A Model for Enhancing Requirements Traceability and Analysis

Ahmed M. Salem
Department of Computer Science, California State University, Sacramento
Sacramento, CA 95819
916-278-3831 – Fax 916-278-6774
salema@ecs.csus.edu

Abstract—**Software quality has been a challenge since the inception of computer software. Software requirements gathering, analysis, and specification; are viewed by many as the principle cause of many of the software complex problems. Requirements traceability is one of the most important and challenging tasks in ensuring clear and concise requirements. Requirements need to be specified and traced throughout the software life cycle in order to produce quality requirements. This paper describes a preliminary model to be used by software engineers to trace and verify requirements at the initial phase. This model is designed to be adaptable to requirement changes and to assess its impact.**

Keywords- Requirements Traceability; Software Faults; Software Quality.

I. Introduction

Consistent and traceable software requirements are critical elements in today's complex software projects. Requirements include business requirements, user requirements, functional requirements, non-functional requirements, and process requirements. It is well documented such that most of the errors in software development occur in the requirements phase. With the complexity of software systems and the interdependencies of requirements, requirement traceability models and tools become very critical for improving software fault detection and the overall software quality.

Requirements traceability can be defined as the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases. The requirements traceability is a characteristics of a system in which the requirements are clearly linked to their sources and to the artifacts created during the system development life cycle based on these requirements. [10]

It provides an efficient method for the detection of software faults, which are the static defects that occur due to an incorrect state or behavior of the system. Through traceability we can track which part of the code is linked to the requirements and which is not, this helps us remove discrepancies if any. These discrepancies if not detected can be really expensive at later stages and can lead to faults and failures.

The benefits of requirements traceability are the most obvious when the system changes. When high-level requirements change, it is implied that lower-level objects need to be changed. This issue alone justifies the need for requirements traceability. Testing and software quality also benefit greatly from requirements traceability. If a low-level requirement should fail during requirements testing, the software engineer would know which high-level requirements will not be met. Furthermore, if there is a defect, all of the segments that will be affected based on the requirements traceability can be identified, documented and reviewed.

II. Background

Requirements engineering has played a vital role in the development of software in recent years. Ever since, requirements traceability has become an important issue as it can help provide answers to a variety of questions, such as: "Is the implementation compliant with the requirements?" or "Is the implementation even complete?" [2]. Many such questions can be answered depending on the completeness of the traceability links between the requirements and other software artifacts. However, in practice, a variety of traceability problems occur which generally include the use of informal traceability methods, failure in the cooperation between people responsible for coordinating traceability, difficulty in obtaining necessary information in order to support the traceability process, and lack of training of personnel in traceability practices.

To deal with such challenges and the additional burden of today's globally distributed development environment, some researchers have introduced an Event-Based method [1]. In this approach, the author proposes a methodology in which requirements and other software engineering artifacts can be linked through publish-subscribe relationships. This type of relationship is widely used in systems such as news service subscriptions and hardware management. In this Event-Based Traceability (EBT) system, requirements and other software artifacts that may induce changes are considered to be publishers while artifacts dependent on such changes act as subscribers. Hence the requirements are published and the performance models subscribe to the system.

