Implementing Case-Based Reasoning Technique to Software Requirements Specifications Quality Analysis

Dr. Anil Kumar Mishra[1], M.P. Mishra[2]
[1] *Computer Science & Engineering, Gandhi Engineering College(GEC, Bhubaneswar), Odisha*
[2] *Computer Science & Engineering, Gandhi Engineering College(GEC, Bhubaneswar), Odisha*

Abstract

Software Requirements Specifications (SRS) or software requirements are basically an organization's interpretation of a customer's system requirements and dependencies at a given point in time. Basically, good quality SRS will lead to good quality software product. It is widely known that companies pay much less to fix problems or defects that are found very early in any software development life cycle (SDLC). In this study, the Software Quality Assurance (SQA) audit technique is applied to determine whether or not the required standards and procedures within the requirements specifications phase are being followed closely. The proposed online SRS quality analysis system ensures that software requirements among others are complete, consistent, correct, modifiable, ranked, traceable, unambiguous, and understandable. The system interacts with the developer through a series of questions and answers session, and requests the developer to go through a checklist that corresponds to the list of desirable characteristics for SRS. The Case-Based Reasoning (CBR) technique is used to evaluate the requirements quality by referring to previously stored software requirements quality analysis cases (past experiences). CBR is an AI technique that reasons by remembering previously experienced cases. It assists in making the SRS quality analysis process more efficient. An executable prototype is developed to demonstrate several selected features and results of the proposed SRS quality analysis system.

Keywords*: Case-based Reasoning, Quality Analysis, Software Requirements Specifications, Software Development Life Cycle*

1. Introduction

The Software Requirements Specifications (SRS) document states all those functions and capabilities a software system must provide, as well as states any required constraints by which the system must abide.

By definition, a requirement is an objective that must be met, while a specification describes how the objective is going to be accomplished [1]. In other words, a specification document describes how specific tasks are supposed to be done. A very critical part of the quality assurance role is proactive involvement during the system's requirements specifications phase.

In the past, several studies have determined that companies will have to pay less to fix problems that are found early in any Software Development Life Cycle (SDLC) [1]. Fig. 1 gives an idea of the cost of change [2] for changes made at different phases of the SDLC.

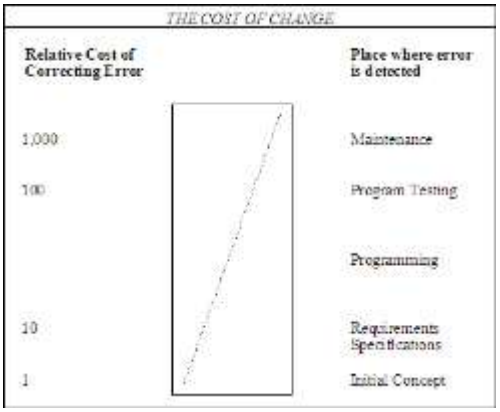

Figure1. The cost of change at various phases of the SDLC [2]

In order to determine the quality of the software development process, a list of quality attributes that software requirements specifications are expected to exhibit need to be compiled [3]. Several desirable characteristics for requirements specifications that have been identified in previous researches are as follows [3][4][5][6]:

- *Complete*: Complete requirements specifications must clearly define all real life situations and include all the necessary capability features.
- *Consistent*: Consistent requirements specifications must not have any conflicting statements among them.
- *Correct*: A correct requirement specification must accurately and precisely identify the individual conditions and limitations of all cases that the desired capability will encounter and it must also properly define the capability's response to those cases.
- *Modifiable*: Related requirements specifications must be grouped together in order to be modifiable, while unrelated requirements specifications must be separated.
- *Ranked*: Requirements specifications must be ranked according to stability or/and importance.
- *Traceable*: Each requirement specification must be uniquely identified to achieve traceability. Uniqueness can be achieved by the use of a consistent and logical scheme for assigning identification to each specification statement within the requirements specifications document.
- *Unambiguous*: Each requirement specification can only be interpreted in one way. The use of weak phrases or poor sentence structure must be avoided.
- *Understandable*: A requirement specification is understandable if the meaning of each of its statements is easily grasped by the readers.
- *Testable*: A testable requirement specification is the one that is in a manner that pass/fail or quantitative assessment criteria can be derived from the specification itself.
- *Verifiable*: In order to be verifiable, each requirement specification at one level of abstraction must be consistent with those at another level of abstraction.
- *Validatable*: A valid requirement specification is the one that has been analyzed, understood, accepted, validated and approved by the project participants, managers, engineers and customer representatives.

