Traceability between Code and Design Documentation in Database Management System: A Case Study

Mohammed Akour, Ahmad Saifan, and Osama Ratha'an
Computer Information Systems Department, Yarmouk University, Jordan

Abstract: *Traceability builds many strong connections or links between requirements and design, so the main purpose of traceability is to maintain consistency between a high level conceptual view and a low level implementation view. The purpose of this paper is to have full consistency between all components over all phases in the oracle designer tool by allowing traceability to be carried out not only between the requirements and design but also between the code and design. In this paper, we propose a new methodology to support traceability and completeness checking between code and design of oracle database applications. The new algorithm consists of a set of interrelated steps to initialize the comparison environment. An example of a student information System is used to illustrate the work.*

Keywords: *Traceability, oracle designer, completeness checking, design, source code, database, pl/sql, testing.*

Received February 18, 2016; accepted June 26, 2016

1. Introduction

In general an effective, maintainable and flexible system allows all stakeholders to trace and follow up the software in all system development phases.

Traceability refers to the ability to link between different artefacts: code, user manuals, design, and documentation development. The Center of Excellence for Software and Systems Traceability (CoEST)[5] defines traceability as “the ability to interrelate any uniquely identifiable software engineering artefact to any other, maintain required links over time, and use the resulting network to answer questions of both the software product and its development process”. Traceability is an important issue in software engineering in that it helps the developer to understand the design better. Moreover, it allows the developer to control the quality and the maintenance process of software [20]. Based on our knowledge, one problem in software engineering is that most software developers do not stay in the same company for long. Another problem is that software engineering, including design and code, evolves over time. This evolution in software systems causes serious complications where the developers do not remember vital details. In this case the model to code trace becomes outdated or incomplete and becomes worse over time. Traceability has been used in different types of system to check the consistency and the differences between the software development phases. It has been used in object oriented systems [3], mobile agent systems [6], and Oracle database application [17]. Several approaches, techniques and tools have been proposed in the literature to support software traceability such as [2, 4, 10, 11, 18, 21]. Surveys of these approaches have been carried out and can be found in [13, 19].

Standardization is being considered for traceability in software development such as MIL-STD-489, IEEE/EIA 12207, and ISO/IEC 12207.

There have been several primary studies on traceability between design and source code base as mentioned in [19]. However, it seems that relatively little work that has been done to support traceability between design and code base in Oracle designer.

Thousands of types of applications have been built using Oracle designer such as online transaction processing systems, decision support systems, and multipurpose applications. The use of Oracle designer has several benefits such as [14]:

- Single point of truth for application meta data.
- Accurate analysis of system requirements.
- Powerful default database and application design transformers.
- Complete mobile infrastructure suitable for many mobile enterprise demands.

During the evolution of software, several changes occur in the source code; these can come about whilst fixing a bug or adding features. Thus the programmer updates the code level without changing the effected requirements or even the design of that system, which means that the code of the system does not reflect the design. In this case when the developer needs to transform the code from the design for some reason, all previous changes to the code will disappear, because changes in the code are not documented in the design (generated design packages, modules, procedures function or any customer service are popular in oracle designer). The problem here occurs because there is no traceability between the code level and the design level [8].

The purpose of this paper is to provide full consistency between all components over all phases in the Oracle designer tool by allowing traceability to be made not only between the requirements and design but also between the code and design. So we need a new traceability algorithm to check the consistency and link the differences (if any) between the code level and the design level. This paper uses Procedural Language/Structured Query Language (PL/SQL) Developer 10.1 to perform all tasks that relate to the main algorithm. PL/SQL Developer is an Integrated Development Environment that is specifically targeted at the development of stored program units for Oracle Databases. PL/SQL programming has become a significant part of the total development process. PL/SQL Developer focuses on ease of use, code quality and productivity, and key advantages during Oracle application development [1]. An example of Student Information Systems (SIS) has been used to illustrate and evaluate our approach. The systems contains 215 packages developed in the Oracle Designer tool with 595512 lines of code.