A change in requirements will cause events to be published to an event server, which in turn will send out notifications to all dependent subscribers. The "publish-subscribe" model used within EBT allows automatic linkage and validation of values within the requirements that are established with the EBT system. The EBT system then acts as an automated change notification agent. When changes are made, the affected artifacts and models are logged, and the developer can determine which artifacts to update, and which to ignore. This is done based on factors such as the criticality of the requirement changed. The developers, can then make the necessary changes. The major advantage of this system is its support for managing changes as well as its ability to support the identification and maintenance of artifacts. Another research project presented the Information Retrieval (IR) approach as the key to recover traceability links between code and documentation [3]. This research introduces a methodology where Information Retrieval techniques are used to link artifacts to each other as well as requirements, through a mechanism that queries the set of artifacts. Next, by using an indexing process, artifact information and semantics are parsed and used in order to rank each artifact on its relevance to a given query. The rankings are then used to create links between artifacts, which are returned to the user. The user can use the rankings in order to understand the relationships between artifacts and even requirements in order validate the links that have been generated by the system.[8] Establishing these traceability links can help support tasks such as: program comprehension, maintenance, requirement tracing, impact analysis and reuse of existing software[5]. The premise of this research relies on the probability that most programmers tend to use meaningful names for program items, such as: function names and variables. Concepts implemented in the code are suggested by the use of such names. By using such correlations, queries can be constructed from modules in the source code. This proposed model concludes that IR provides a practical solution to the problem of semi-automatically recovering traceability links. In a different research work titled “Rule-Based Generation of Requirements Traceability Relations”, the authors address the challenges that arise when analyzing requirements [6]. This approach uses traceability rules to support the automatic generation of traceability relations. The generation of such relations is based on the *requirement-to-object-model* traceability rules. They help to trace the requirements and use case specification documents to an analysis object model, and *inter-requirements* traceability rules to trace requirement and use case specification documents to each other.[9] Throughout this approach, the authors focus on the challenge that requirements are expressed in natural language and often contain ambiguity.

Other new models have also been proposed to support the ideology of requirement traceability, one such model is the Conceptual Trace Model. It consists of three parts; A fine-grained trace model, which determines the types of documentation entities and relationships to be traced to support impact and implementation of system requirements changes; A set of process descriptions that describe how to establish traces and how to analyze the impacts of changes and the third part is tool support that provides (semi-) automatic impact analyses and consistency checking of implemented changes. [7] Our proposed model shares some of these concepts, but with a unique approach to requirement traceability.

III. PROPOSED TRACEBILITY MODEL

The proposed model is an extension to a previously suggested traceability model [4] which allows the software developer to achieve traceability at the source code level. The model focused on keeping track of the sets of working modules specific to satisfying the requirements. This model is the base for our extension and the new model thus offers a number of enhancements and features. There are two types of users for requirements traceability, high-end users and low-end users. Low-end users tend to consider traceability only because it is required, while high-end users recognize the importance of traceability [4]. This new model is simple for low-end users, yet comprehensive for high-end users.

It is composed of a Traceability Engine Component (TEC), a Traceability Viewer Component (TVC), and a Quality Assurance Interface (QAI). The first component, the TEC, is used to assist developers correlate source code elements with the software requirements. It functions by first reading in the requirements data from the requirements database, analyzes the source code and corresponding requirements, and creates its own internal traceability matrix. The TEC supplies this data to the QAI for evaluation, and is then updated with the results. The data that the TEC receives and the results of its own analysis are kept in a Traceability Database where it is accessible for re-evaluation at each stage of software development.

The TEC also contains an interface that enables the developer to indicate flags relating each piece of code or file to a specific requirement. When the code is checked into the CVS (Concurrent Version System), a version control system which is used to record source code history and documents, the TEC detects any change that has been made, and will prompt the developer to indicate the specific requirements related to each piece of code. Once all these relationships have been entered, the QAI is notified that there is data that needs to be verified. In addition, once the QAI has completed its process, the TEC will be able to determine which pieces of code do not have corresponding requirements and which requirements have no corresponding code.

The TVC acts as the 'client' portion of the proposed model. The TVC provides the software engineer with a unique way to view all the information that the TEC has gathered. It will have the ability to provide custom data such as: a detailed list of all requirements, reports regarding which requirements have been met and in which modules they are implemented, and the results of the verification and validation completed by the QAI.

The business analyst must insert the requirements into a spreadsheet, which is then imported into the database tables using a specialized tool. The interface added is called the Quality Assurance Interface (QAI), which a quality assurance specialist may use to verify that the code being checked meets the corresponding requirements. The importing of requirements and the QAI will be discussed in greater detail in the sections to follow.