The above characteristics are closely related to each other, and some cannot exist without the others. During the analysis and audit review of the requirements specifications, several primitive indicators that provide some evidence that the desired attributes are present or absent are being linked to the above quality attributes.

2. Literature review

Several papers have been written on software requirements quality analysis, and a few them are discussed below.

Knauss and El Boustani [7] defined their own model in assessing the software requirements specifications quality based on the Goal-Question-Metric (GQM) method. They analyzed more than 40 software projects using this method. Their main goal was to assess the quality of a Software Requirements Specifications (SRS) document and to connect it to the corresponding project success. Project success was measured based on the answers given to several questions related to the project's results.

Heck and Parviainen [8] presented a method called LSPCM (The LaQuSo Software Product Certification Model) that was developed for certifying software product quality [9]. Here, they described their experiences from using this method for analyzing requirements quality in different cases (systems), which are the Central Registration System, Counter Automation Solution, and Embedded Systems Case. They showed that the checks in LSPCM were able to discover inconsistencies in requirements specifications, regardless of the

application domain.

Another research work on quality analysis of the SRS was conducted by Dang Viet Dzung and Atsushi Ohnishi [10] in which they established a method of checking a SRS using requirements ontology. A set of rules that were classified into general rules and domain specific rules were used in determining the correctness and completeness of the SRS.

Firesmith [11], in his paper, proposed a comprehensive questionnaire that can be used when specifying and technically assessing software requirements.

An inspection technique to assess SRS called "Specification Quality Gate" (QG-Spec) was introduced by Salger et al. [12]. QG-Spec is an approach targeted for early and comprehensive evaluation of SRS. It has been applied to a series of large scale commercial projects. The authors discussed which quality attributes are effectively assessed by using the QG-Spec technique.

Other related research on software requirements analysis, performance, and system optimization that are useful in improving SRS quality analysis techniques are covered in the following papers.

Chaczko et al. [13] in their paper proposed the development of software requirements analysis CASE tools that can be used in cross time-zone development projects. Its main objective was to investigate the suitability of a data/knowledge-transfer model for the development of a software requirements analysis CASE (SRAC) tool.

A prediction model-driven engineering to provide relevant analyses and also to predict performances of distributed systems throughout of the whole software development cycle was proposed by Jing and Jiang [14] that aimed at assuring quality of service (QoS) requirements. This can avoid any delays in the discovery of performance problems so that problems can be solved at early stage.

Ali [15] presented a combined "optimization procedures" by introducing the dynamic programming procedure and artificial intelligence (AI) techniques, which "were mixed together for finding a smart simulator, packed in one program (i.e. expert system) [15]". Here, the rule-based expert system called "CADION ANALYZER" was used to obtain the desired results. This combined method may be useful when dealing with systems that require the use of rules and past knowledge (knowledge base).

3. Research objectives

The objectives of the research study are as follows:

- To propose and design an online system that automates the quality analysis of Software Requirements Specifications (SRS).
- To research on the possibility of applying the Case-Based Reasoning (CBR) AI technique to the Online Software Requirements Specifications Quality Analysis system using the evolutionary prototyping software development method.
- To develop a prototype of the proposed SRS quality analysis system (SRSQAS).

4. The proposed SRS quality analysis system

In order to simplify the process of ensuring that the requirements specifications fulfill the desired software quality standards, the online system will request the software developer to indicate whether each requirement specification has fulfilled the desired characteristics based on the specified quality indicators.

