International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys.14, No.1 (Jul-23)

http://dx.doi.org/10.12785/ijcds/140123

# Requirements Traceability:
## Recovering and Visualizing Traceability Links Between Requirements and Source Code of Object-oriented Software Systems

**Ra'Fat Al-Msie'deen**[1]

[1]*Department of Software Engineering, Faculty of IT, Mutah University, Mutah 61710, Karak, Jordan*

*Received 5 Oct. 2022, Revised 6 May. 2023, Accepted 8 May. 2023, Published 1 Jul. 2023*

**Abstract:** Requirements traceability is an important activity to reach an effective requirements management method in the requirements engineering. *Requirement-to-Code Traceability Links* (RtC-TLs) shape the relations between requirement and source code artifacts. RtC-TLs can assist engineers to know which parts of software code implement a specific requirement. In addition, these links can assist engineers to keep a correct mental model of software, and decreasing the risk of code quality degradation when requirements change with time mainly in large sized and complex software. However, manually recovering and preserving of these TLs puts an additional burden on engineers and is error-prone, tedious, and costly task. This paper introduces *YamenTrace*, an automatic approach and implementation to recover and visualize RtC-TLs in Object-Oriented software based on *Latent Semantic Indexing* (LSI) and *Formal Concept Analysis* (FCA). The originality of YamenTrace is that it exploits all code identifier names, comments, and relations in TLs recovery process. YamenTrace uses LSI to find textual similarity across software code and requirements. While FCA employs to cluster similar code and requirements together. Furthermore, YamenTrace gives a visualization of recovered TLs. To validate YamenTrace, it applied on three case studies. The findings of this evaluation prove the importance and performance of YamenTrace proposal as most of RtC-TLs were correctly recovered and visualized.

**Keywords:** Software engineering, Requirements traceability, Requirements engineering, Formal concept analysis, Latent semantic indexing, Object-oriented source code.

## 1. INTRODUCTION

*Requirements Engineering* (RE) aims at discovering, documenting, and maintaining a collection of requirements for the software system [1] [2]. RE involves five steps which are requirement discover, analysis, specification, validation, and management [3]. *Requirements Management* (RM) helps in maintaining requirement evolution during software development. RM is interested in all processes that lead to changing functional requirement of the software system [4]. *Requirements Traceability* (RT) is the key activity of RM process. RM process aims at finding and maintaining a traceability link of a particular requirement from its origins (or sources), across its specification and development to its, consequent deployment and use, and over a cycles of continuous improvement and repetition in any of these stages [5].

In RE, RT is an important task to attain a successful and effective RM process [6]. RtC-TLs shape the relations between requirement and source code artifacts. RtC-TLs can assist software engineers to know which segments of code implement a particular requirement. Recovering TLs between software *Requirements and Source code* (RaS) is very useful in numerous *Software Engineering* (SE) tasks such as software maintenance, reuse, and change [7] [8] [9]. Manually recovering and maintaining these TLs puts a further burden on engineers and is error-prone, tedious, costly mission, and some TLs may be missing. Traceability is a unique way to guarantee that the software code is consistent with its functional requirements. Traceability also ensures that software engineers have implemented all and only the required functional requirements [10]. This paper introduces *YamenTrace*, an automatic approach to recover and visualize RtC-TLs in *Object-Oriented* (OO) software system [11].

The manual creation of TLs among software RaS is time consuming, error-prone, tedious task, and complex activity in the SE domain [12]. Therefore, this study suggests mainly an automatic approach to recover TLs between RaS. TLs between RaS is an important process for software engineers to know which segments of code implement a certain requirement [13]. Thus, when requirement changes are suggested software developers know which parts of

*E-mail address: rafatalmsiedeen@mutah.edu.jo*

software code have to be modified [14]. Throughout software maintenance, a modification (or change) can not only influence source code but also cause an influence upon other artifacts such as requirements. Consequently, impact analysis [15] can use TLs to comprehend relations and dependencies between software RaS.

The traceability concept is defined as the degree of which every component in a software determines its reason for existing [16]. Furthermore, traceability can defined as the degree of which a link can be formed among two or more software artifact [17]. In this work, TLs is established between requirement and class documents of software system based on the *Textual Similarity* (TS) between those documents. In RE domain, RT term is defined as the capacity to illustrate and follow the life cycle of a requirement, in both a forwards and backwards orientation [18]. Therefore, the ability of software engineers to follow the traces to and from a requirement (*e.g.,* from its origins to its implementation) called RT [19]. *Backward traceability* means the capacity to follow a TL from a particular software artifact to its origins (or sources) of which is has been developed (*e.g.,* code $\longrightarrow$ requirement). While *forward traceability* [20] means the capacity to follow a TL from software artifact sources to its developed artifact (*e.g.,* requirement $\longrightarrow$ code). Figure 1 shows the Backward and forward directions of requirement traces. Software requirements define what the software should do. Functional requirements of software are statements of the services (or tasks) that the software must provide to its users [21] [22]. In this work, functional requirements are described by natural language [23]. Thus, for each requirement, there is a document describing single software service via short paragraph(s).

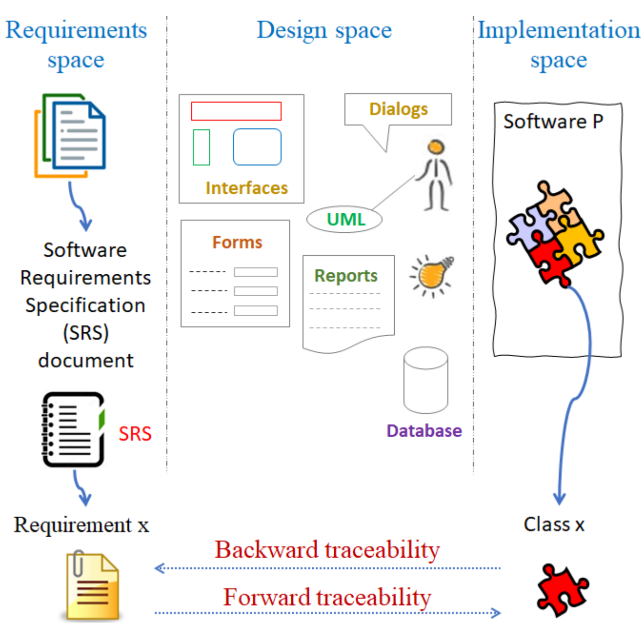


Figure 1. Backward and forward directions of requirement traces.

*YamenTrace* combines two techniques in order to recover TLs between RaS. The first technique is LSI and the second one is FCA [24] [25]. *Information Retrieval* (IR) techniques aim at identifying the documents that are relevant to a query in a group of documents [26]. LSI is an IR technique. It uses *Singular Value Decomposition* (SVD) on the *Term-document Matrix* (TDM). In the scope of this study, LSI can be described as an IR technique that uses a set of documents as the inputs and produces an indicator with document similarities (*i.e.,* TS) as the output [27]. FCA is a clustering technique. It allows to obtain an ordered collection of concepts from a *dataset* consisted of objects expressed by attributes [28]. The researcher who is concerned with FCA and LSI can find additional information in several studies [29] [30] [31].

FCA considers as an important clustering technique in SE filed [32]. FCA enables software engineers to extract an ordered set of concepts (*i.e.,* C ={C0, C1, C2, C3, C4, ...}) from a considerable dataset. This dataset is called a *Formal Context* (FC) which contained *Objects* (O) expressed by *Attributes* (A). An FC is a triple X = (O, A, BR) where BR is a *Binary Relation* between O set and A set (*i.e.,* $BR \subseteq OA$). An illustrative example of FC is presented in Table I. This FC describing *Jordan maps application* releases (*i.e.,* O) by their requirements (*i.e.,* A). A *Formal Concept* (FO) is a pair (X, Y) made of an object set $X \subseteq O$ and their common attribute set $Y \subseteq A$. For the given concept C1 = (X1, Y1), X1 is the extent of the concept C1, while Y1 is the intent of the concept C1. In this paper, author will use the *AOC-poset* for concepts visualization [33]. Figure 2 shows AOC-posets for FC of Table I.

TABLE I. FC describing Jordan maps application releases by their requirements.

| | Registration | Login | View map | View mosques | View restaurants | View museums | Change map view | Set favorite places |
|---|---|---|---|---|---|---|---|---|
| *Release_1* | | | x | | | | | |
| *Release_2* | | | x | | | | | x |
| *Release_3* | x | x | x | | | | | x |
| *Release_4* | x | x | x | | x | | x | x |
| *Release_5* | x | x | x | x | x | x | x | x |

As concepts (*i.e.,* FOs) of AOC-posets are well ordered, the intent of the top concept (*i.e.,* Concept_0) contains requirements that are shared with all Jordan maps releases. While the intents of all remaining FOs (*i.e.,* Concept_1 to Concept_4) contain sets of requirements shared to a subset of Jordan maps releases but not all releases. Moreover, the extent of each concept in AOC-posets is the set of Jordan maps releases that shared these requirements. For example, the extent of Concept_0 is ”Release_1”, and the intent of this concept is ”View map”.

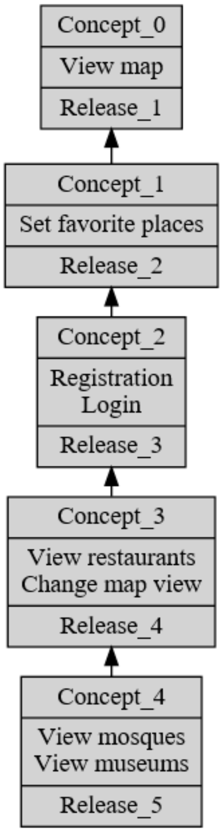


Figure 2. AOC-poset for FC of Table I.

In this paper, after measuring TS between software requirement and code documents using LSI, *YamenTrace* relies on FCA as a clustering technique to group similar documents together (*cf.* Figure 3). As another example regarding FCA technique, Table II shows a FC of a set of software requirements described by their implementation (*i.e.,* classes).

