VOL 4 (2020) NO 2

e-ISSN : 2549-9904
ISSN : 2549-9610

INTERNATIONAL JOURNAL ON INFORMATICS VISUALIZATION

Concerns-Based Reverse Engineering for Partial Software Architecture Visualization

Hind Alamin M[#], Hany H Ammar[*]

[#] *College of Computer Science and Information Technology, Sudan University of Science and Technology, SUST University, SUDAN*

[*] *Lane Computer Science and Electrical Engineering Depart, College of Engineering and Mineral Resources, West Virginia University, USA*
E-mail: hindalamin@sustech.edu, hindalamin@gmail.com, Hany.Ammar@mail.wvu.edu, ammar.hany@gmail.com

***Abstract*— Recently, reverse engineering (RE) is becoming one of the essential engineering trends for software evolution and maintenance. RE is used to support the process of analyzing and recapturing the design information in legacy systems or complex systems during the maintenance phase. The major problem stakeholders might face in understanding the architecture of existing software systems is that the knowledge of software architecture information is difficult to obtain because of the size of the system, and the existing architecture document often is missing or does not match the current implementation of the source code. Therefore, much more effort and time are needed from multiple stakeholders such as developers, maintainers and architects for obtaining and re-documenting and visualizing the architecture of a target system from its source code files. The current works is mainly focused on the developer viewpoint. In this paper, we present a RE methodology for visualizing architectural information for multiple stakeholders and viewpoints based on applying the RE process on specific parts of the source code. The process is driven by eliciting stakeholders' concerns on specific architectural viewpoints to obtain and visualize architectural information related these concerns. Our contributions are three fold: 1- The RE methodology is based on the IEEE 1471 standard for architectural description and supports concerns of stakeholder including the end-user and maintainer; 2- It supports the visualization of a particular part of the target system by providing a visual model of the architectural representation which highlights the main components needed to execute specific functionality of the target system, 3- The methodology also uses architecture styles to organize the visual architecture information. We illustrate the methodology using a case study of a legacy web application system.**

***Keywords*— Reveres Engineering, Software Architecture visualization, Extracting Architectural Information, Visualizing Architectural Information.**

I. Introduction

Nowadays, reverse engineering (RE) is becoming one of essential engineering trends for software evolution and maintenance. Generally; RE is defined as the way of analysing an existing software system to identify its current components and the dependencies between these components to recover design information, and create new forms of system representations [1]-[4]. The core of RE consists of extracting information from the available software artifacts (such as: source code) and representing it into visual models to be understandable by stakeholders [3], [5]. The main objectives of RE are focused on generating alternative views of system's architecture, recapture design information, re-documentation of software system, facilitate software system's reuse, and represent software systems at higher level of abstractions (by putting the system's users in the maintenance loop so that users can give feedback on the information related the target system). Furthermore; RE is used to support recapturing the design information for restructuring the architecture into more maintainable architecture [3], [5]. Hence, most of the companies rely on reengineering the legacy systems which are important for their business process and keep them in operations [3].

Moreover, software documentation is essential for the system's stakeholders (such as: developers, end-users, testers, maintainers, architects, system administrators, etc.) to decide on activities in order to evolve and maintain the software system. For example, "source code" is considered as the detailed documentation for the software system implementation, and in most cases, it is the only source of information that up to date and available for legacy software systems. Accordingly; IEEE_1219 standards recommend the RE as a key supporting technology to deal with source code as the "reliable representation" of software systems [3], [5].

Recovering and documenting software architectures (either fully or partially) has been an area of active research where programmers, architects, maintainers, testers and software engineers spend a lot of time using their expertise in resolving such problems of mapping existing source code of a target system into architecture components and for supporting the understand-ability and maintainability of software systems.

Previous research made great progress to overcome the problems of documenting and recovering software architectures to reflect the system's changes at the code level. However, to deal with complex legacy systems, there is a significant need to develop a new RE approaches or methods for documenting the only part of the architecture in order to simplify and visualize the available information of complex architectures. This should be based on stakeholders concerns and their decisions about the architecture of the target system. Hence, it's important to determine what to look for and focus in obtaining specific information on the architecture of the implemented software system.