The online quality analysis for requirements specifications is only meant for checking whether or not the system developer has followed certain standards and procedures, and it is not a tool to actually audit the contents of the requirements specifications documentation. In other words, it is only useful as guidance to software quality assurance.

In a previous work in [6], a simple online quality analysis system that measures the quality of the requirements specifications phase's results of the SDLC was developed. Since the online software requirements quality analysis is going to follow the same structure, except for the part on the application of the Case-Based Reasoning (CBR) technique when analyzing the requirements' quality, the descriptions of the system's concept are reproduced here.

Basically, this online quality analysis system poses a series of questions to the system developer based on the relationship between the requirements specifications' quality attributes and the relevant quality indicators for each quality attribute. Table 1 summarizes the relationships between requirements specifications' quality attributes and categories of quality indicators [3][6].

Table 1. Relationships between requirements specifications' quality attributes and categories of quality indicators

INDICATORS OF QUALITY ATTRIBUTES											
Categories of Quality Indicators	Quality Attributes										
	1. Complete	2. Consistent	3. Correct	4. Modifiable	5. Ranked	6. Testable	7. Traceable	8. Unambiguous	9. Understandable	10. Validatable	11. Verifiable
1. Imperatives	X			X			X	X	X	X	X
2. Continuances	X			X	X	X	X	X	X	X	X
3. Directives	X		X			X		X	X	X	X
4. Options	X					X		X	X	X	
5. Weak Phrases	X		X			X		X	X	X	X
6. Size	X					X		X	X	X	X
7. Text Structure	X	X		X	X		X		X		X
8. Spec. Depth	X	X		X			X		X		X
9. Readability				X		X	X	X	X	X	X

The online SRS quality analysis system will request the developer (preferably the project manager or SRS reviewer) to go through a checklist that corresponds to the list of desirable characteristics for requirements specifications. The user (e.g. project manager) of the system must honestly provide correct information regarding the quality indicators of each quality attribute. The online quality analysis system will keep track of the user's responses and will display the audit results at the end of the session. The summary will include the percentage of conformance to the Software Quality Assurance (SQA) standards and procedures and also the list of nonconformance attributes along with the categories of quality indicators that have not yet been fulfilled. If the percentage of conformance is satisfactory, then the requirements are considered to have met the minimum requirement of the SQA standards.

The proposed system will use the typical software quality measurement method in which quality is measured with a weighted sum of criteria measurements [16]. The following steps are used in measuring a quality attribute/factor of a software entity, and these steps are adopted in this paper, with minor modifications to the original method, into the previous online quality analysis system in [6].

The following gives a list of steps (algorithm) that is used to measure software quality for the proposed online quality analysis system [6]:

Step 1: Select quality indicators categories to measure each software quality attribute.
Step 2: Select a weight w for each quality indicators category (usually 0 <= w <= 1; depends on the number of quality indicators categories that correspond to a particular software quality attribute).
Step 3: Select a scale of values for quality indicators categories scores (1 – 5, where 5 is the highest).
Step 4: Select minimum and maximum target values for each quality indicator category score (here, the value 3 is set as the minimum, and the value 5 as the maximum).
Step 5: Select minimum and maximum target values for the software quality attribute score (here, the value 3 is set as the minimum, and the value 5 as the maximum).
Step 6: Give each quality indicators category score (entered by the user).
Step 7: Compute a weighted sum.
Step 8: Compare the weighted sum with the preset min-max software quality attribute scoring range.
Step 9: If the weighted sum is outside the min-max scoring range, compare each individual quality indicators category score with the preset min-max criterion score range to direct software improvement activities (all this information will be displayed in the audit report).

The weighting formulas for each software quality attribute in the quality measurement framework have the form $w_1c_1+w_2c_2+ \ldots + w_nc_n$, where $w_1,..,w_n$ are weights and $c_1,\ldots,c_n$ are quality indicators categories measurements. A weighting formula measures the aggregative effect of weighted quality indicators categories [6].