2. Related Work

Due to the importance of traceability, as we have already mentioned a great deal of work exists in the literature. In this paper we will mention some of them, focusing on the traceability between the design and code approaches. As mentioned above there have been several primary studies on traceability between design and source code base; however, it seems that relatively little work has been carried out to support traceability between design and code base in Oracle designer. Javed and Zdun [13] studied 11 different traceability approaches and tools between the architecture design and source code. They studied the benefits and liability, existing empirical existence, and the challenges of the approach, and classified the approaches based on different criteria such as nature of the approach and automation of the approach.

Several event-based traceability tools (between design and code) have been presented in the literature e.g., [4, 11]. Buchgeher and Weinreich [4] developed a semi-automatic tool called Language for Integrated Software Architecture (LISA) that captures the traceability between the architectural component model and the source code. The LISA tool is based on a semi-formal architectural specification model e.g., UML or Architecture Description Languages (ADL). Another tool, developed by Hammad *et al.* [11], is the SrcTracer that is used to check whether the design is consistent with the code especially when changes occur in the code. An approach that is similar to our approach has been presented in [12]. The Catia prototype [12] is used to support traceability for change impact analysis of object oriented software. The approach present the ability of integrating the high level with low level software models. They applied their approach to a case study of an embedded system.

Ubashi and Kamei [22] presented an approach to support traceability between the architecture design and code based on observer design pattern. In their approach they selected what are called architecture points that describe the design, e.g. message send; then, they selected the corresponding program points, e.g., method call in the code; after that, they established the traceability by mapping the architecture points with the program points based on observer design pattern.

Antoniol *et al.* [3] used information retrieval to identify the consistencies and differences between the design and code base. They used the reverse engineering process to extract the design from code.

Then, they built a tool to check the similarities and differences between the extracted design and the actual design. Their work is limited to checking the consistencies and differences in the classes only; it does not cover the methods and relationships.

Ghabi and Egyed [9] proposed an approach to determine the shortcomings, errors and uncertainty in traceability between the design and the code. They defined a set of rules to identify the relationship between the model elements (e.g., components, states, transitions) and code elements (line of code, method, class or package).

Alves-Foss *et al.* [2] produced an XML-based traceability tool called ArgUML, which is used to support traceability between Unified Modeling Language (UML) design specification and its corresponding code. In this tool, they used XML metadata interchange to specify the design and JavaML to represent the code.

Cysneiros and Zisman [6] proposed a similar approach to ours, in which they described a rule-based approach to support automatic generation of traceability to identify the missing elements in Prometheus models [15] and JACK code [24] specification for agent-oriented systems. The model is represented in Extensible Markup Language (XML) format. Moreover, they developed a translator component using ANother Tool for Language Recognition (ANTLR) parser generator [16] to transform the JACK code into an XML format. After that, they compared the design with the code in order to check the correctness and completeness of the models.

3. Requirement To Design Traceability in Oracle Designer Tool

Oracle Designer provides a multi-user repository based on Oracle SCM, and is closely integrated with Oracle Forms Developer, Oracle's declarative database application development tool. In this way, Designer allows organizations to design and rapidly deliver scalable, client/server systems that can adapt to changing business needs [14]. Oracle designer tool has a strong traceability technique that ensures a high level (up to 100%) of consistency between model system requirements and design or even database level and modules which act as an interface for end users. This return to have huge arrangement effective repository contains all connected components by linking all the software stored in it and the tools have many utilities to manage repository.

When a test case discovers an error in the source code the developer uses the traceability algorithm to correct the requirement(s) that caused the error and then correct the design. However, we can start the correction process from the design level. The following example shows the traceability between the system model requirements and the design level in Oracle designer tool.

In order to build the system model requirements we first construct the entities using the Entity Relationship Diagrammer. In this example, we create an entity and call it Courses. This entity has three attributes: id as primary key with numeric data type, code with char data type, and NO with numeric data type. We then transform the entity (system model requirements) into database design using database design transformer. Figure 1 shows this process.