TABLE II. FC describing software requirements by their implementation (*i.e.,* classes).

| | class F | class G | class H | class I | class J | class K | class L | class M |
|---|---|---|---|---|---|---|---|---|
| *requirement A* | x | | | | | | | |
| *requirement B* | | x | | | | | | x |
| *requirement C* | | | x | x | | | | |
| *requirement D* | | | | x | | | x | |
| *requirement E* | | | | | x | x | | x |

Figure 3 displays AOC-posets for FC of Table II. Through this AOC-posets, we can notice that the Concept_0 contains the requirement A in its extent, and class F in its intent. Thus, the requirement A is textually similar to the class F, and is therefore grouped together into a disjoint concept. Also, we can notice from this AOC-posets that the requirement B is textually similar to class G, class I, and class M. Moreover, class H is the implementation of requirement C. Similarly, requirement D implemented by class L and class I.

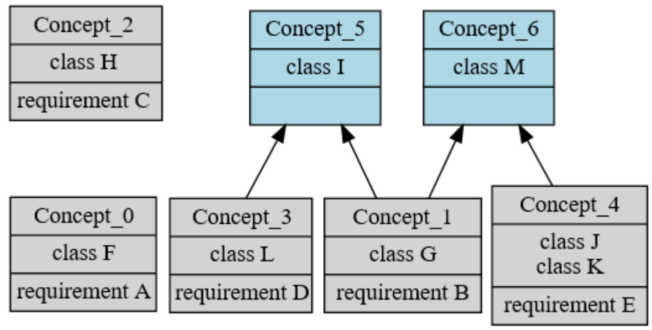


Figure 3. AOC-poset for FC of Table II.

LSI refers to a technique that calculates TS between different documents. TS is calculated using the occurrences of terms in documents of the corpus. If several documents share a significant number of terms, then those documents are deemed to be similar [34]. A complete example explaining how to calculate TS between a set of documents using LSI technique is presented in Section 3-C.

*YamenTrace* takes the software requirements and code as its inputs (*cf.* Figure 4). Then, *YamenTrace* recovers and visualizes the identified TLs between RaS. The first step of *YamenTrace* aims at extracting software source code (*cf.* Section 3-A). The second step generates all class documents of a given software code (*cf.* Section 3-B). Then, in the third step, *YamenTrace* relies on LSI method to define the similarity between requirement and class documents (*cf.* Section 3-C). Finally, in the fourth step, *YamenTrace* uses the similarity measure to identify TLs between RaS by using FCA (*cf.* Section 3-D).

When software engineers maintain and evolve software system, RT becomes outdated because engineers don’t care about updating traceability information. However, recovering RT later is a tedious and costly task for engineers. Thus, current studies have proposed several approaches to recover RT either semi-automatically or automatically (*cf.* Section 2). Among the suggested studies, the current approaches revealed that IR methods can automatically retrieve TLs between RaS. Though, IR methods lack accuracy. This paper suggests an automatic approach to recovering and visualizing TLs between RaS based on LSI and FCA techniques. The originality of *YamenTrace* is that it exploits all code identifier names (*e.g.,* method and attribute), comments, and relations (*e.g.,* inheritance) in TLs recovery process. *YamenTrace* uses LSI to find textual similarity across software code and requirements. While FCA employs to cluster similar code and requirements together. Furthermore, *YamenTrace* gives a visualization of recovered TLs.

The rest of this paper is organized as follows. Studies relevant to *YamenTrace* contributions are included in Section 2. *YamenTrace* is detailed in Section 3. Experiments are shown in Section 4. Finally, Section 5 wraps up this paper and makes suggestions for future work.

## 2. A LITERATURE REVIEW OF RT RECOVERY: A MINI SYSTEMATIC SURVEY

This section presents a systematic literature review related to YamenTrace contributions. The closest approaches to *YamenTrace* contributions are selected and presented in Table III. In this Table, the used code relations, element names, and IR techniques (*resp.* FCA) are highlighted.

Antoniol *et al.* [35] presented an IR technique to retrieve TLs between software source code and documentation. They applied the suggested approach to trace software code to functional requirements (*resp.* manual pages). The authors used IR technologies, such as *Probabilistic Model* (PM) and *Vector Space Model* (VSM).

Tsuchiya *et al.* [36] offered a semi-automatic approach to retrieve TLs between RaS in a collection of *Product Variants* (PVs). The authors exploited commonality and variability at code elements and requirements to reduce the search space, then recover the TLs. The authors recovered TLs using the *Configuration Management Log* (CML).

Gethers *et al.* [37] presented an automatic approach to create a links between RaS of a single software system. Their approach combines several IR techniques such as: *Jensen and Shannon* (JS) model, VSM and *Relational Topic Modeling* (RTM). Their approach shows that the integrated technique outperforms separate IR techniques. The authors reached an average precision of about 40%. On the other hand, they did not offer specific values for recall metric.

Marcus and Maletic [38] used LSI technique for recovering TLs between software RaS. By using the identifier names and comments appear in software source code, they manage to mine semantic information valuable for retrieving TLs.

Ali *et al.* [39] suggested an automatic IR approach in order to decrease the number of recovered false positive TLs by other IR studies. The suggested approach considers that information obtained from various code entities (*e.g.,* class and comments) are unique sources of information. Where, every source of information may serve as a specialist recommending TLs. The approach is used to decrease false positive TLs of VSM technique. The results reveal that the approach increases the accuracy of VSM, and it also decreases the efforts needed to manually eliminate false positive links. The current approaches for recovering TLs between RaS are summarized in Table III.

Table IV shows a comparison between current approaches related to YamenTrace contributions. The selected approaches are evaluated according to the following criteria: link creation, tool support, empirical evidence, evaluation metrics, and code language.

Several studies have recovered TLs between requirements and a variety of software artifacts, such as design documents, *Unified Modeling Language* (UML) diagrams. Dagenais *et al.* [40] have suggested a technique to extract TLs from API and learning sources based on code-like terms in documents. Their technique automatically analyzed the software documentation and linked code-like terms (*e.g., day()*) to explicit code elements (*e.g., DateTime.day()*) in the API. Kaiya *et al.* [41] have suggested a technique to discover change impacts on software code produced by requirements changes. They suggested a technique and a tool for impact analysis on source code produced by requirements changes. In their technique, an IR method is utilized to determine TLs between RaS.

Lin *et al.* [42] have introduced the *Poirot* tool. This tool supports traceability of diverse software artifacts. A PM is utilized as an IR technique to automatically create traces among different kinds of software artifacts, involving software code and requirements. The authors did not employ precision (*resp.* recall) metric to evaluate the quality of the produced TLs. Charrada *et al.* [43] have suggested an approach to distinguishing outdated requirements by using changes in software source code. Their approach first finds changes in code that are likely to influence software requirements. Then it obtains a collection of keywords depicting changes. These keywords are tracked to *Software Requirements Specification* (SRS) document to detect influenced requirements. The authors did not offer values for recall metric.

Yadla *et al.* [44] have presented an approach to tracing software requirements to bug reports by using LSI. Their approach is implemented in the *RETRO* tool. RETRO uses LSI technique to find TS among requirements and bug reports. Eaddy *et al.* [45] have proposed *CERBERUS*, a hybrid method for concern (or concept) location. The concern location problem is identifying code elements within a software that are relevant to the implementation of a feature or requirement. Their approach gave a value of 73% (*resp.* 75%) to the recall (*resp.* precision) metric. Khetade and Nayyar [46] have suggested a method based on LSI to find TLs between software code and free text documents. They used LSI technique to automatically identify TLs among software code and requirements.

Eyal-Salman *et al.* [47] have suggested an approach based on LSI to recover TLs among source code and features of a collection of PVs. While YamenTrace recovers RtC-TLs in a single software system. Chen *et al.* [48] [49] have suggested an automatic approach that exploits hierarchical tree and tree map visualization techniques to

TABLE III. Summary of RT approaches (comparison table).

| ID | Approach | Source code | | | | | | | | | | | IR techniques | | | | | | FCA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Code relations | | | Element names | | | | | | | | | | | | | | |
| | | Inheritance | Attribute access | Method invocation | Package | Class | Attribute | Method | Method parameter | Method local variable | Code Comment | Class file | LSI | VSM | PM | CML | JS | RTM | |
| 1 | Antoniol *et al.* [35] | | | | | x | x | x | x | | x | | | x | x | | | | |
| 2 | Tsuchiya *et al.* [36] | | | | | x | | x | x | x | | | | | | x | | | |
| 3 | Gethers *et al.* [37] | | | | | | | | | | | x | | x | | | x | x | |
| 4 | Marcus & Maletic [38] | | | | | x | x | x | | | x | | x | | | | | | |
| 5 | Ali *et al.* [39] | | | | | x | x | x | x | x | x | | | x | | | | | |
| $\rightsquigarrow$ | YamenTrace | x | x | x | x | x | x | x | x | x | x | | x | | | | | | x |

TABLE IV. An overview of RT approaches (comparison table).

| ID | Approach | Link creation | | Tool support | Empirical evidence | | | Evaluation metrics | | Code language | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Automatic | Semi-automatic | | Academic | Industrial | Open Source | Precision | Recall | *C++* | Java |
| 1 | Antoniol *et al.* [35] | x | | x | | x | | x | x | x | x |
| 2 | Tsuchiya *et al.* [36] | | x | x | | x | | x | x | x | x |
| 3 | Gethers *et al.* [37] | x | | x | x | | | x | | | x |
| 4 | Marcus and Maletic [38] | x | | | | x | | x | x | x | x |
| 5 | Ali *et al.* [39] | x | | x | | | x | x | x | x | |
| −− $\rightsquigarrow$ | YamenTrace | x | | | x | x | x | x | x | | x |

offer a universal structure of requirement traces and a comprehensive overview of each trace. While YamenTrace provides graph-based visualization of RT information. This graph visualizes traceability information among software RaS.