The only modification that is introduced in this paper to improve the proposed requirements quality analysis system is by applying Case-Based Reasoning (CBR) in producing the most suitable set of solutions to the evaluated SRS [18].

The CBR technique is used to evaluate the requirements quality by referring to previously stored software requirements quality analysis cases (past experiences). CBR is an AI technique that reasons by remembering

previously experienced cases. Within this research study, the CBR is used to evaluate the requirements information (quality attributes & indicators) provided by the user, and give the corresponding quality analysis results along with a proposed most suitable solution to any problems related to the quality of the software requirements provided by the user [18]. In CBR, there are four main steps (CBR cycle), which are retrieve, reuse, revise, and retain.

The following gives brief descriptions of the steps [17]:

- Step 1 - *Retrieve*: Retrieve the most similar case or group of cases.
- Step 2 - *Reuse*: Reuse the information, knowledge, and solution in that case to solve the problem at hand if there is a perfect match.
- Step 3 - *Revise*: Revise and adapt the most similar case or group of cases as appropriate if a perfect match is not found.
- Step 4 - *Retain*: Retain or save the new experience or case for future retrievals and problem solving, and the case base is updated by saving the newly learned case.

Within this research study, a case is an example of a previously experienced software requirements quality analysis result that is stored in a file (that can be considered as a knowledge base or case base). When a new case (a new requirements quality analysis) is evaluated, previously stored cases are retrieved from the case base and are used to come up with the quality analysis results for the new case [18]. If the new case (entered case) has been experienced before, the previously stored similar (exactly the same) case is retrieved and reused directly (Step 1 and Step 2 of the CBR cycle). This will save a lot of time. If the new case has not been experienced in the past, the most similar case or a group of most similar cases is retrieved and revised (Step 3), and the newly experienced case is retained/stored (Step 4) inside the knowledge base file.

Fig. 2 presents a diagram of the CBR cycle [17][18].

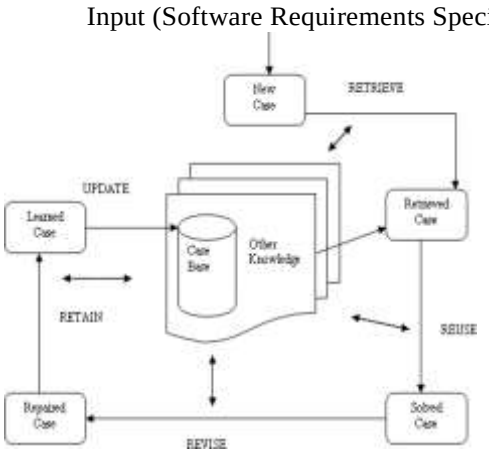

Figure 2. The CBR cycle

Basically, the input to the CBR system within the proposed online software requirements quality analysis system is a set of software requirements characteristics for a specific software project. The CBR technique is used to evaluate the requirements quality and proposes some suggestions [18].

At the end of the session, the quality analysis audit report will be displayed. It indicates which quality attributes are not meeting the SRS standards, and which categories of quality indicators within a quality attribute are unsatisfactory (do not meet the minimum standards), and hence need to be improved. Along with the report, suggestions for improving the quality of the software requirements are given. These suggestions are the results of applying the CBR technique to the SRS quality analysis system.

5. Implementation and the prototype

An executable prototype of the proposed SRS Quality Analysis System (SRSQAS) has been successfully implemented and tested. Its implementation and sample user interface along with the results are described in the following sub-sections.

Implementation

The prototype is implemented in Java using the NetBeans IDE. Evolutionary prototyping object- oriented development methodology is applied to ensure that the system can be easily modified and maintained in order to accommodate new standards and requirements. The system (SRSQAS) is developed incrementally,

component-by-component. Each component is represented by an object class that contains several related methods to perform the required tasks.