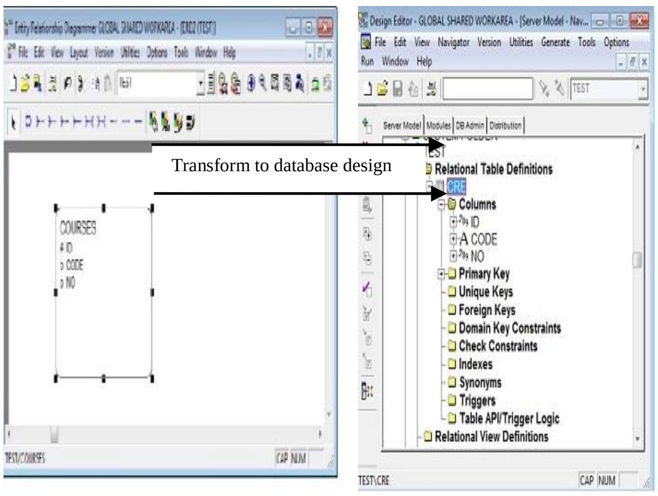


Figure 1. Transform system model requirement into database design.

After that, the developer generates the tables (database level). However, it is clear that the structure of this table has an error especially when the content of the NO attribute contains a value with a letter for example 281A. In this case the developer will change the data type of the NO attribute from numeric to char at the design level. Once this is done, the developer will automatically notice that some changes have occurred in the design and the design does no longer confirms the requirements. The traceability algorithm in Oracle designer tool will help in this case, as the changes made to the design will be shown in red or by a red circle filled beside the corresponding requirements model, as shown in Figure 2.

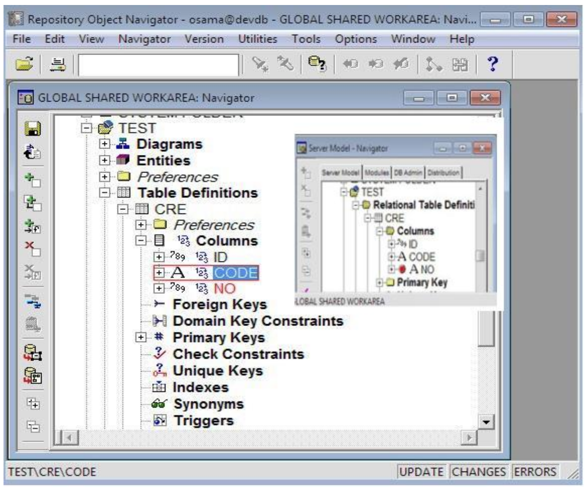


Figure 2. Requirement to design traceability result.

4. Code To Design Traceability Approach

Our code to design traceability approach consists of the following four steps:

1. Prepare the code at the design and database levels.
2. Manage distributed database systems by database link.
3. Compare the records of the two tables.
4. Provide suggestions.

4.1. Prepare the Code at the Design and Database Levels

In this paper we deal with the code at both the design level and the code level as text. Both levels have their own syntax which must be understood in order to convert both of them into the same format. The purpose of this conversion is to facilitate the comparison process.

In Oracle all the source codes of the database level (code level) will be saved in a table called USER_SOURCE. Table 1 shows part of the contents of USER_SOURCE Table. This Table consists of multiple recodes. Each record has several attributes such as:

- *Name*. Which refer to name of package, trigger or function.
- *Type*. Which contains procedure, package, trigger and function.
- *Line*. Which means sequence of line in source code.
- *Text*. Which represents one line of the source code.

Table 1. Part of the USER_SOURCE (code level) table.

Name	Type	Line	Text
Component name	Trigger	6	Declare
Component name	Trigger	7	--declare
Component name	Trigger	8	V_Ace_Code Number(2);
Component name	Trigger	9	V_Rce_Code Number(2);
Component name	Trigger	10	V_Program Number(1);
Component name	Trigger	11	V_Student_No Number(10);
Component name	Trigger	12	--local variable here

This Table exists in the schema owned by Oracle database. In our approach we call this Table CODE_LEVEL_DB.

At the design level the code structure is more complex than at the code level, because codes are saved as blocks in a table called RM_TEXT_LINES.