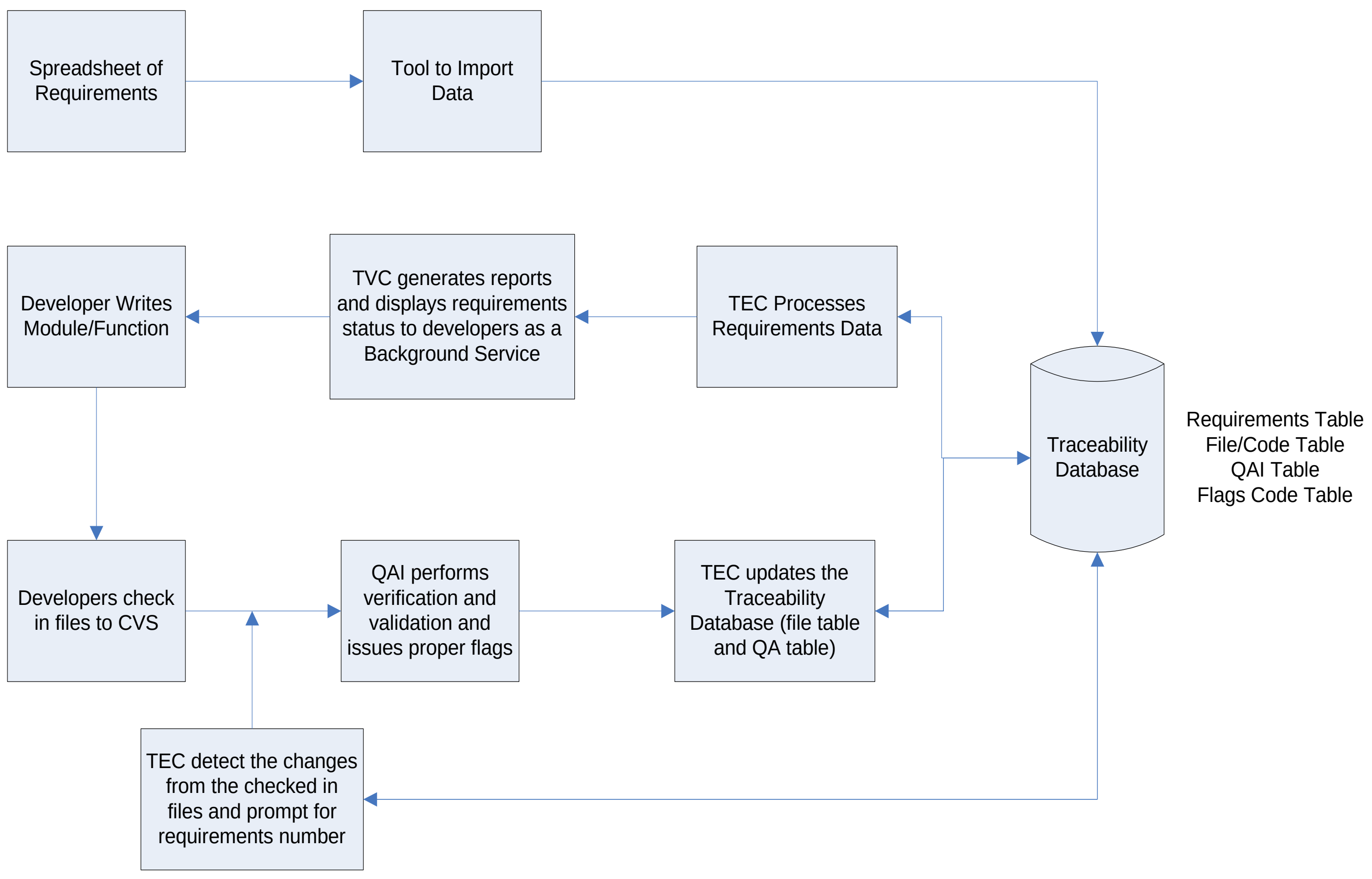
Spreadsheet of Requirements
Tool to Import Data
Developer Writes Module/Function
TVC generates reports and displays requirements status to developers as a Background Service
TEC Processes Requirements Data
Traceability Database
Requirements Table
File/Code Table
QAI Table
Flags Code Table
Developers check in files to CVS
QAI performs verification and validation and issues proper flags
TEC updates the Traceability Database (file table and QA table)
TEC detect the changes from the checked in files and prompt for requirements number