A study of the literature and comparisons of current approaches showed that there is no study or approach in the literature that uses the code relations (*resp.* FCA) in the process of recovering TLs between software RaS. In this paper, LSI is used to measure TS between requirement and class documents. The use of LSI in YamenTrace is not considered a novel aspect, as several studies have employed LSI to recover TLs between RaS. On the other hand, FCA technique is used to cluster similar requirement and class documents together based on TS measured by LSI. The use of FCA here is considered a novel aspect of YamenTrace approach where it has not been used before in RT studies, especially in the context of RaS. Also, YamenTrace prepares the class document in a novel way, where it exploits the identifier names, code relations and comments to construct the class document. Existing approaches used class files as it without any preprocessing. Finally, YamenTrace visualizes the recovered TLs between RaS.

## 3. YAMENTRACE APPROACH

A summary of *YamenTrace* is presented in Figure 4. The inputs of *YamenTrace* are software source code and requirements. While the outputs of *YamenTrace* are TLs across software RaS.

This study focuses on the recovering of TLs between RaS. Functional requirements describe the software services that must present to end users. This study considers that functional requirements are implemented via software source code. *YamenTrace* works only with OO software system [50]. Consequently, functional requirements are implemented using main OO code elements (*e.g.,* class and method). This study also assumes that code identifiers that implement a particular requirement are textually similar to requirement name and description.

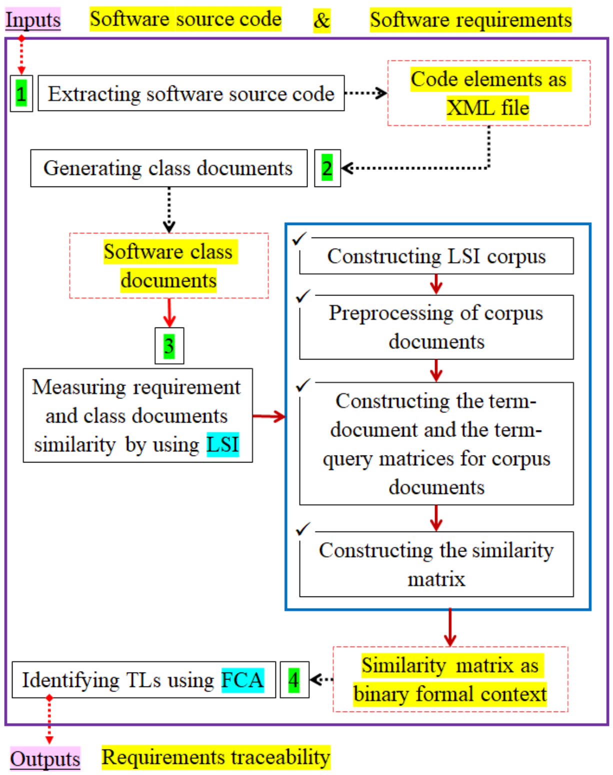


Figure 4. RT recovering process - *YamenTrace* approach.

Figure 5 displays a meta model representing the relation (or link) between software requirements and classes. In this work, requirement document is linked to one or more class documents based on the TS. Thus, for each requirement traceability, there is a requirement document and one or more class documents (*cf.* Figure 5). Each class document contains main source code elements that belong to this class. The software class may inherits attributes and methods from the superclass (*i.e.,* inheritance relation) [51]. Also, each software class belongs to a particular software package. Also, software class contains many attributes, methods, and several code comments. In addition, software method may contains parameter name(s) in its signature. Also, in its body, the method may contains a local variable, code comment, attribute access, or method invocation. All code elements, code dependencies and requirement traceability relations are given in Figure 5.

To illustrate some steps of *YamenTrace*, the author considers the *Drawing Shapes* (DS) software as an illustrative example in this study [52]. DS permits the user to draw several types of shapes such as line, and rectangle. DS software is considered as a small sized software system [53]. The author uses this example to better clarify the steps of *YamenTrace* approach. The author does not know the TLs between class and requirement documents in advance. *YamenTrace* only uses software RaS as an inputs for RT recovering process. Figure 6 displays the Graphical User Interface (GUI) of DS software.

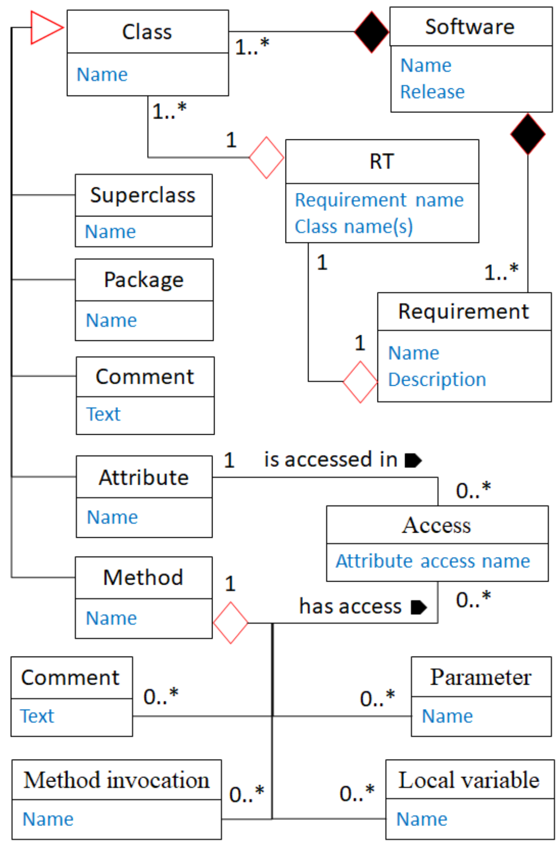


Figure 5. RT meta model of *YamenTrace* approach.

DS allows user to choose a color and a kind of shape to be drawn from software interface. The possible shapes involve line, rectangle, and oval [54] [55]. The software engineer can extend this version by adding other kind of shapes. Furthermore, DS lets the user to press a mouse button to generate a shape on drawing zone. Users of this software can resize the drawn shape by dragging the mouse anywhere on the drawing zone. DS allows the user to draw a picture by mixing multiple shapes together. Table V describes the functional requirements of DS software.

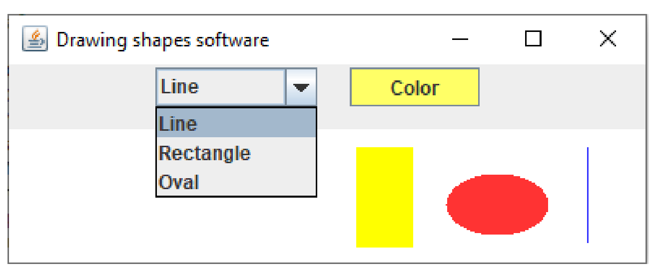


Figure 6. The GUI of DS application.

SRS document is the official document that contains functional (*resp.* non-functional) requirements of software system. This document contains the names of the requirements and a detailed description of each requirement. Natural language is usually used to describe a requirement of a program [56].

TABLE V. Requirements and their description of DS software.

| Requirement | Requirement description |
|---|---|
| *Draw a line* | The software shall allow user to draw lines, and choose the right color of the drawn lines. Also, it shall allow end user to draw a single line or unlimited lines on the drawing zone. To draw a line, software shall provide a method like *drawLine()* to draw a line between two points. |
| *Draw oval* | The software shall allow user to draw ovals, and choose the right color of the drawn ovals. Also, it shall allow end user to draw a single oval or unlimited ovals on the drawing zone. To draw an oval, the software shall provide a method like *drawOval()* to draw an oval. |
| *Draw rectangle* | The software shall allow user to draw rectangles, and choose the right color of the drawn rectangles. Also, it shall allow end user to draw a single rectangle or unlimited rectangles on the drawing zone. To draw a rectangle, the software shall provide a method like *drawRectangle()* to draw a rectangle. |

According to the suggested approach, *YamenTrace* recovers TLs between software RaS in *four* steps as described in the following.

### A. Extracting software source code

The initial step of *YamenTrace* is the extraction of software source code. Static code analysis [57] aims at identifying main OO elements (*e.g.,* class, method and comment). Static code analysis examines structural information (*e.g.,* data dependencies) of code [58]. For example, MyOval class extends MyShape class in DS software. This step takes software code as input and gives the code elements of software as output. *YamenTrace* relies on this code elements to construct the class documents of the whole software. The main code elements such as class and method names are important sources of information in order to identify TLs between RaS. Figure 7 shows the extracted code elements from DS software as XML file [59].

```
▼<Project ProjectName="Drawing shapes software">
  ▼<Packages>
    ▼<Package PackageName="Drawing">
        <Classes/>
      </Package>
    ▼<Package PackageName="Drawing.Shapes">
        <Classes/>
      </Package>
    ▼<Package PackageName="Drawing.Shapes.coreElements">
      ▼<Classes>
        ▼<Class ClassName="MyLine" Superclass="MyShape"---->
            <SuperInterfaces/>
            <Attributes/>
          ▼<Methods>
            ▼<Method MethodName="MyLine" MethodReturn ----->
                <LocalVariables/>
              ▼<AttributeAccesses>
                  <AttributeAccess AttributeAccessName="f.../>
```

Figure 7. The code elements file of DS software (partial).

### B. Generating class documents

In this step, *YamenTrace* relies on the code elements file that is extracted in the previous step (*cf.* Section 3-A). *YamenTrace* constructs the class documents for whole software based on code elements file. Each class document contains main code element names of this class, in addition to class and method comments (*resp.* . code relations). Figure 8 shows the code of *MyLine* class from DS software.

```java
// by Ra'fat AL-msie'deen  2016

package Drawing.Shapes.coreElements;

// Class that declares a line object.

import java.awt.*;

import Drawing.Shapes.coreFrame.MyShape;

public class MyLine extends MyShape {
    // constructor
    public MyLine(int firstX, int firstY, int secondX,
            int secondY, Color shapeColor) {
        super(firstX, firstY, secondX, secondY, shapeColor);
    }   // end constructor
        // draw a line

    public void draw(Graphics g) {
        g.setColor(getColor());
        g.drawLine(getX1(), getY1(), getX2(), getY2());
    } // end method draw
} // end class MyLine
```

Figure 8. The code of *MyLine* class from DS software.