This table has multi attributes such as TXT_TEXT, TXT_SEQ … etc., The Table exists in the design level schema owned by Oracle designer as shown in Table 2:

Table 2. A block saved in RM_TEXT_LINES (design level).

TXT_TEXT	TXT_SEQ
L_first := false; l_prefix:= 'where'; else…	274
L_command := l_command||prefix||'sad.adm_rating=to_nur…	275
--p_transfer is not null if p_program is not null then	276
l_first := false; l_prefix := 'where' ; else…	277
L_command:= l_command ||prefix||'sad.check_prerequiste…	278

Therefore, there is a clear difference between the structure of the design level table and the code level table. The text on both sides has been saved in tables in which each record in CODE_LEVEL_DB table corresponds to one line in the source code. On the other hand, a record in the second table RM_TEXT_LINES corresponds to multiple lines (block of code) in the source code table.

To facilitate the process of comparison, the blocks in the RM_TEXT_LINES should be split into separate lines of code so that the two tables have the same format; we use the PL/SQL developer to do this, but first we need to create a procedure which we call *Split_block.* This procedure has one parameter, *packge_name,* that represents the name of the package that we are working on. Below is the algorithm of this procedure:

CREATE PROCEDURE Split_block (packge_name varchar2 (20)as
Begin
<Source code>;
End;

The body of the *split_block* procedure (source code) is used to construct a cursor in order to fetch all blocks in the package_name parameter which exists in rm_text_lines, ci_plsql_modules, sdd_folders, and sdd_folder_members.

The procedure splits each block into multi lines for each record in the RM_TEXT_LINES table by reading each block, line by line, and inserts each line in an intermediate table using r substr, instr and replace functions. Figure 3 shows the pseudo code for split_block procedure.

```
Read line from first character to end of line
         <line_name> := substr(<block index>, 1, instr(<block>, chr(13), 1, 2))
Remove    <line_name > contents from main block by
            REPLACE ((<block index>,<line_name>)
insert into <intermediate table> (Name, type, line, text)    values
            (package name, 'PACKAGE BODY', <line sequence>,<line_name>);
```

Figure 3. Split block procedure.

By applying this procedure we obtain an intermediate table (DESIGN_LEVEL_DB) for the design level that has the same structure as the CODE_LEVEL_DB table. Both tables have four attributes: name, type, line, and text.

4.2. Manage Distributed Database Systems by Database Link

The previous step constructed two tables, both with the same structure, in preparation for conducting the comparison process. However, it should be noted that both tables exist on two different databases schemas which means we have distributed database systems.

The central concept in distributed database systems is a database link. A database link is a connection between two physical database servers that allows a client to access them as one logical database. In other words, a database link is a pointer that defines a one-way communication path from one Oracle Database server to another. The link pointer is defined as an entry in a data dictionary table. To access the link, we must be connected to the local database that contains the data dictionary entry [23].

This requires work to build a DATABASE LINK between the two different instant (CODE_LEVEL_DB and DESIGN_LEVEL_ DB) to enable each schema to see the other using the following PL/SQL code:

Create DATABASE LINK <link_name> local
CONNECT TO <user_name>
IDENTIFIED BY <password>
USING <service_name>

When we run the previous statement under CODE_LEVEL instance we can see all objects (tables, views, sequences, package etc.,) that exist at the DESIGN_LEVEL; for example if we need to execute a selected statement from CODE_LEVEL to retrieve all records from student table that exists at the DESIGN_LEVEL then we should use database link after table name in select statement after "@" symbol :-
*select **
from student@db_link

4.3. Compare the Records of the Two Tables

The two different tables contain a text of code taken from the source code and design code. In this case, we can carry out the comparison process between the source code and the design code and vice versa as required. In our case we are going to check whether the source code (code level) matches with the design code or not by following steps:

4.3.1. Create Procedure to Perform the Comparison Process

This procedure has two parameters; the first parameter is package name which represents the name of the package that is used in the comparison process. The second parameter is service name which refers to the name of the service on which the comparison process is to be applied. Service Name is an optional parameter but package name is a mandatory parameter. If service name is not null, then the comparison is made at the service level, whereas if it is null the comparison is made for the package code. Then we construct a cursor to fetch the sequence to construct the rows of the two Tables: CODE_LEVEL_DB and DESIGN_LEVEL_DB as shown in Table 3.