Figure 1: Proposed Traceability Model

TABLE 1: Requirements Flags

Flag Name	**Description**	**Comments**
NCF	Code Not Checked Flag	File name/line number and requirement set should be evaluated
CVF	Checked and Valid Flag	Signifies a good file name/line number and requirement set
CNF	Checked and Not Valid Flag	Developer of the code is working on his/her code to satisfy the requirement
CNFDATA	Checked and Not Valid Flag (DATA Error)	Developer of the code is working on his/her code to fix the data error and satisfy the requirement
CNFLOGIC	Checked and Not Valid Flag (Coding Bug)	Developer of the code is working on fixing a bug in his/her code to satisfy the requirement
CRF	Code with no Requirement Flag	There is a significant amount of code that is not assigned a requirement match
RCF	Requirement with no Code Flag	A certain requirement has not been met with any of the source code from the project
RCFF	Requirement Changed Flag	Indicates that the requirement has been modified
RRCF	Related Requirement Changed Flag	Indicates that a requirement related to this requirement has been changed

A. Quality Assurance Interface (QAI)

The proposed model provides a mechanism to address the issue of validating and verifying that the requirements are actually being met. This model allows the software engineer to choose either to add the tags directly to the source code, or, to choose which requirement is being met from a list of all requirements. The QAI will be able to perform aspects of verification and validation such as to “double-check” that a requirement has actually been met. The QAI performs a dual job, which is to insure that the requirements are accomplished, and to report requirements or parts of source code that do not match.

The table (See table 1) shows the requirements flags that are used in the QAI to indicate the status of the verification and validation of the requirements and the code. When the source code is checked into a version control system (CVS), a table will be populated with all the file names and line numbers which satisfy certain requirements. Each of these file name/line number and requirement matches are assigned a flag, NCF, which signifies that they have not been verified by the QAI. In addition to this, each requirement is assigned an NCF flag to indicate that it does not have any corresponding code that meets the requirement or that the validity of the correlation has not been verified. To insure that the requirements are truly met, the QAI will parse this table and go through a six step process to determine if the requirements have been met.

First, the QAI will look up the requirement from the database in the initial list of requirements that were given for the project. Second, it will read in the description of the requirement that has been linked with the requirements in the database. Third, the QAI will find the file and the line from the file name/line number combination from the match. Fourth, it will read and evaluate the source code. Fifth, it will determine if the match is a good match and if the requirement is actually met. Lastly, the sixth step is to create a flag for this file name/line number and requirement match that signifies that the match has been checked. Furthermore, it will indicate that either the requirement has been accomplished or it has not been sufficiently met. If it has not been met an email will be sent to the developer with the message that the source code should be re-done.

The second major task of the QAI is to handle the flags for the software requirements. This will help solve the problem of reporting which requirements or parts of the source code do not have matches. There are nine different flags that can be assigned; see Table 1. The first of these flags is the Not Checked flag (NCF). This signals to the QAI that the file name/line number and requirement set should be evaluated as outlined above. The second flag is the Checked and Valid flag (CVF); this signifies that a file name/line number and requirement set are valid. The third flag is the Checked and Not Valid flag (CNF); this means that the developer of the code is working on the code to satisfy the requirements. The fourth flag is the Checked and Not Valid Flag as a result of a Data Error (CNFDATA); this indicates that the developer needs to work on the code to fix the data error in order to fulfill the requirement. The fifth flag is the Checked and Not Valid Flag as a result of a coding bug (CNFLOGIC); this indicates that there is a bug in the code that needs to be resolved before the requirement can be met. The sixth flag is the Requirement with No Code flag (RCF); this signifies that a certain requirement has not been met with any of the source code from the project. This should signal that the requirement needs to be completed. The seventh flag is the Code with No Requirement flag (CRF); this indicates that there is a significant amount of code that is not assigned a related requirement has changed since it was imported into the database.