Figure 9 gives an example of the class document extracted from DS application. This document contains the package name (*i.e., Drawing.Shapes.coreElements*). This document also contains the class name (*i.e., MyLine*). In addition, it includes attribute and method names (*e.g., draw*). Also, from the method signature, it involves parameter names (*i.e., g*). Regarding the method body, it contains the local variable names. Also, class document contains code relation names such as: inheritance (*i.e., MyShape*), attribute access (*e.g., X1*) and method invocation (*e.g., drawLine*). Finally, the class document involves class and method comments (*e.g., // draw a line*). Moreover, *YamenTrace* names the document with the name of the class (*i.e., MyLine*).

```
// by Ra'Fat AL-msie'deen 2016
Drawing.Shapes.coreElements
// Class that declares a line object.
MyLine
MyShape
// constructor
MyLine
firstX
firstY
secondX
secondY
shapeColor
// end constructor
// draw a line
firstX
firstY
secondX
secondY
shapeColor
draw
g
g
g
setColor
getColor
drawLine
getX1
getY1
getX2
getY2
// end method draw
// end class MyLine
```

Figure 9. An example of a class document (*i.e., MyLine*) from DS software.

### C. *Measuring requirement and class documents similarity by using LSI*

This paper considers that functional requirements are implemented by software classes. *YamenTrace* bases the detection of subsets of software classes, which each implements a functional requirement, on the measurement of TS between these classes and software requirements. This similarity measure is determined based on LSI technique. *YamenTrace* relies on the truth that classes engaged in implementing (or realizing) a functional requirement are textually nearer to one another than to the remainder of software classes. To calculate TS between software requirements and classes, *YamenTrace* applied LSI technique in *four* steps: 1) constructing LSI corpus, 2) preprocessing of corpus documents, 3) constructing TDM and Term-Query Matrix (TQM) and, finally, 4) constructing the *Cosine Similarity Matrix* (CSM). The similarity of requirement and class documents is constructed with LSI as detailed in the following.

#### 1) *Constructing LSI corpus*

LSI technique is a textual matching technique that aims to discover the TS between a query and a specified corpus of documents [60]. A corpus represents a group of documents. In *YamenTrace*, LSI corpus contains all software class documents. For query documents, each requirement document represents a query. The query document includes a description of a single software requirement, and it is named based on the name of that requirement. In *YamenTrace*, the document-corpus contains all class documents, while the query-corpus contains all requirement documents. Table VI offers the document and query corpus for DS software.

TABLE VI. Document and query corpus for DS software.

| Query-corpus (*i.e.,* requirement documents) | Document-corpus (*i.e.,* class documents) |
|---|---|
| *Draw a line* | *DrawingShapes* |
| *Draw oval* | *MyLine* |
| *Draw rectangle* | *MyOval* |
| | *MyRectangle* |
| | *MyShape* |
| | *PaintJPanel* |

#### 2) *Preprocessing of corpus documents*

When corpus is created, the textual data of each document must be preprocessed in order to recover TLs between RaS. At the beginning, *YamenTrace* removes stop words (*e.g.,* my, to, an, a, the, *etc.*), punctuation marks (*e.g.,* ?, !, *etc.*), special characters (*e.g.,* //, &, $, *etc.*), and numbers (*i.e.,* 0-9) from all corpus documents. Then, all document words are split into word tokens (*cf.* Table VII) by using the camel-case syntax (*e.g.,* fillRect → fill and Rect). Camel-case is a generally applied technique for code splitting (or dividing) algorithms in the SE field [61]. Finally, *YamenTrace* performs word stemming (*e.g.,* drawn → draw) on all document words (*cf.* Table VIII). In a SE domain, stemming (*e.g.,* removing word endings) is a text normalization method [62] [63]. In this step of YamenTrace, the word stemming is made by using *WordNet* [64]. All documents in document-corpus (*resp.* query-corpus) are preprocessed based on the previous procedure.

TABLE VII. Samples of the split word tokens from *MyLine* class document.

| Word | Tokens |
|---|---|
| *shapeColor* | shape and color |
| *getColor* | get and color |
| *Shapes.coreElements* | shapes, core, and elements |

This step aims at removing noise data from LSI corpus, saving memory space, and increasing the scale of *YamenTrace* to work with the large sized software system. Thus, preprocessing helps software engineers to find better textual matching between RaS and improves the achieved results.

TABLE VIII. Samples of the word stems (or roots) retrieved from *MyLine* class document.

| Word | Word stem (or root) |
|---|---|
| *Drawing* | draw |
| *Elements* | element |
| *Declares* | declare |

### 3) Constructing TDM and TQM for corpus documents

LSI technique starts with a TDM to count the occurrences of the `t` terms within a set of `d` documents. Thus, TDM is of the size `t` × `d` (*i.e.,* `TDM[t][d]`) where `t` (*resp.* `d`) is the number of unique-terms (*resp.* class documents) obtained from processed document-corpus (*cf.* Table IX). In this matrix, each unique term (*resp.* class document) is denoted by a row (*resp.* column), with each matrix cell (*i.e.,* `TDM[t][d]`) representing an indicator of the weight of the `t`*th* distinctive term in the `d`*th* class document. The weight is really specified based on the value of term occurrence of the `t`*th* term in the `d`*th* class document [57].

TABLE IX. TDM mined from document-corpus of DS software (partial).

| Term / Class | DrawingShapes | MyLine | MyOval | MyRectangle | MyShape | PaintJPanel |
|---|---|---|---|---|---|---|
| *line* | 0 | 6 | 0 | 0 | 0 | 1 |
| *draw* | 1 | 5 | 3 | 3 | 2 | 2 |
| *shape* | 21 | 4 | 3 | 3 | 6 | 29 |
| ... | ... | ... | ... | ... | ... | ... |

In the same way of TDM, TQM is aimed to count the iterations of the `t` terms within a set of `q` query documents. Table X shows part of TQM mined from query-corpus of DS software.

TABLE X. TQM mined from query-corpus of DS software (partial).

| Term / Requirement | Draw a line | Draw oval | Draw rectangle |
|---|---|---|---|
| *draw* | 7 | 7 | 7 |
| *line* | 7 | 0 | 0 |
| *shape* | 0 | 0 | 0 |
| ... | ... | ... | ... |

TQM is of the size `t` × `q` (*i.e.,* `TQM[t][q]`) where `t` (*resp.* `q`) is the number of unique-terms (*resp.* requirement documents) obtained from processed document-corpus (*resp.* query-corpus).

### 4) Constructing the similarity matrix

TS between query-corpus and document-corpus is represented by CSM. CSM columns and rows are represented as vectors of documents. The columns of CSM are documents of the document-corpus (*i.e.,* class documents), and the rows of CSM are documents of the query-corpus (*i.e.,* requirement documents). CSM cells take a value in a range between `-1` to `1` [65]. Table XI shows the extracted similarity matrix from requirement and class documents of DS software.

## D. Identifying TLs using FCA

In this step, *YamenTrace* uses FCA technique to recover, from requirement and class documents, which documents are textually similar. To convert the (numerical) CSM of the previous phase into (binary) FC, *YamenTrace* utilizes a commonly used threshold for cosine similarity which is `0.70`. Thus, the only pairs of requirement and class documents having a counted similarity larger than or equal to the selected threshold (*i.e.,* ≥`0.70`) are deemed textually similar. Table XII illustrates the FC achieved by transforming CSM from Table XI to binary FC.

TABLE XII. FC obtained from CSM in table XI.

| | DrawingShapes | MyLine | MyOval | MyRectangle | MyShape | PaintJPanel |
|---|---|---|---|---|---|---|
| *Draw a line* | 0 | 1 | 0 | 0 | 0 | 0 |
| *Draw oval* | 0 | 0 | 1 | 0 | 0 | 0 |
| *Draw rectangle* | 0 | 0 | 0 | 1 | 0 | 0 |

As an instance, in FC of Table XII, the requirement query document "*Draw a line*" is associated to the class document "*MyLine*" since their similarity value equals `0.99`, which is larger than the chosen threshold (*i.e.,* ≥`0.70`). On the other hand, requirement query document "*Draw a line*" and the class document "*Drawing-Shapes*" are not associated since their TS equals `0.03`, which is fewer than the selected threshold (*i.e.,* ≥`0.70`). Figure 10 shows the resulting AOC-poset from FC of Table XII.

AOC-poset in Figure 10 displays *four* concepts. Each concept of AOC-poset involves two elements: concept intent and extent [66]. Concept intent contains class documents, while concept extent contains requirement documents. For example, "*draw a line*" requirement is textually similar to "*MyLine*" class (*cf.* intent and extent of Concept_0). Also, some class documents do not show any TS with any requirement documents (*cf.* intent of Concept_3). Moreover, the

TABLE XI. Similarity matrix for requirement and class documents of DS software.

| | DrawingShapes | MyLine | MyOval | MyRectangle | MyShape | PaintJPanel |
|---|---|---|---|---|---|---|
| *Draw a line* | 0.037023712 | 0.989989848 | -0.131160318 | 0.010703977 | -0.03271227 | 0.012714135 |
| *Draw oval* | 0.026077607 | -0.070876593 | 0.88846873 | -0.452309648 | -0.011457777 | 0.014299836 |
| *Draw rectangle* | 0.033470816 | 0.014789518 | -0.395774033 | 0.9171953 | -0.020937944 | 0.018392209 |

intent of Concept_2 (*i.e., MyRectangle*) is textually similar to the extent of the same concept (*i.e., Draw rectangle*).

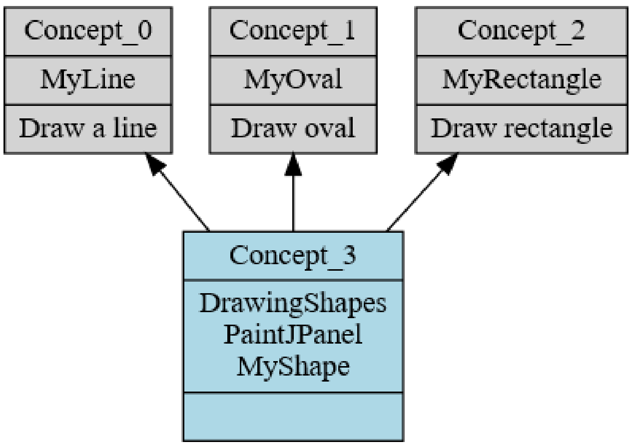


Figure 10. AOC-poset for FC of Table XII.