SRSQAS interacts with the user who is either a project manager or a reviewer through a set of user interface frames where each frame presents the quality indicators for a particular quality attribute that must be rated by giving the appropriate scores. As mentioned earlier, there are altogether 11 quality attributes that must be assessed, and each quality attribute relates to at least 2 out of the 9 categories of quality indicators.

Based on the given steps (algorithm) in Section 4, the scores that need to be assigned to the corresponding categories of quality indicators range from 1 to 5, where 5 is the best and 1 is the worst. A score of 3 is considered satisfactory, and it is the minimum acceptable score for each quality indicator (under each quality attribute) and also the minimum average score for each quality attribute. The overall average (within the overall results) must also be at least 3, or 60% if converted to a percentage.

Once the new case evaluation is completed, the CBR technique is applied to come up with the most suitable set of solutions to the evaluated SRS. The overall results of evaluating the new case are compared with the existing cases' solutions or reports (within the case base), and based on the 4 steps of the CBR cycle as described above, the most optimal solution or suggestion (a complete report) is presented to the user. The report is to be used as a guideline in improving the SRS document under evaluation.

The prototype: results

Several screen captures of the user interface and results of the proposed SRS quality analysis system are presented below.

Fig. 3 shows the main page of the system, which displays options that are available to the user. The user must provide the necessary information for the project under evaluation, and this is later considered as the identification information for the new case to be evaluated.

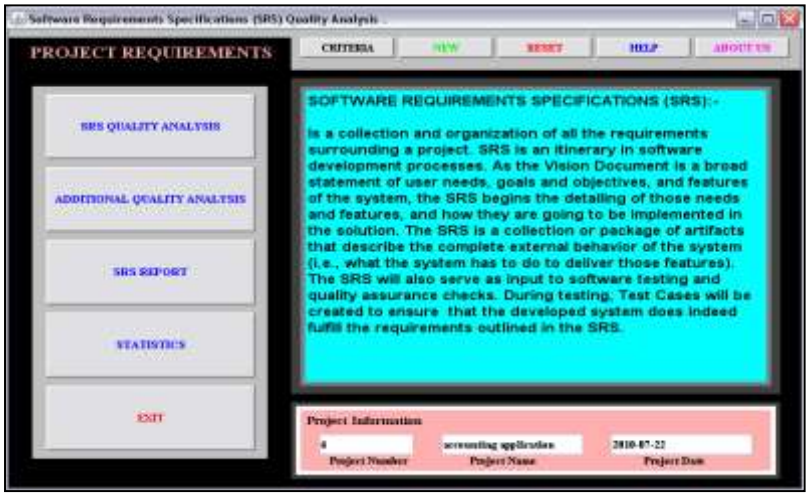

Figure 3. SRSQAS main page

Fig. 4 gives the first frame for getting the evaluation scores for all 8 quality indicators under the "Complete" quality attribute (based on Table 1), which are *Imperatives*, *Continuances*, *Directives*, *Options*, *Weak Phrases*, *Size*, *Text Structure*, and *Specification Depth*.