The QAI also has the ability to handle changes made to the requirements. When a requirement is changed or simply modified, the corresponding flag field in all records in the QAI table containing this requirement will be reset to the Requirement Changed Flag (RCHF). In addition, all requirements listed in the related requirements field will have their flag in the QAI reset to Related Requirement Changed Flag (RRCF). With the use of these two flags, when requirements change, only the code that could possibly relate to the requirements will need to be reviewed to ensure that it still satisfies the new requirement. This model enables changes to be made to the requirements without a need for all requirements and code to be rechecked. Only the changed requirement and requirements that relate to it need to have their corresponding code reviewed by the QAI.

B. Traceability Database Tables

The Traceability Database contains five tables. The Requirements Table (See table 2) is populated with the requirements that were imported from the business analyst’s spreadsheet. The table will contain the following fields: the requirements key, which is the primary key for the table, the person adding the requirement, and a description of the requirement.

The related requirement field contains any requirements that are directly related to the listed requirement; any changes made to the requirement or its related requirements can be indicated in the QAI table (See table 5).

The Code/File Change Table (See Table 3) contains the following fields: the code key, which is the primary key for the table, the file path, the file name, the method name, the class name, the date the change was entered, and the name of the coder.

When the coder checks the code into requirements without a need for all requirements and code to be rechecked. Only the changed requirement and CVS, changes that have been made are tracked by assigning a new code key. For example, if Door.cs and abc.config, the new record will look like table 2. After the new record has been added, the TEC will prompt for the requirements key from the requirements table (Table 2).

TABLE 2: Requirements Table

RequirementKey (Primary Key)	**AddedBy**	**Description**	**Related Requirements**
1000	BA1	Display Name in the header.	1010
1001	BA2	Display Date.	1010
1002	BA1	Be able to select style.	1011
1003	BA1	To store names.	1012, 1013
1004	BA1	To change names.	1012, 1013
1005	BA2	To delete names.	1012, 1013

TABLE 3: Code/File Change Table

CodeKey (Primary Key)	**FilePath**	**FileName**	**Method Name**	**Class Name**	**Date**	**Coded by**
1000	\Main\Project1\	Display.cs	GetName()	Display	10/1/06	Coder1
1001	\Main\Project1\	Render.cs	GetLineNumber()	Render	10/12/06	Coder2
1002	\Main\Project2\	Cashier.cs	NULL	Cashier	10/13/06	Gary
1003	\Main\Project1\	Interface.cs	NULL	Interface	10/12/06	Coder4
1004	\Main\Project3\	Window.cs	NULL	Window	10/13/06	Coder5
1005	\Main\Project5\	Door.cs	Main()	Door	10/18/06	Gary
1006	\Main\	abc.config	NULL	NULL	10/18/06	Gary

The Code Flags table (See Table 4) contains fields for the Flag Key, which is the primary key, the flag name, the description of the flag, and the purpose of the flag. When flags are assigned in the QAI table (See table 5), the Flag Key is used; the Flag Name is used for display purposes only. In addition, by using the key in place of the name, it will be more functional for purposes such as grouping records. The QAI can also add custom-defined code flags to the system.

The QAI table (Table 5) is a link between the Requirements records and the Code/File records. After a new record has been added to the Code/File Change table, the TEC will prompt for a requirement key to correspond to the entry. Once it has been entered, the TEC will create a new record in the QAI table to show which files or lines of code correspond to which requirements. he fields in this table are: the QAIKey, the primary key for the relationship; the requirements key, a foreign key to the Requirements table; the Code/File Key, a foreign key to the Code/File Change Table; the Flag Key, a foreign key to the Code Flags Table (discussed later); the date the record was entered into the database; and the person who performed the QA. Based on table 2, once the TEC has received the requirements key input, a table will be displayed as in table 4. The QA analyst then performs the verification and validation, and the flags will be assigned to each record through the QAI interface. After this has been completed, the TEC gathers the data and imports it into the QAI table as reflected in table 6.