*YamenTrace* measures the quality and soundness of the recovered TLs using *precision* and *recall* measures. Precision and recall are two standard metrics widely employed in IR techniques [67]. Precision measure is the portion of recovered instances that are related (*cf.* Equation 1), while recall measure is the portion of related instances that are recovered (*cf.* Equation 2). The precision and recall metrics are computed as follows [68]:

$$Precision = \frac{RelatedLinks \cap RecoveredLinks}{RecoveredLinks} \quad (1)$$

$$Recall = \frac{RelatedLinks \cap RecoveredLinks}{RelatedLinks} \quad (2)$$

The most critical parameter to LSI technique is the number of selected *Term-topics* (T). This parameter is called the *Number of Topics* (NoT). In LSI, T is a set of terms that commonly co-occur in LSI corpus. *YamenTrace* needs a sufficient T to obtain real term associations. *YamenTrace* cannot employ a fixed NoT for LSI because it deals with several case studies of different sizes. For DS software case study, NoT is equal to 6.

Figure 11 shows the YamenTrace visualization of TLs between requirement and class documents for DS software. Visualizing TLs supports software engineers to recover and browse inter-relations among software documents in an intuitive manner [69].

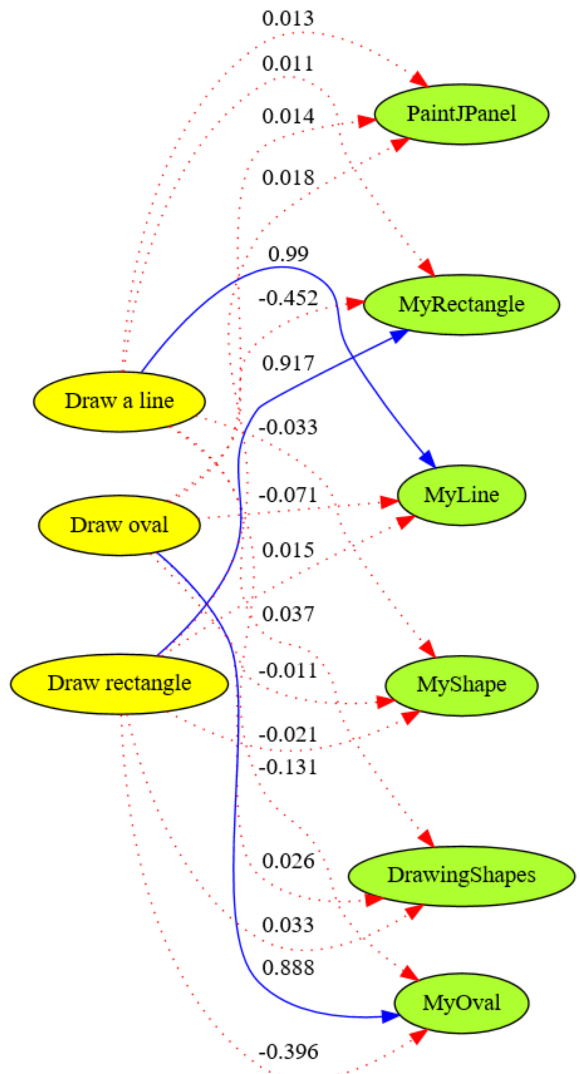


Figure 11. YamenTrace visualization of the recovered TLs from DS software.

The author manually identified the correctly recovered TLs between software artifacts (*i.e.,* requirement and class documents) based on his excellent knowledge about DS software. Thus, RT information of DS software is well known and documented. RT information helps software developers or engineers in order to check and validate *YamenTrace* findings. The success of *YamenTrace* is determined by the values of the metrics of precision and recall. Each measure has a value between 0 and 1. Figure 12 shows that recall measure is equal to 100% for all recovered TLs, which means that all related links are recovered. Also, Figure 12 shows that precision measure is equal to 100% for all recovered TLs, which means that all recovered links are related.

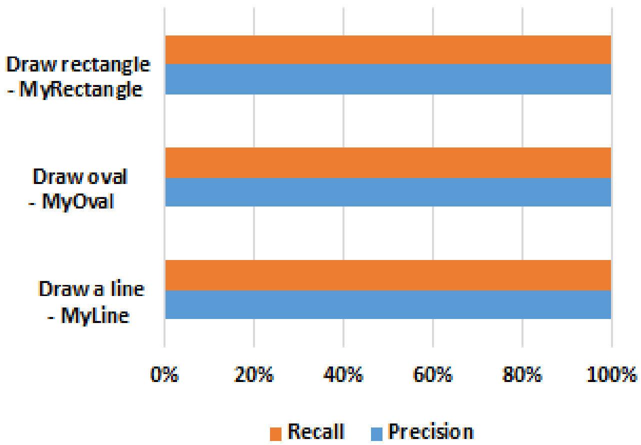


Figure 12. Evaluation metrics for the recovered TLs from DS software.

*YamenTrace* obtained excellent results based on the evaluation metrics calculated for each TL. One explanation for this excellent finding is that a shared vocabulary is utilized in requirement descriptions and their implementations; therefore, TS was an appropriate way to identify TLs between RaS.

## 4. EXPERIMENTATION

This section presents the selected case studies, experimental results, evaluation metrics, implementation information, and threats to the validity of *YamenTrace*.

Mobile Media (MM) is a Java open-source software system [70]. This software manipulates media (*e.g.,* photo) on mobile phones [71]. The reason behind choosing MM as a case study is that this study is well documented and known. Also, software requirements and source code of MM are available freely online [70]. Moreover, the implementation of each requirement is well-known. Thus, MM (*i.e.,* release 8) artifacts are accessible for comparison with YamenTrace results, and to validate the approach proposal.

Health Watcher (HW) is a public health system [72]. It is a Java open source software system [72]. HW is a real health complaint software system [73]. This software allows the citizen to register numerous types of health complaints (*e.g.,* complaints against food shops) [74]. Table XIII shows the standard software metrics for HW (*resp.* MM) software system. HW software considered as a large sized software system. HW software is well documented and known in SE filed. Requirements document of HW software is available for researchers [75]. In this case study, the implementation of each requirement is well known and documented.

DS, MM and HW software systems are presented in Table XIII. DS, MM and HW are described by the following metrics: Number of Software Packages (NOP), Number of Software Classes (NOC), Number of Software Attributes (NOA), Number of Software Methods (NOM), Number of Software Identifiers (NOI), Number of Software Comments (NOO), Number of Local Variables (NOL), Number of Method Invocations (NOI), and Number of Attribute Accesses (NOE). All software metrics are extracted by the YamenTrace parser [59]. The extracted XML code file for MM contains all needed code information for YamenTrace approach [76].

TABLE XIII. DS, MM, and HW software metrics.

| Case study / Metric | NOP | NOC | NOA | NOM | NOI | NOO | NOL | NOI | NOE |
|---|---|---|---|---|---|---|---|---|---|
| *DS* | 4 | 6 | 16 | 29 | 55 | 112 | 13 | 99 | 125 |
| *MM* | 17 | 51 | 166 | 271 | 505 | 904 | 258 | 1200 | 1790 |
| *HW* | 22 | 88 | 187 | 527 | 824 | 210 | 524 | 1952 | 3303 |

The number of requirements for MM (*resp.* HW) software is equal to 17 (*resp.* 9). Requirement names and descriptions are extracted from SRS document for each case study [77] [75]. Table XIV gives a samples of requirement names and their descriptions from MM and HW case studies. The complete set of requirements and their descriptions are available at YamenTrace webpage [59].

Let's imagine that a software engineer is expected to trace a *receive photo* requirement to its source code (*cf.* Figure 13). The code is developed in Java language, and software requirements are created (or written) in English language (*cf.* Table XIV). Let's assume that software engineer is using YamenTrace, to identify TLs between requirements and code of MM case study. The software engineer is tracing a *receive photo* requirement (*cf.* Req02 in Table XIV). Let's assume that the *SmsReceiverThread* and *SmsReceiverController* classes are the real implementation of *receive photo*. In this case, engineer identifies a link via YamenTrace between Req02 to SmsReceiverThread and SmsReceiverController because the approach will find matched terms between Req02 and these two class documents. So, it is essential to consider all source code elements, comments, and relations in YamenTrace approach.

TABLE XIV. Samples of requirement names and their descriptions from MM and HW case studies.

| **MM** | **Requirement** | **Requirement description** |
|---|---|---|
| Req01 | *Edit media label* | Edit media label feature allows a user to label (or name) a photo (or media) with specific text. Labels or captions could be utilized for future search functionality. |
| Req02 | *Receive photo* | Receive photo via SMS message allows mobile user to receive a photo (or media) from other users by short messaging service. This requirement shall allow the SMS receiver controller to accept or reject the photo or media. |
| Req03 | *Exception handling* | Exception handling is a non-functional requirement, and allows MM to handle exception related to media such as: image, photo album, and persistence exception. |
| **HW** | Requirement | Requirement description |
| Req01 | *Specify complaint* | This requirement lets a citizen to register complaints. Complaints can be: animal, food, or special complaint. |
| Req02 | *Register new employee* | This requirement lets new employees to be registered on HW software. |
| Req03 | *Update employee* | This requirement lets of the employee's data or information to be updated on HW software. |

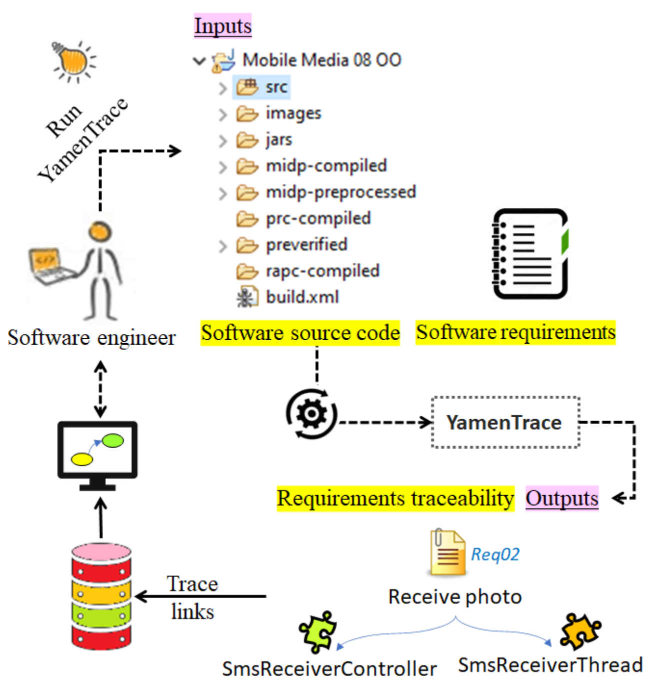


Figure 13. Running *YamenTrace* approach on MM software.