Table 3. Constructs the records from CODE_LEVEL_DB and DESIGN_LEVEL_DB.

Row No	Code_ level	Rowno	Design_ level	result
1	Function cs_create_ Student_ no...	1	Function cs_create_student_no. ..	
2	(p_program_ Nature...	2	(p_program_ Nature...	
3	,p_program...	3	,p_program...	
4	,p_adm_id...	4	,p_adm_id...	
5	,p_dpt_id...	5	,p_dpt_id...	
6	,p_id...	6	,p_id...	
7	)	7	)	
8	Return number	8	Return number	
9	is	9	is	

The Table contains four columns where column 1 and column 2 represent the record number and one line of code respectively taken from the CODE_LEVEL_DB. Column 3 and column 4 represent the record number and one line of code respectively taken from DESIGN_LEVEL_DB.

4.3.2. Comparison Mechanism

The rtrim, ltrim, trim, subst and instr functions are used to remove any extra space from the left hand side and right hand sides. Now the two columns became ready for comparison. To complete the comparison process we should compare all the rows of columns 2 and 4. If the row content of the first column matches the content of the corresponding row in the second column then we insert the word "matched" in the corresponding result column. Otherwise, "unmatched" is inserted.

The comparison process does not always go smoothly. For example, when there is "unmatched" in the result column this result will affect the comparison process between the lines that follow it. For clarification, see the following example in Table 4.

Table 4. Code to design traceability algorithm results.

Row no	Code level	Row no	Design level	Result
1	Begin	1	Begin	matched
2	X:= 5;	2	Y:=6;	unmatched
3	Y:= 6;	3	Z:=100;	matched
4	Z:=100;	4	M:=18;	matched
5	C:=C+1;	5	W:=30;	unmatched
6	M:=18;	6	If Y>M then	matched

The comparison result of row 2 is (unmatched) but we note that in the code of the design level the statement Y:=6; appears in the code level in the next line (line 3). This indicates that line 2 of the code level (X:=5;) is a new line and it does not exists in the design level. But line 3 in the code level is an old line and exists in the design level in line 2. Therefore, we need to go back one step to compare between code level line 3 and design level line 2. This explains why the result is matched in line 3 even though the lines are not matched. The number of steps we go back will increase according to the number of unmatched results between the lines of codes. This appears in line 6 when we go back two steps because two unmatched results appear from the beginning of the comparison process. The pseudo code in Figure 4 represents how we solved the problem we mentioned previously in the comparison process.

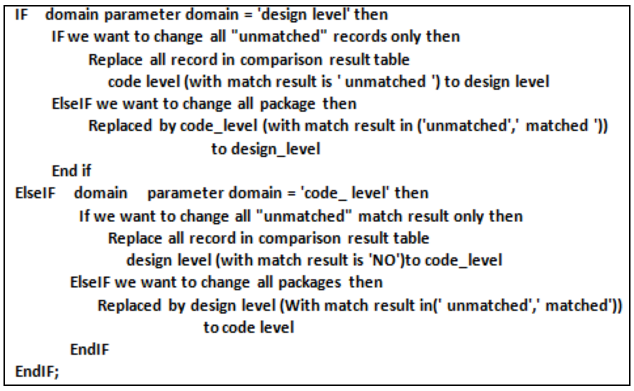
```
Initialize Backward_Step to zero
For each row exists in comparison_result loop
      Set ProdText to current prod text column
      Select the designer text column into Designer Text
      From comparison result table where
        Row number = current row number(from loop) - Backward Step;
   IF current ProdText = DesignerText then
        Update comparison result table
        Set result= "matched" the current row of the loop;
   Otherwise
      Backward_Step: = Backward_Step + 1;
   EndIF;
End loop;
```

Figure 4. Backward procedure for the comparison algorithm.