TABLE 4: Code Flags Table

FlagKey (Primary Key)	**Flag**	**Description of Flag**	**Purpose**
1000	NCF	Code Not Checked Flag	File name/line number and requirement set should be evaluated
1001	CVF	Checked and Valid Flag	Signifies a good file name/line number and requirement set
1002	CNF	Checked and Not Valid Flag	Developer of the code is working on his/her code to satisfy the requirement
1003	CNFDATA	Checked and Not Valid Flag (DATA Error)	Developer of the code is working on his/her code to satisfy the requirement
1004	CNFLOGIC	Checked and Not Valid Flag (Coding Bug)	Developer of the code is working on his/her code to satisfy the requirement
1005	CRF	Code with no Requirement Flag	There is a significant amount of code that is not assigned a requirement match
1006	RCF	Requirement with no Code Flag	A certain requirement has not been met with any of the source code from the project
1007	RCHF	Requirement Changed Flag	Indicates that the requirement has been modified
1008	RRCF	Related Requirement Changed Flag	Indicates that a requirement related to this requirement has been changed

TABLE 5: QAI Table Default Entry

QAIKey (foreign key to Requirements Table)	**RequirementKey (foreign key to Requirements Table)**	**Code/File (foreign key to the Code File Table)**	**Flag (foreign key to Code Flags Table)**	**Date**	**QA by**
1001	1001	1001	1000 (NCF, Default)	10/1/06	QA1
1002	1001	1002	1000 (NCF)	10/12/06	QA1
1003	1001	1003	1000 (NCF)	10/13/06	Gary
1004	1001	1004	1000 (NCF)	10/12/06	QA1
1005	1002	1004	1000 (NCF)	10/13/06	QA2
1007	1003	1005	1000 (NCF)	NULL	NULL
1008	1004	1006	1000 (NCF)	NULL	NULL

TABLE 6: QAI Table after QAI is Performed

RequirementKey (foreign key to Requirements Table)	**Code/File (foreign key to the Code File Table)**	**Flag (foreign key to Code Flags Table)**	**Date**	**QA by**
1001	1001	1000 (NCF, Default)	10/1/06	QA1
1001	1002	1000 (NCF)	10/12/06	QA1
1001	1003	1000 (NCF)	10/13/06	Gary
1001	1004	1000 (NCF)	10/12/06	QA1
1002	1005	1000 (NCF)	10/13/06	QA2
1003	1005	1001 (CVF)	10/19/06	Dan
1004	1006	1003 (CNFDATA)	10/19/06	Dan
1005	NULL	1006 (RCF)	10/19/06	Dan

The TEC also has the ability to indicate which pieces of code do not have corresponding requirements and which requirements do not have corresponding code. The TEC takes the keys from the Code/File Change table and the Requirements table and verifies that each exists in a record in the QAI table. If they do not exist, then a record is created in the QAI table with a null value in the field of the item that does not exist, as in the last record in table 6.

C. Populating the Database

Another advantage of this model is that it considers the initial state of the database as shown in figure 1. But the question that follows is: how does the database get populated initially? Using a tool to fill the database with requirements can be one technique. Most current Database Management Systems such as SQL Server and Oracle, have a specialized tool to allow the importation of data from other types of file formats. One example is the Data Transformation Service (DTS) that is available with the more recent versions of SQL Server. This service allows the user to import data using a convenient user interface in the form of a wizard. Even though this wizard is convenient, it is not likely that the business analysts will have direct access to the database. Therefore they will not be able to use this wizard and must list the requirements in an organized manner. The business analysts will have to give the data to the developers in a format that can be imported into the database.

In general, business analysts are skilled in working with spreadsheets. Therefore, the business analysts should list the requirements in a spreadsheet with each row containing one requirement. This spreadsheet will be comprised of the following columns: requirement number, requirement name, description of the requirement, and related requirements. The requirement number must be unique and can serve as a key. This number will be chronological and the business analysts will be given a block of numbers from the developer that he or she will be able to use for that particular spreadsheet. The related requirements column will give the requirement number for any requirements that are related to it. This list of requirement numbers must be a set of valid values that are separated by commas. This would make it simple for the developer to parse these fields and extract the requirement numbers. The columns in this spreadsheet must be named as follows:

• RequirementKey,

• AddedBy

• RequirementName,

• Description,

• RelatedReqNumbers

The columns must have these names to maintain database consistency and to allow the tool to recognize which column in the spreadsheet corresponds to which column in the database. There are tools available for SQL Server, Oracle, MySQL, and Access to import data from Excel spreadsheets, therefore spreadsheets created in Microsoft Excel would be the most versatile.