Table XV shows RtC-TLs results obtained from MM and HW software systems. Considering recall measure, its value is 100% for all recovered TLs. This implies that all class documents that implement software requirements are recovered correctly. Also, results appear that precision metric seems to be high for some requirements and low for others. This means that not all recovered class documents are relevant to software requirements.

Results show that some class documents are associated with more than one requirement documents. For instance, *addMediaToAlbum*, *captureVideoScreen*, and *videoCaptureController* class documents are linked to *capture photo* and *capture video* requirement documents. The reason behind this result is that capture photo and capture video requirements are implemented by same classes. Also, there are many textually matched terms between these documents.

TABLE XV. RtC-TLs results for MM and HW case studies.

| **MM** | **RtC-TL** | **Evaluation Metrics** | |
|---|---|---|---|
| | | **Precision** | **Recall** |
| R01 | *Create album* | 55% | 100% |
| R02 | *Delete album* | 80% | 100% |
| R03 | *Set favorite* | 30% | 100% |
| R04 | *View favorite* | 20% | 100% |
| R05 | *Sorting photo* | 70% | 100% |
| R06 | *Create media* | 60% | 100% |
| R07 | *Delete media* | 80% | 100% |
| R08 | *Edit media label* | 74% | 100% |
| R09 | *Copy media* | 50% | 100% |
| R10 | *Receive photo* | 80% | 100% |
| R11 | *Send photo* | 80% | 100% |
| R12 | *View photo* | 40% | 100% |
| R13 | *Capture photo* | 60% | 100% |
| R14 | *Play song* | 50% | 100% |
| R15 | *Play video* | 40% | 100% |
| R16 | *Capture video* | 70% | 100% |
| R17 | *Exception handling* | 60% | 100% |
| **HW** | **RtC-TL** | **Precision** | **Recall** |
| R01 | *Query information* | 50% | 100% |
| R02 | *Specify complaint* | 80% | 100% |
| R03 | *Login* | 60% | 100% |
| R04 | *Register tables* | 65% | 100% |
| R05 | *Update complaint* | 40% | 100% |
| R06 | *Register new employee* | 30% | 100% |
| R07 | *Update employee* | 40% | 100% |
| R08 | *Update health unit* | 50% | 100% |
| R09 | *Change logged employee* | 70% | 100% |

The results revealed that the use of all details from the class file, including identifier names, comments, and relations between code elements, caused a high value for the recall metric for all requirement documents. Thus, *YamenTrace* approach is capable of identifying the real implementation of software requirements. Furthermore, the results of the MM and HW software systems demonstrated that a single requirement can be implemented by one or more classes. On the other hand, one class can implement more than one requirement.

Moreover, results proved the ability of *YamenTrace* to identify the implementation of functional and non-functional requirements. For instance, YamenTrace recovered the real implementation of the *exception handling* requirement from MM case study. Based on the manual analysis of the obtained RtC-TLs results, usually there is textual similarity between a requirement and its implementation. Generally, engineers name the code elements through which the requirement is implemented with a vocabulary similar to the requirement description (or name). For each case study (*i.e.,* DS, MM, or HW), all experiment artifacts (*e.g.,* similarity matrix, AOC-poset, RtC-TLs visualization, *etc.*) are available on the YamenTrace webpage [59].

**Implementation:** In order to recover TLs between RaS of a software system, *YamenTrace* tool was developed and available on YamenTrace webpage [59]. In order to extract the main OO elements, author has developed a code parser that depends on the *Abstract Syntax Tree* (AST). AST is broadly employed in several areas of SE [78]. AST is utilized as representation of software code. *YamenTrace* parser uses the *JDOM* library to extract the code elements in the form of an XML file. In order to apply LSI, the author has developed his LSI tool, which is available on YamenTrace webpage [59]. For applying FCA, author used the *Eclipse eRCA* [79]. Also, in order to visualize AOC-poset and recovered RtC-TLs, YamenTrace uses the *Graphviz* library [80].

**Threats to validity:** *YamenTrace* works only with Java applications. This considers a threat to implementation validity that restricts YamenTrace capability to work only with the applications that are written by Java language. Another threat to the validity of YamenTrace is that developer might not employ the same vocabularies used in requirement description to name source code elements that implement this requirement. This implies that TS may be not trustworthy (or should be enhanced with other methods) in all cases to recover TLs between software RaS.

## 5. CONCLUSION AND PERSPECTIVES

This paper suggested an approach based on LSI and FCA to recover and visualize RtC-TLs in a single software system. *YamenTrace* has been implemented and evaluated on three case studies (*i.e.,* DS, MM, and HW). Findings displayed that most of RtC-TLs were recovered correctly. Figure 14 illustrates the key elements of YamenTrace approach.

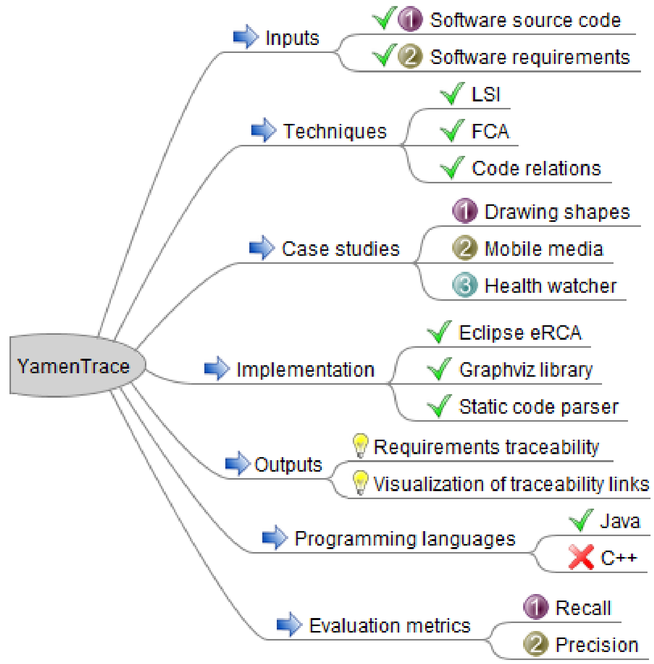


Figure 14. The key elements of *YamenTrace* approach.

The current approach works only with single software; therefor, one direction for future work of YamenTrace approach is to extend the current approach to work with a collection of PVs [81]. Then, it is important to extend the approach to identify the TLs between features and code of these PVs (*i.e.,* feature location) [82].

*YamenTrace* can be extended in many ways. For instance, YamenTrace approach is designed for product written in Java language, thus, future work could aim on extending the current implementation of *YamenTrace* to deal with other programming languages (*e.g.,* C++). Also, a further evaluation of *YamenTrace* can be done with other case studies. To do this, it would be necessary to find suitable case studies whose requirements and source code are freely available to carry out the whole approach described in this paper.

*YamenTrace* also plans to exploit useful information available in SRS document (*e.g., requirements dependency*) in TLs recovery process. Requirements dependency is an important aspect in tracing software requirements [83]. Furthermore, there is an urgent need to convert *YamenTrace* to a generic approach, in order to be able to find TLs between any kind of software artifacts (*e.g.,* design documents, or features) and source code.

## References

[1] I. Sommerville and P. Sawyer, *Requirements Engineering: A Good Practice Guide*, ser. The Kluwer International Series on Information Retrieval. Wiley, 1997.

[2] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A.

Ajagbe, E. Chioasca, and R. T. Batista-Navarro, “Natural language processing for requirements engineering: A systematic mapping study,” *ACM Comput. Surv.*, vol. 54, no. 3, pp. 55:1–55:41, 2022. [Online]. Available: https://doi.org/10.1145/3444689

[3] I. Sommerville, *Software Engineering*. Pearson, 2016.

[4] R. Al-Msie'deen, “A requirement model of local news web/wap application for rural communities,” Master's thesis, Universiti Utara Malaysia, Utara, Malaysian, 2008.

[5] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja, and K. Kamran, “Requirements traceability: A systematic review and industry case study,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 3, pp. 1–49, 2012.

[6] R. White and J. Krinke, “Tctracer: Establishing test-to-code traceability links using dynamic and static techniques,” *Empir. Softw. Eng.*, vol. 27, no. 3, p. 67, 2022. [Online]. Available: https://doi.org/10.1007/s10664-021-10079-1

[7] D. Hanspeter, A. Janes, A. Sillitti, and G. Succi, “Improving the identification of traceability links between source code and requirements,” in *Proceedings of the 18th International Conference on Distributed Multimedia Systems, DMS 2012, August 9-11, 2012, Eden Roc Renaissance, Miami Beach, FL, USA*. Knowledge Systems Institute, 2012, pp. 95–100.

[8] M. Zhang, C. Tao, H. Guo, and Z. Huang, “Recovering semantic traceability between requirements and source code using feature representation techniques,” in *21st IEEE International Conference on Software Quality, Reliability and Security, QRS 2021, Hainan, China, December 6-10, 2021*. IEEE, 2021, pp. 873–882.