This paper represents RE methodology for extracting a particular architectural information based on applying RE process on specific parts of the implemented source code to support the understand-ability and maintainability process for particular parts of the software system.
The rest of the paper is organized as follows: Section 2; presents the proposed RE methodology and the detailed design of RE methodology's phases. Section 3; describes how to apply RE methodology's phases to a case study. Section 4; compares the proposed methodology with related works. Finally, Section 5 concludes with the main contributions and highlights the future research

II. The Proposed Reverse Engineering Methodology

This section presents an overview of the proposed RE methodology. We discuss the principles of proposed methodology, and describe the detailed design of the main phases of the methodology.

A. *Overview of the Proposed RE Methodology*

The main goal of proposed methodology is to define a RE process for extracting particular architectural information based on stakeholder's viewpoints and concerns related to the target software system.

The RE methodology is based on three main concepts defined in the IEEE1471 standard for architectural description such as (stakeholder, viewpoint and concern). The main idea is to elicit stakeholders' concern on specific architectural viewpoint of the target software system. Then we apply the RE process to extract and document a particular architectural information about the target software system driven by the elicited concern.

The extraction process of RE methodology is driven by addressing the specific concerns of the stakeholder(s) for extracting only partial architectural information. Therefore, it's doesn't address the RE of the whole architecture of target software system. The general overview of RE methodology is shown in Figure 1 as follow. The inputs are the source code and documentation as well as the stakeholders concerns regarding the software system. The output is a model of a particular architectural information based on the specific concerns.

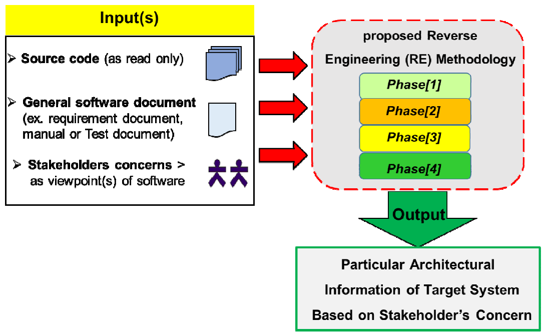

Input(s)
Source code (as read only)
General software document (ex. requirement document, manual or Test document)
Stakeholders concerns > as viewpoint(s) of software
proposed Reverse Engineering (RE) Methodology
Phase[1]
Phase[2]
Phase[3]
Phase[4]
Output
Particular Architectural Information of Target System Based on Stakeholder's Concern

Fig. 1 Overview of RE Methodology

B. *RE Methodology Principles*

The principles of RE methodology are summarized as:

- RE methodology is based on three concepts defined in the IEEE1471 standard for architectural description as shown in Figure 2. These concepts are described as follows [6]-[8]:
  - ***Stakeholder*** is a person, group or entity with an interest in the realization of the architecture.
  - ***Concern*** related to specific functional or non-functional requirements of the software system is defined as: a concern to a requirement, an objective, an intention, or aspiration which a stakeholder has for the software system.
  - ***Viewpoint*** defines the perspective from which the view is taken; and each viewpoint covers a set of concerns related to one or more stakeholder(s).

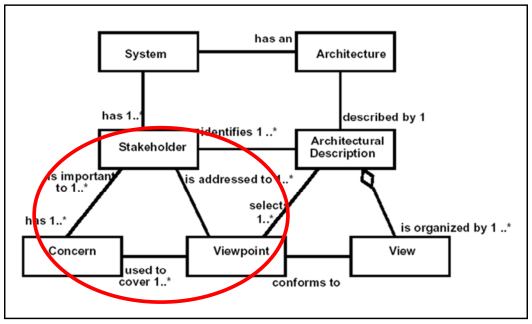


Fig.2 IEEE 1471 Conceptual Framework. Adapted from [8, p15]

- Our RE methodology extends additional stakeholders such as: end-user, maintainer, analyst, architect and tester.
- The Methodology supports the understand-ability and maintainability of legacy software systems using partial architecture visualization

C. *RE Methodology Phases*

The Methodology consists of four phases described as follows:

- ***Phase(1):*** Define stakeholders concerns based on one of the architectural viewpoints.