IV. CONCLUSION AND FUTURE WORK

As previously noted, requirements traceability in the early stages plays a crucial role in the software development lifecycle. This model provides a very intuitive and dynamic way of requirements traceability. It provides a formal and measurable process to carry out traceability which can really be critical in exposing the defects at very early stages of the lifecycle. The suggested model amalgamates the features of the event based tracking and information retrieval tracking and adds new features in the design which makes it a very efficient method for requirement traceability. However, in order for this approach to be successful it does require commitment and support from the quality assurance group. The proposed model will prove beneficial to the software engineer and the software quality assurance process and will help in optimizing the process as a whole. The TEC, TVC, and QAI components provide a very efficient way of tracking and tracing requirements which can be quite tedious to detect considering the complex nature of requirements and its relationship. The Quality Assurance Interface can facilitate the verification of the code to the requirements by following a six-step process that we have prescribed. The process ensures all the changes in the requirements are accessed thoroughly and their impact are foreseen clearly. Furthermore, a mechanism to import requirements into a database was outlined with detailed account of all the objects that will be used such as the tables . Finally, a flagging procedure is designed using Requirements Flags to provide traceability between the requirements and the source code. The flagging procedure clearly demarcates which parts of the code is linked to the requirements and which are not. This preliminary model provides a simple interface that allows developers to seamlessly locate the correct requirements and link them to the correct source code elements, thus providing a very dynamic and intuitive method of requirement traceability during the software development process.

Several directions for future work are possible. First and foremost, a tool implementing this model and its corresponding database will be useful in determining the feasibility of the proposed system. Case studies need to be conducted to further evaluate the effectiveness of such an approach. Further add-ons to the TEC and TVC can be done to make it more flexible and generic. The database can be further developed to accommodate more flags and features that helps in more detailed description of mapping attributes. The QAI can be further developed to be more dynamic and effective. Finally, it will be important to incorporate the tracing of software design documentation into this traceability model. The ultimate goal will be to provide traceability over every software artifact of the software development lifecycle.

References

[1] Cleland-Huang, J., Chang, C.K., and Christensen, M. “Event-based traceability for managing evolutionary change”. IEEE Transactions on Software Engineering, Volume: 29, Issue: 9, Sept. 2003, Pages: 796 – 810.

[2] Requirements Tracing – An Overview. Retrieved on March 9, 2005, from http://www.sei.cmu.edu/str/descriptions reqtracing_body.html

[3] Antoniol, Giuliano, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo.“Recovering Traceability Links between Code and Documentation”. IEEE Transactions on Software Engineering, VOL. 28, NO. 10, OCTOBER 2002

[4] Ahmed Salem and Johnny Lee “Requirements Traceability Model for the Coding Phase”, Conference on Computer Science, Software Engineering, Information Technology, E-Business and Applications, 2004

[5] Ramesh, Balasubramaniam. “Factors Influencing Requirements Traceability Practice”. Association for Computing Machinery, Inc., 1998.

[6] Spanoudakis, George, Andrea Zisman, Elena Perez- Minana, and Paul Krause. “Rule-based generation of requirements traceability relations”. Software Engineering Group, Department of Computing, City University, Northampton Square, August 2003.

[7] Mirka Palo. "Requirements Traceability", Department of Computer Science, University of Helsinki, October 2003.

[8] Grant Zemont. “Towards Value-Based Requirements Traceability”, DePaul University, Chicago Illinois, March 2005 .

[9] George Spanoudakis, Andrea Zisman, Elena Pérez-Miñana and Paul Krause. **“**Rule-based generation of requirements traceability relations”, Journal of Systems and Software, July 2004.

[10] Gotel and A. Finkelstein. “An Analysis of the Requirements Traceability Problem,” Proceedings of the First International Conference on Requirements Engineering, Colorado Springs, April 1994