[9] L. Linsbauer, S. Fischer, G. K. Michelon, W. K. G. Assunção, P. Grünbacher, R. E. Lopez-Herrejon, and A. Egyed, “Systematic software reuse with automated extraction and composition for clone-and-own,” in *Handbook of Re-Engineering Software Intensive Systems into Software Product Lines*, R. E. Lopez-Herrejon, J. Martinez, W. K. G. Assunção, T. Ziadi, M. Acher, and S. R. Vergilio, Eds. Springer International Publishing, 2023, pp. 379–404. [Online]. Available: https://doi.org/10.1007/978-3-031-11686-5_15

[10] N. Ali, Y. Guéhéneuc, and G. Antoniol, “Trustrace: Mining software repositories to improve the accuracy of requirement traceability links,” *IEEE Trans. Software Eng.*, vol. 39, no. 5, pp. 725–741, 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.71

[11] R. Al-Msie'deen, “Visualizing object-oriented software for understanding and documentation,” *International Journal of Computer Science and Information Security (IJCSIS)*, vol. 13, no. 5, pp. 18–27, 2015.

[12] G. Spanoudakis and A. Zisman, *Software traceability: A roadmap*. World Scientific Publishing, 2004, pp. 395–428.

[13] M. Rahimi and J. Cleland-Huang, “Evolving software trace links between requirements and source code,” *Empir. Softw. Eng.*, vol. 23, no. 4, pp. 2198–2231, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9561-x

[14] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, “A case study on the impact of refactoring on quality and productivity in an agile team,” in *Balancing Agility and Formalism in Software Engineering, Second IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2007, Poznan, Poland, October 10-12, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, B. Meyer, J. R. Nawrocki, and B. Walter, Eds., vol. 5082. Springer, 2007, pp. 252–266.

[15] T. W. W. Aung, H. Huo, and Y. Sui, “A literature review of automatic traceability links recovery for software change impact analysis,” in *ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020*. ACM, 2020, pp. 14–24.

[16] O. Gotel, J. Cleland-Huang, J. H. Hayes, A. Zisman, A. Egyed, P. Grünbacher, A. Dekhtyar, G. Antoniol, J. I. Maletic, and P. Mäder, “Traceability fundamentals,” in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. Springer, 2012, pp. 3–22.

[17] IEEE, “IEEE standard glossary of software engineering terminology,” *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[18] O. C. Z. Gotel and A. Finkelstein, “An analysis of the requirements traceability problem,” in *Proceedings of the First IEEE International Conference on Requirements Engineering, ICRE '94, Colorado Springs, Colorado, USA, April 18-21, 1994*. IEEE Computer Society, 1994, pp. 94–101.

[19] S. Winkler and J. von Pilgrim, “A survey of traceability in requirements engineering and model-driven development,” *Softw. Syst. Model.*, vol. 9, no. 4, pp. 529–565, 2010.

[20] IEEE, “IEEE guide for software requirements specifications,” *IEEE Std 830-1984*, pp. 1–26, 1984.

[21] R. Al-Msie'deen, A. H. Blasi, and M. A. Alsuwaiket, “Constructing a software requirements specification and design for electronic it news magazine system,” *International Journal of Advanced and Applied Sciences*, vol. 8, no. 11, pp. 104–118, 2021.

[22] R. Al-Msie'deen, *A Requirement Model of Local News Application for Rural Communities: A New Model for Rural News*. LAP LAMBERT Academic Publishing, 2014.

[23] A. M. Alfrijat and R. Al-Msie'deen, “A requirement model of local news WAP/WEB application for rural community,” *Advances in Computer Science and Engineering*, vol. 4, no. 1, pp. 37 –53, 2010.

[24] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, “Mining features from the object-oriented source code of a collection of software variants using formal concept analysis and latent semantic indexing,” in *The 25th International Conference on Software Engineering and Knowledge Engineering*. Knowledge Systems Institute Graduate School, 2013, pp. 244–249.

[25] R. A. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, “Feature mining from a collection of software product variants,” in *Actes des Cinquièmes journées nationales du Groupement De Recherche CNRS du Génie de la Programmation et du Logiciel*, ser. GDR GPL, L. Duchien, Ed., 2013, pp. 185–186.

[26] J. Lin, Y. Liu, and J. Cleland-Huang, “Information retrieval versus deep learning approaches for generating traceability links in bilingual projects,” *Empir. Softw. Eng.*, vol. 27, no. 1, p. 5, 2022.

[27] A. Yürekli, C. Kaleli, and A. Bilge, “Alleviating the cold-start playlist continuation in music recommendation using latent semantic indexing,” *Int. J. Multim. Inf. Retr.*, vol. 10, no. 3, pp. 185–198, 2021.

[28] J. Carbonnel, K. Bertet, M. Huchard, and C. Nebut, "FCA for software product line representation: Mixing configuration and feature relationships in a unique canonical representation," *Discret. Appl. Math.*, vol. 273, pp. 43–64, 2020.

[29] R. Al-Msie'deen, A. H. Blasi, H. E. Salman, S. S. Alja'afreh, A. Abadleh, M. A. Alsuwaiket, A. Hammouri, A. J. Al_Nawaiseh, W. Tarawneh, and S. A. Al-Showarah, "Detecting commonality and variability in use-case diagram variants," *Journal of Theoretical and Applied Information Technology*, vol. 100, no. 4, pp. 1113–1126, 2022.

[30] R. A. Al-Msie'deen and A. H. Blasi, "Software evolution understanding: Automatic extraction of software identifiers map for object-oriented software systems," *Journal of Communications Software and Systems*, vol. 17, no. 1, pp. 20–28, 2021.

[31] R. Al-Msie'deen, M. Huchard, A. Seriai, C. Urtado, and S. Vauttier, "Automatic documentation of [mined] feature implementations from source code elements and use-case diagrams with the REVPLINE approach," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 24, no. 10, pp. 1413–1438, 2014.

[32] M. R. Hacene, M. Huchard, A. Napoli, and P. Valtchev, "Using formal concept analysis for discovering knowledge patterns," in *Proceedings of the 7th International Conference on Concept Lattices and Their Applications, Sevilla, Spain, October 19-21, 2010*, ser. CEUR Workshop Proceedings, M. Kryszkiewicz and S. A. Obiedkov, Eds., vol. 672. CEUR-WS.org, 2010, pp. 223–234.

[33] R. Al-Msie'deen, M. Huchard, A. Seriai, C. Urtado, and S. Vauttier, "Reverse engineering feature models from software configurations using formal concept analysis," in *Proceedings of the Eleventh International Conference on Concept Lattices and Their Applications, Košice, Slovakia, October 7-10, 2014*, ser. CEUR Workshop Proceedings, K. Bertet and S. Rudolph, Eds., vol. 1252. CEUR-WS.org, 2014, pp. 95–106.

[34] F. Can, "Information retrieval data structures & algorithms, by william b. frakes and ricardo baeza-yates (book review)," *SIGIR Forum*, vol. 27, no. 3, pp. 24–25, 1993. [Online]. Available: https://doi.org/10.1145/182119.1096164

[35] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Trans. Software Eng.*, vol. 28, no. 10, pp. 970–983, 2002.

[36] R. Tsuchiya, H. Washizaki, Y. Fukazawa, T. Kato, M. Kawakami, and K. Yoshimura, "Recovering traceability links between requirements and source code using the configuration management log," *IEICE Trans. Inf. Syst.*, vol. 98-D, no. 4, pp. 852–862, 2015.

[37] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*. IEEE Computer Society, 2011, pp. 133–142.

[38] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, L. A. Clarke, L. Dillon, and W. F. Tichy, Eds. IEEE Computer Society, 2003, pp. 125–137.

[39] N. Ali, Y. Guéhéneuc, and G. Antoniol, "Requirements traceability for object oriented systems by partitioning source code," in *18th Working Conference on Reverse Engineering, WCRE 2011, Limerick, Ireland, October 17-20, 2011*, M. Pinzger, D. Poshyvanyk, and J. Buckley, Eds. IEEE Computer Society, 2011, pp. 45–54.

[40] B. Dagenais and M. P. Robillard, "Recovering traceability links between an API and its learning resources," in *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, M. Glinz, G. C. Murphy, and M. Pezzè, Eds. IEEE Computer Society, 2012, pp. 47–57.

[41] H. Kaiya, A. Osada, K. Hara, and K. Kaijiri, "Design, implementation and evaluation of a system for finding change impacts on source codes caused by requirements changes," *IEICE TRANSACTIONS on Information and Systems (Japanese Edition)*, vol. J93-D, no. 10, pp. 1822–1835, 2010.

[42] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan, and X. Zou, "Poirot: A distributed tool supporting enterprise-wide automated traceability," in *14th IEEE International Conference on Requirements Engineering (RE 2006), 11-15 September 2006, Minneapolis/St.Paul, Minnesota, USA*. IEEE Computer Society, 2006, pp. 356–357.

[43] E. B. Charrada, A. Koziolek, and M. Glinz, "Identifying outdated requirements based on source code changes," in *2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, September 24-28, 2012*, M. P. E. Heimdahl and P. Sawyer, Eds. IEEE Computer Society, 2012, pp. 61–70.

[44] S. Yadla, J. H. Hayes, and A. Dekhtyar, "Tracing requirements to defect reports: an application of information retrieval techniques," *Innov. Syst. Softw. Eng.*, vol. 1, no. 2, pp. 116–124, 2005. [Online]. Available: https://doi.org/10.1007/s11334-005-0011-3

[45] M. Eaddy, A. V. Aho, G. Antoniol, and Y. Guéhéneuc, "CERBERUS: tracing requirements to source code using information retrieval, dynamic analysis, and program analysis," in *The 16th IEEE International Conference on Program Comprehension, ICPC 2008, Amsterdam, The Netherlands, June 10-13, 2008*, R. L. Krikhaar, R. Lämmel, and C. Verhoef, Eds. IEEE Computer Society, 2008, pp. 53–62.

[46] P. N. Khetade and V. V.Nayyar, "Establishing a traceability links between the source code and requirement analysis, a survey on traceability," *IOSR Journal of Computer Science (IOSR-JCE)*, vol. 3, no. ICAET-2014, pp. 66–70, 2014.

[47] H. E. Salman, A. Seriai, C. Dony, and R. Al-Msie'deen, "Recovering traceability links between feature models and source code of product variants," in *Proceedings of the VARiability for You Workshop - Variability Modeling Made Useful for Everyone, VARY '12, Innsbruck, Austria, September 30, 2012*, Ø. Haugen, J. Jézéquel, A. Wasowski, B. Møller-Pedersen, and K. Czarnecki, Eds. ACM, 2012, pp. 21–25.