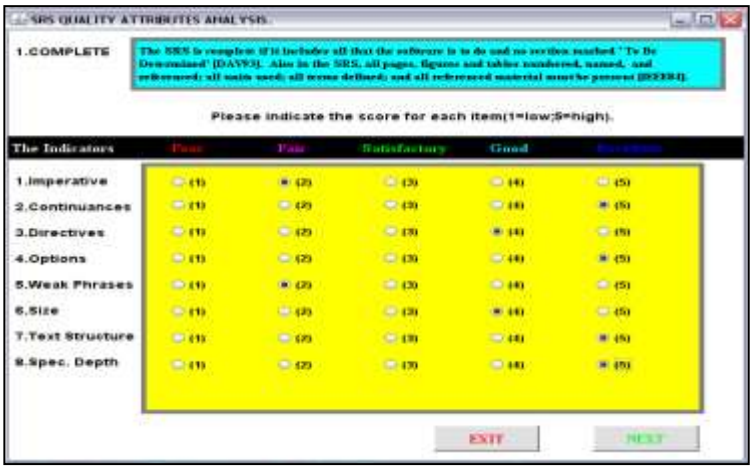

Figure 4. SRSQAS input frame for the "Complete" quality attribute

Finally, Fig. 5 presents the overall results of evaluating the new case (SRS of the current project). The results are presented in tabular form. These results are used by the CBR component within the SRSQAS in deriving the most appropriate report or the most optimal suggestion for improving the quality of the evaluated software requirements.

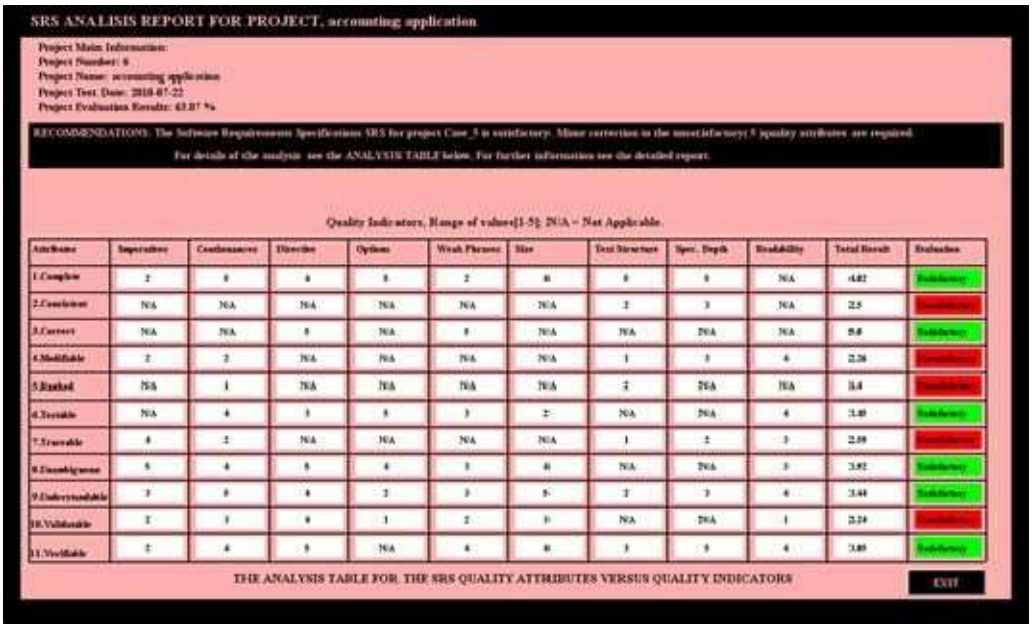

Figure 5. SRSQAS new case evaluation results

6. Future work and conclusion

Once the implementation of the proposed online Software Requirements Specifications (SRS) Quality Analysis system is fully-realized, methods and approaches that use other artificial intelligence techniques such as the rule-based reasoning, fuzzy logic, fuzzy rules, neural network, fuzzy-genetic algorithm and etc. can be considered for future implementations. Currently, a prototype of the system using both fuzzy logic and neural network techniques are under development.

In conclusion, it is believed that this research idea can contribute significantly in improving the SRS quality analysis process because the Case-Based Reasoning (CBR) technique is able to provide solutions to a software requirements quality analysis problem really fast. CBR allows the system to reuse past cases or experiences in order to come up with quick suggestions to newly posed SRS quality analysis problems or cases without having to reconstruct solutions from scratch for cases that have been encountered many times in the past. This technique will definitely improve the SRS quality analysis process tremendously.

