" only one truncated word component, that is *sterge*.

A possible drawback of the $\mathcal{CWT}$ measure is generated by the truncation to a fixed number of characters. It is possible that two non-related words to be truncated to the same k characters. We manually verified our data and we found that there are no such situations.

5.2. **Problem Specification, FS, UM and UI Compliance.** We have evaluated the $\mathcal{CW}$ and $\mathcal{CWT}$ measures for a set of: one problem specification, four functional specifications (FS) and four user manuals (UM). For $\mathcal{CWT}$, we have chosen the parameter $k = 6$.

The results are presented in Figure 1. The first bar set indicates the values of $\mathcal{CW}$ scores, and the second bar set indicates the values of $\mathcal{CWT}$ scores. Each set represents the $\mathcal{CW}$ or $\mathcal{CWT}$ scores for program specification, functional specification and user manual. For FS and UM we have represented the minimum and maximum values of $\mathcal{CW}$ and $\mathcal{CWT}$ obtained for the four specifications. Minimum is depicted in dark grey and maximum in light grey.

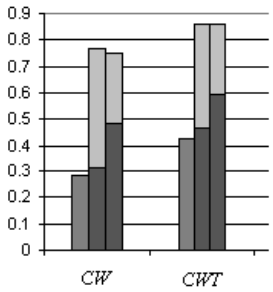

FIGURE 1. A comparative view over the values of $\mathcal{CW}$ and $\mathcal{CWT}$ for problem specification, FS and UM

Problem specification is a more general specification while functional specification is a more detailed one. That is why we expect lower values for $\mathcal{CW}$ and $\mathcal{CWT}$ for the problem specification.

Usually, the user manual refers to every detail in the UI. That is why the highest scores for $\mathcal{CW}$ and $\mathcal{CWT}$ are expected to be achieved for UM. On the other hand, a UM can contain screen shots from the application, that is why it is possible that the text in the UM does not contain every word from the UI.

The results depicted in Figure 1 confirm our expectations: the lowest values for $\mathcal{CW}$ and $\mathcal{CWT}$ are achieved for the problem specification and the highest ones for the functional specification and the user manual.

6. Conclusions and further work

A good functional requirements specification is a key step towards a successful software product. In this article, we have proposed two measures, $\mathcal{CW}$ and $\mathcal{CWT}$, to evaluate the compliance between product specification in natural language and the user interface. In our experiments, we have obtained high scores for some good quality specifications.

In the future, our intention is to improve the measures we have introduced in this paper. The starting point could be the use a stemmer instead of a blind truncation of words to a fixed number of characters. We also plan to further investigate the relations induced by the presence of the concepts, not only by the presence of the words. This can be done using some sort of semantic analysis.

(1) Babeş-Bolyai University, Str. M. Kogălniceanu nr. 1, Cluj-Napoca, Romania
E-mail address: dana@cs.ubbcluj.ro

(2) Babeş-Bolyai University, Str. M. Kogălniceanu nr. 1, Cluj-Napoca, Romania
E-mail address: adriana@cs.ubbcluj.ro