- ***Phase(2):*** Elicit specific stakeholder's concern.
- ***Phase(3):*** Extract related requirement information based on the elicited concern.
- ***Phase(4):*** Apply the RE process for extracting the particular architectural information driven by the extracted requirement information.

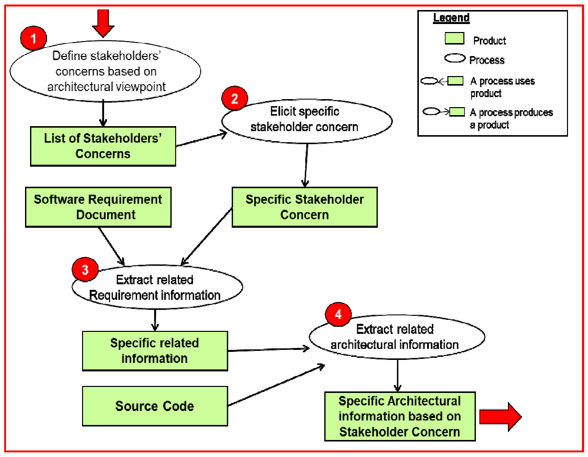

Legend
Product
Process
A process uses product
A process produces a product
1
Define stakeholders' concerns based on architectural viewpoint
List of Stakeholders' Concerns
2
Elicit specific stakeholder concern
Software Requirement Document
Specific Stakeholder Concern
3
Extract related Requirement information
Specific related information
4
Extract related architectural information
Source Code
Specific Architectural information based on Stakeholder Concern

Fig.3 The RE Methodology's Phases

As shown in Figure3; the phases of RE methodology is described using a process modelling language. The following paragraphs elaborate on the detailed design of each phase:

1) Define stakeholders concerns based on architectural viewpoint:

This phase is based on the definition of "stakeholders" and "concerns" in IEEE 1471 standard for architectural description. We follow the classification of architectural viewpoints that are presented in literature. The activities in this phase includes the following two steps:

- *Select a viewpoint from a given catalog which describes specific architectural viewpoint for the target software system.*
- *Categorize common stakeholders related to the selected viewpoint.*

1.1) Select a viewpoint from a given catalog:

The definitions of stakeholders' concerns are based on a set of architectural viewpoints about software system. These viewpoints have been considered by several researchers form different perspectives [6], [9], [10-18]. We choose the classification of viewpoints catalog that were presented by Nick Rozanski and Eoin Wood in 2005 [10], [17]. They developed a set of core viewpoints which are based on extending the well-known "4+1" standard view model of software architectures (Logical, Process, Physical, and Development) that was defined by Philippe Kruchten in 1995. The viewpoint catalog includes six core viewpoints for information systems architecture, namely: Functional viewpoint, Information viewpoint, Concurrency viewpoint, Development viewpoint, Deployment viewpoint, and Operational viewpoint (see Figure 4). Each one of these viewpoint defines a set of concerns related to one or more stakeholder(s).

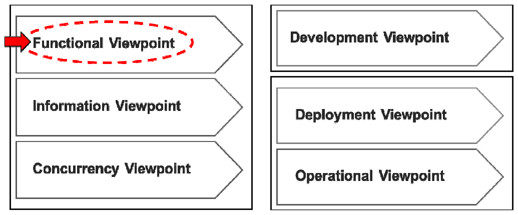


Fig.4 The Viewpoints Catalog [10, 17]

Summarized the viewpoints catalog in Figure 4; the first three viewpoints: Functional, Information and Concurrency characterize the fundamental organization of the software system. The development viewpoint exists to support the system's construction. The deployment and operational viewpoints characterize the system's runtime environment [10], [17]. The last three viewpoints mainly covers the concerns of the developers and maintainers stakeholders.

The methodology we present in this paper is focused on the "Functional viewpoint" from the catalog of Nick et al. The justification for selecting this "Functional viewpoint" is that it is applicable to all types of software systems; and reflects the essential architectural information for most of the stakeholders (such as: maintainer, end-user, developer, system administrator, tester, acquirer, assessor and communicator).