4.4. Provide Suggestions

After we complete the comparison process our approach provides the ability to automatically make some changes, whether in the code level or the design level. For example, it can automatically change all the records that have unmatched result.

Moreover, our approach has the ability to replace all codes of the design level based on the code level or vice versa. These two types of changes can be made by passing two parameters to the PL/SQL procedure. The first parameter is the domain parameter which indicates the domain we are going to change. This parameter has two values 'design level' and 'code level'. If it is 'design level' then we change the design so it matches the code level. If it is 'code level' then we change the code so it matches the design level. The second parameter indicates the type of change we are going to make. This parameter has two values ‘true’ or ‘false’. If it is ‘true’, then all records that have an unmatched result can be changed; otherwise the package can be replaced with another package. Figure 5 shows the suggested change procedure.



```
IF   domain parameter domain = 'design level' then
     IF we want to change all "unmatched" records only then
          Replace all record in comparison result table
             code level (with match result is ' unmatched ') to design level
     ElseIF we want to change all package then
          Replaced by code_level (with match result in ('unmatched',' matched '))
                           to design_level
     End if
ElseIF   domain   parameter domain = 'code_ level' then
         If we want to change all "unmatched" match result only then
             Replace all record in comparison result table
                design level (with match result is 'NO')to code_level
        ElseIF we want to change all packages then
            Replaced by design level (With match result in(' unmatched',' matched'))
                          to code level
        EndIF
EndIF;
```

Figure 5. Suggested changes procedure.

5. Evaluate The Results

After applying the suggested changes procedure and in order to check the correctness of our approach we used the diffchecker tool [7] which takes two texts as input and checks the similarity between them. We collected all the records as text for design and code level and input them to the diffchecker tool. We found that the two texts were 100% identical. So, the algorithm accomplished traceability between code and design documentation with extra suggestions.

This result is very high because the source code packages which are used in the code level were generated from design level and the changes in code level was very little. This result may change when this algorithm is applied to determine the similarity between design and code level if we have huge changes between the codes.

6. Threats to Validity

Threats to Validity our case study was with single large project written in Oracle, so the results might not generalize to other, projects, languages, or implementations. Moreover, the suggested changes algorithm is applicable on the text of the source code only. However, it should be also applied on the structure of the database for example column, domain, index, sequence, cluster definitions. Finally, the comparison process is not always that easy this because the source code written in a sequential manner and the comparison process must be performed between the corresponding Lines in both side's source code level and design level. The comparison process started by comparing a line with row number (n) in source code level to the corresponding line with row number (n - i) in design level, where (i) represent the backward step start with zero and i will be increased when the unmatched result is appeared. In other words, in the first iteration of the comparison process we compare the line with row number (1) in code level into line with row number (1) in design level. If the comparison result was unmatched then the next iteration in comparison is done between line with row number (2) in code level into line with row number (2-1) in design level which means current row number minus the backward step as shown in Figure 4.

7. Conclusions

We can avoid a lot of documentation problems in design level by applying traceability between code and design. In this paper we have developed an approach to support automatic traceability between the code and the design and vice versa for the Oracle designer tool. In this approach we prepared the design code and the database code so that they had the same format in order to carry out the comparison process. A Student Information System was used to evaluate and illustrate the work. The results of our evaluation suggest that our approach is very effective in the code to design traceability for Oracle database applications.

References

[1] Allround Automations, PL/SQL, 2012 Developer, http://www.allroundautomations.com, /plsqldev.html, Last Visited, 2016.

[2] Alves-Foss J., de Leon D., and Oman P., “Experiments in the Use of Xml to Enhance Traceability between Object-Oriented Design Specifications and Source Code,” *in Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, Big Island, pp. 3959-3966, 2002.

[3] Antoniol G., Caprile B., Potrich A., and Tonella P., “Design-Code Traceability for Object-Oriented Systems,” *Annals of Software Engineering*, vol. 9, no. 1-4, pp. 35-58, 2000.

[4] Buchgeher G. and Weinreich R., “Automatic Tracing of Decisions to Architecture and Implementation,” *in Proceedings of the 9th Working IEEE/IFIP Conference on Software*, Boulder, pp. 46-55, 2011.