7. References

[1] "Requirements and Specifications", Retrieved 30 Dec 2010, http://www.philosophe.com/design/requirements.html.
[2] Treasury Board of Canada Secretariat, "Systems Under Development (Audit Guide)", Retrieved 31 Dec 2010, http://www.tbs-sct.gc.ca/pubs_pol/dcgpubs/TB_H4/systems-systemes03_e.asp.
[3] W.M. Wilson, L.H. Rosenberg, and L.E. Hyatt, "Automated Quality Analysis of Natural Language Requirement Specifications", Retrieved 30 Dec 2010, http://satc.gsfc.nasa.gov/support/PNSQC_OCT96/pnq.html, 1996.
[4] R. Japenga, "How to Write a Software Requirements Specifications", Retrieved 1 Jan 2011, http://www.microtoolsinc.com/Howsrs.php, 2003.
[5] J.F. Peters, and W. Pedrycz, Software Engineering: An Engineering Approach, John Wiley & Sons, Inc., 2000.
[6] H. Mat Jani and A. Lee, "Online Quality Analysis of the Requirements Specifications Phase of the Software Development Cycle", In Proc. of the 7th Annual SEAAIR Conference (SEAAIR 2007), pp. 166-179, 2007.
[7] E. Knauss and C. El Boustani, "Assessing the Quality of Software Requirements Specification", In Proc. of the 16th IEEE International Requirements Engineering Conference", pp. 341-342, 2008.
[8] P. Heck and P. Parviainen, "Experiences on Analysis of Requirements Quality", In Proc. of the Third International Conference on Software Engineering Advances, pp. 367-372, 2008.
[9] P. Heck and M. van Eekelen. The LaQuSo Software Product Certification Model, CS-Report 08- 03, Technical University Eindhoven, 2008, Cited in Heck and Parviainen, "Experiences on Analysis of Requirements Quality".
[10] V.D. Dang and A. Ohnishi, "Improvement of Quality of Software Requirements with Requirements Ontology", In Proc. of the Ninth International Conference on Quality Software, pp. 284-289, 2009.
[11] D. Firesmith, "Specifying Good Requirements", Journal of Object Technology, Vol. 2, No. 4, pp. 77-87, July-August, 2003, Available at http://www.jot.fm/issues/issue_2003_07/column7/.
[12] F. Salger, G. Engels, and A. Hofmann, "Inspection Effectiveness for Different Quality Attributes of Software Requirement Specifications: An Industrial Case Study", In Proc. of the 7th ICSE Workshop on Software Quality (WoSQ 09), IEEE Press, pp. 15-21, 2009.
[13] Z. Chaczko, J. Quang, and B. Moulton, "Knowledge Transfer Model for the Development of Software Requirements Analysis CASE Tools to Be Used in Cross Time-Zone Projects", International Journal of Digital Content Technology and Its Applications (JDCTA), Vol. 4, No. 1, pp. 10-15, 2010.
[14] S. Jing and C.J. Jiang, "An Approach to Predict Performance of Component-based Software with the Palladio Component Model and Stochastic Well-formed Nets", International Journal on Advances in Information Sciences and Service Sciences (AISS), Vol. 2, No. 1, pp. 31-42, 2010.
[15] F.A. Ali, "Expert System Design of Two Electrostatic Lenses Column by Mixing Dynamic Programming and AI Techniques", International Journal of Advancements in Computing Technology (IJACT), Vol. 2, No. 5, pp. 66-74, 2010.
[16] T.P. Bowen, G.B. Wigle, and J.T. Tsai, "Specification of Software Quality Attributes: Software Quality Evaluation Guidebook", Technical Report RADC-TR-85-37, Vol. II, Rome Air Development Center, Griffins Air Force Base, NY 13441-5700, 1985.
[17] A. Aamodt and E. Plaza, "Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches", AI Communications, IOS Press, Vol. 7, No. 1, pp. 39-59, 1994.
[18] H. Mat Jani, "Applying Case-Based Reasoning to Software Requirements Specifications Quality Analysis System", In Proc. of the 2nd International Conference on Software Engineering and Data Mining (SEDM 2010), IEEE/AICIT, pp. 140-144, 2010.