[48] X. Chen, J. G. Hosking, and J. Grundy, "Visualizing traceability links between source code and documentation," in *2012 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2012, Innsbruck, Austria, September 30 - October 4, 2012*, M. Erwig, G. Stapleton, and G. Costagliola, Eds. IEEE, 2012, pp. 119–126.

[49] X. Chen, J. G. Hosking, J. C. Grundy, and R. Amor, "Dctracvis: a system retrieving and visualizing traceability links between source code and documentation," *Autom. Softw. Eng.*, vol. 25, no. 4, pp. 703–741, 2018. [Online]. Available: https://doi.org/10.1007/s10515-018-0243-8

[50] R. Al-Msie'deen, *Object-oriented Software Documentation*. Lap Lambert Academic Publishing, 2019.

[51] R. A. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Mining features from the object-oriented source code of software variants by combining lexical and structural similarity," in *IEEE 14th International Conference on Information Reuse & Integration, IRI 2013, San Francisco, CA, USA, August 14-16, 2013*. IEEE Computer Society, 2013, pp. 586–593.

[52] R. Al-Msie'deen, H. E. Salman, A. H. Blasi, and M. A. Alsuwaiket, "Naming the identified feature implementation blocks from software source code," *Journal of Communications Software and Systems*, vol. 18, no. 2, pp. 101–110, 2022.

[53] R. Al-Msie'deen. (2018) Drawing shapes software. [Online]. Available: https://sites.google.com/site/ralmsideen/tools

[54] R. A. Al-Msie'deen, "Automatic labeling of the object-oriented source code: The lotus approach," *Science International-Lahore*, vol. 30, no. 1, pp. 45–48, 2018.

[55] R. Al-Msie'deen and A. Blasi, "The impact of the object-oriented software evolution on software metrics: The iris approach," *Indian Journal of Science and Technology*, vol. 11, no. 8, pp. 1–8, 2018.

[56] A. Nayak, H. Timmapathini, V. Murali, K. Ponnalagu, V. G. Venkoparao, and A. Post, "Req2spec: Transforming software requirements into formal specifications using natural language processing," in *Requirements Engineering: Foundation for Software Quality - 28th International Working Conference, REFSQ 2022, Birmingham, UK, March 21-24, 2022, Proceedings*, ser. Lecture Notes in Computer Science, V. Gervasi and A. Vogelsang, Eds., vol. 13216. Springer, 2022, pp. 87–95.

[57] R. Al-Msie'deen, "Reverse engineering feature models from software variants to build software product lines: REVPLINE approach," Ph.D. dissertation, Montpellier 2 University, France, 2014. [Online]. Available: https://tel.archives-ouvertes.fr/tel-01015102

[58] R. A. Al-Msie'deen and A. Blasi, "Supporting software documentation with source code summarization," *International Journal of Advanced and Applied Sciences*, vol. 6, no. 1, pp. 59–67, 2019.

[59] R. A. A. Al-Msie'deen. (2023) YamenTrace approach. [Online]. Available: https://drive.google.com/drive/folders/16XbIgKQu1LylADtWfPQNSrNEaSKEQiqy

[60] V. Bauer, T. Volke, and S. Eder, "Combining clone detection and latent semantic indexing to detect re-implementations," in *10th International Workshop on Software Clones, IWSC@SANER 2016, Osaka, Japan, March 15, 2016*. IEEE Computer Society, 2016, pp. 23–29.

[61] R. A. Al-Msie'deen, "Softcloud: A tool for visualizing software artifacts as tag clouds," *Mutah Lil-Buhuth wad-Dirasat - Natural and Applied Sciences Series*, vol. 37, no. 2, pp. 93–116, 2022.

[62] R. Al-Msie'deen, "Tag clouds for object-oriented source code visualization," *Engineering, Technology & Applied Science Research*, vol. 9, no. 3, pp. 4243–4248, 2019. [Online]. Available: https://doi.org/10.48084/etasr.2706

[63] R. A. Al-Msie'deen, "Tag clouds for software documents visualization," *International Journal on Informatics Visualization*, vol. 3, no. 4, pp. 361–364, 2019.

[64] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[65] D. A. Grossman and O. Frieder, *Information Retrieval - Algorithms and Heuristics, Second Edition*, ser. The Kluwer International Series on Information Retrieval. Kluwer, 2004, vol. 15.

[66] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, and S. Vauttier, "Documenting the mined feature implementations from the object-oriented source code of a collection of software product variants," in *The 26th International Conference on Software Engineering and Knowledge Engineering, Hyatt Regency, Vancouver, BC, Canada, July 1-3, 2013*, M. Reformat, Ed. Knowledge Systems Institute Graduate School, 2014, pp. 138–143.

[67] A. Delater, "Tracing requirements and source code during software development," Ph.D. dissertation, Heidelberg University, 2013. [Online]. Available: https://d-nb.info/1047691701

[68] K. Porter, "Approximate string matching and filesystem metadata carving: A study of improving precision and recall for assisting the digital forensics backlog," Ph.D. dissertation, Norwegian University of Science and Technology, Trondheim, Norway, 2022. [Online]. Available: https://hdl.handle.net/11250/2976608

[69] A. Marcus, X. Xie, and D. Poshyvanyk, "When and how to visualize traceability links?" in *The 3rd International Workshop on Traceability in Emerging Forms of Software Engineering, co-located with the ASE 2005 Conference, TEFSE@ASE 2005, Long Beach, CA, USA, November 88, 2005*, J. I. Maletic, J. Cleland-Huang, J. H. Hayes, and G. Antoniol, Eds. ACM, 2005, pp. 56–61.

[70] E. Figueiredo. (2008) Mobile media java implementation. [Online]. Available: https://homepages.dcc.ufmg.br/~figueiredo/spl/icse08/

[71] E. Figueiredo, N. Cacho, C. Sant'Anna, M. Monteiro, U. Kulesza, A. Garcia, S. Soares, F. C. Ferrari, S. S. Khan, F. C. Filho, and F. Dantas, "Evolving software product lines with aspects: an empirical study on design stability," in *30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10-18, 2008*, W. Schäfer, M. B. Dwyer, and V. Gruhn, Eds. ACM, 2008, pp. 261–270.

[72] Iowa-State-University. (2022) Health watcher - java code. [Online]. Available: https://ptolemy.cs.iastate.edu/design-study/

[73] P. Greenwood, T. T. Bartolomei, E. Figueiredo, M. Dósea, A. F. Garcia, N. Cacho, C. Sant'Anna, S. Soares, P. Borba, U. Kulesza, and A. Rashid, "On the impact of aspectual decompositions on design stability: An empirical study," in *ECOOP 2007 - Object-Oriented Programming, 21st European Conference, Berlin, Germany, July 30 - August 3, 2007, Proceedings*, ser. Lecture Notes in Computer Science, E. Ernst, Ed., vol. 4609. Springer, 2007, pp. 176–200.

[74] S. Soares, E. Laureano, and P. Borba, "Implementing distribution and persistence aspects with aspectJ," in *Proceedings of the 2002 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages and Applications, OOPSLA 2002, Seattle, Washington, USA, November 4-8, 2002*, M. Ibrahim and S. Matsuoka, Eds. ACM, 2002, pp. 174–190.

[75] S. Soares. (2022) Requirements document - health-watcher (version 1.0). [Online]. Available: https://drive.google.com/drive/folders/16XbIgKQu1LylADtWfPQNSrNEaSKEQiqy

[76] R. Al-Msie'deen. (2023) Mobile media code - xml file. [Online]. Available: https://drive.google.com/file/d/14zt2A9hWPfQlK-52Nv5iG59EGx349U25/view?usp=share_link

[77] L. P. Tizzei, M. Dias, C. M. Rubira, A. Garcia, and J. Lee. (2011) Mobile media test bed. [Online]. Available: https://www.ic.unicamp.br/~tizzei/mobilemedia/index.html#scenarios-table

[78] G. Fischer, J. Lusiardi, and J. W. von Gudenberg, "Abstract syntax trees - and their role in model driven software development," in *Proceedings of the Second International Conference on Software Engineering Advances (ICSEA 2007), August 25-31, 2007, Cap Esterel, French Riviera, France*. IEEE Computer Society, 2007, p. 38.

[79] J.-R. Falleri and X. Dolques. (2010) Erca tool. [Online]. Available: http://code.google.com/p/erca/

[80] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull, "Graphviz and dynagraph - static and dynamic graph drawing tools," in *Graph Drawing Software*, M. Jünger and P. Mutzel, Eds. Springer, 2004, pp. 127–148. [Online]. Available: https://doi.org/10.1007/978-3-642-18638-7_6

[81] R. Al-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Feature location in a collection of software product variants using formal concept analysis," in *Safe and Secure Software Reuse - 13th International Conference on Software Reuse, ICSR 2013, Pisa, Italy, June 18-20. Proceedings*, ser. Lecture Notes in Computer Science, J. M. Favaro and M. Morisio, Eds., vol. 7925. Springer, 2013, pp. 302–307.

[82] R. Al-Msie'deen, A. Seriai, and M. Huchard, *Reengineering Software Product Variants Into Software Product Line: REVPLINE Approach*. LAP LAMBERT Academic Publishing, 2014.

[83] G. Deshpande, "Requirements dependency extraction: Advanced machine learning approaches and their ROI analysis," Ph.D. dissertation, University of Calgary, Alberta, Canada, 2022. [Online]. Available: http://hdl.handle.net/1880/114394

**Ra'Fat Al-Msie'Deen** is an Associate Professor in the Software Engineering department at Mutah University since 2014. He received his PhD in Software Engineering from the Université de Montpellier, Montpellier - France, in 2014. He received his MSc in Information Technology from the University Utara Malaysia, Kedah - Malaysia, in 2009. He got his BSc in Computer Science from Al-Hussein Bin Talal University, Ma'an - Jordan, in 2007. His research interests include software engineering, requirements engineering, software product line engineering, feature identification, word clouds, and formal concept analysis.