[5] CoEST: Center of Excellence for Software Traceability, http://www.CoEST.org. Last Visited, 2015.

[6] Cysneiros G. and Zisman A., “Traceability and Completeness Checking for Agent-Oriented Systems,” *in Proceedings of the ACM Symposium on Applied Computing*, Fortaleza, pp. 71-77, 2008.

[7] Diff checker tool, https://www.diffchecker.com/, Last Visited, 2016.

[8] Espinoza A., Botterweck G., and Garbajosa J., “A Formal Approach to Reuse Successful

Traceability Practices in SPL Projects," *in Proceedings of the ACM Symposium on Applied Computing*, Sierre, pp. 2352-2359, 2010.

[9] Ghabi A. and Egyed A., "Exploiting Traceability Uncertainty Between Architectural Models and Code," *in Proceedings of Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, Helsinki, pp. 171-180, 2012.

[10] Haitzer T. and Zdun U., "DSL-Based Support For Semi-Automated Architectural Component Model Abstraction Throughout the Software Lifecycle," *in Proceedings of the 8th International ACM SIGSOFT Conference on Quality of Software Architectures*, Bertinoro, pp. 61-70, 2012.

[11] Hammad M., Collard M., and Maletic J., "Automatically Identifying Changes That Impact Code-To-Design Traceability During Evolution," *Software Quality Journal*, vol. 19, no. 1, pp. 35-64, 2011.

[12] Ibrahim S., Idris N., Munro M., and Deraman A., "Integrating Software Traceability for Change Impact Analysis," *The International Arab Journal of Information Technology*, vol. 2, no. 4, pp. 301-308, 2005.

[13] Javed M. and Zdun U., "A Systematic Literature Review of Traceability Approaches between Software Architecture And Source Code," *in Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, London, p. 16, 2014.

[14] Oracle Designer 6i Product Overview. http://www.oracle.com/technetwork/testcontent/otn-des6i-pover-wp-134818.pdf. Last Visited, 2015.

[15] Padgham L. and Winikoff M., *Developing Intelligent Agent Systems: A Practical Guide*, John Wiley and Sons, 2005.

[16] Parr T., *The Definitive ANTLR Reference: Building Domain Specific Languages*, Pragmatic Programmer, 2007.

[17] Requirements Management Ensuring Project Success.http://www.serena.com/docs/repository/products /rm/ds900-001-0704.pdf, Last Visited, 2016.

[18] Saifan A., Akour M., Alazzam I., and Hanandeh F., "Regression Test-Selection Technique Using Component Model Based Modification: Code to Test Traceability," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 4, pp. 90-95, 2016.

[19] Spanoudakis G. and Zisman A., *Handbook of Software Engineering and Knowledge Engineering*, World Sci Pub Co, 2005.

[20] Tiako P., *Designing Software-Intensive Systems: Methods and Principles: Methods and Principles*, IGI Global, 2008.

[21] Tran H., Zdun U., and Dustdar S., VbTrace: "Using View-Based and Model-Driven Development to Support Traceability in Process-Driven Soas," *Software and Systems Modeling*, vol. 10, no. 1, pp. 5-29, 2011.

[22] Ubayashi N. and Kamei Y., "Architectural Point Mapping For Design Traceability," *in Proceedings of the 11th Workshop on Foundations of Aspect-Oriented Languages*, Potsdam, pp. 39-44, 2012.

[23] Urman S. and McClain L., Oracle database 10g PL/SQL programming. McGraw-Hill, Inc.; Last Visited, 2004.

[24] Winikoff M., *Multi-Agent Programming*, Springer, Boston, MA, 2005.

Mohammed Akour is an Assistant Professor in the Department of Computer Information System at the Yarmouk University (YU). obtained his Ph.D degree in software engineering from NDSU with honor.

Ahmad Saifan is an assistant professor in the department of computer information systems at Yarmouk University (YU). He obtained his Ph.D degree in software engineering from Queen's University (Canada).

Osama Ratha'an is a master student in the department of computer information systems at Yarmouk University (YU).