Furthermore, the functional viewpoint includes a set of general stakeholders' concerns which reflect and realize the essential and basic architectural information about the software system. This information include the internal structure which determines the main elements of software system, the responsibilities of each element and primary interactions between elements, the functional capabilities that defines what the specific action(s) that system should take in a given situation, and the functional design philosophy that reflects how the system will work step by step from the user's perspective as represented in Table 1.

TABLE 1
FUNCTIONAL VIEWPOINT CATALOG [10], [17]

Functional viewpoint	
Description	Describes the system's runtime functional elements and their responsibilities, interfaces, and primary interactions between these elements.
General Concerns	▪ Internal structure ▪ Functional capabilities ▪ Functional design philosophy ▪ The external interfaces
Related Stakeholders	End-User, Maintainer, Developer, Tester, Acquirer, System Administrator, Assessor and Communicator.

1.2) Categorize common stakeholders concerns related to the selected viewpoint:

This step includes the categorization of common stakeholders and their architectural concerns based on selected viewpoint catalog. The main idea is to address the following points: who are the stakeholders of target software system; and which concerns do they have according to the selected viewpoint.

Table 2 represents the categorization of stakeholder's and their architectural concerns based on selected functional viewpoint catalog.

TABLE 2
FUNCTIONAL VIEWPOINT: STAKEHOLDERS AND CONCERNS [10], [17]

Functional viewpoint > Categorize the Stakeholder Concern(s)	
Acquirer Users	▪ **Internal structure:** (The main elements of system, **responsibilities** of each elements and **primary interactions** between elements). ▪ **Functional capabilities:**(define what the specific action(s) the system should taken in a given situation). ▪ **The external interfaces**
System Administrator	▪ **Internal structure** ▪ **Functional design philosophy**
Maintainer Tester Communicator Developer	▪ **Internal structure** ▪ **Functional capabilities** ▪ **Functional design philosophy** ▪ **The external interfaces**

2) *Elicit specific stakeholder concern:*

In "*Phase*(2)" we define an elicitation process to clearly describe a specific concern from the general architectural concerns defined in Phase(1).

The specific concern of the stakeholder is defined as a specific question that can be used to query the functional requirements document of the target software system. It can for example be used to select related use cases defined in the use case diagram of the requirements model. Accordingly, each elicited concern should have a question format and has two elements as follows:

- *CIDn*: refers to concern *ID* (where *n* is an integer number), which written in dotted diamond box.
- *Question:* refers to elicited concern from the functional scenario of a target software system, and written in dotted rectangular box.

As shown in Figure 5; the association between the functional requirement (FR) and elicited concern appears with dotted lines in the use case diagram of the target system. Moreover, it's possible to have multiple elicited concerns for one FR which are numbered as CID1, CID2,.. ,CIDn

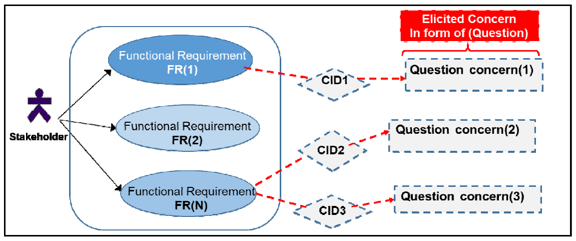


Fig.5 Elicitation of Specific Stakeholder's Concern(s)

3) *Extract related requirement information based on elicited stakeholder's concern*

In "Phase(3)" which describes how to extract the related requirement information related to the elicited functional concern produced in Phase(2). The stakeholder's functional concern should be focused on the functionality offered by the target software system.

To support the activities of this phase, we developed a prototype tool which has a graphical user interface (GUI). The tool allows stakeholders to enter a specific concern in form of a "query". The specific concern will be elicited from the functional requirements repository assumed to be available for the target software system.

The tool extracts a set of related requirement information based on elicited concern, and creates a trace link between elicited concern and its relevant information. Figure 6 shows screen shots described as follows:

3.1) Extraction of related requirement information:

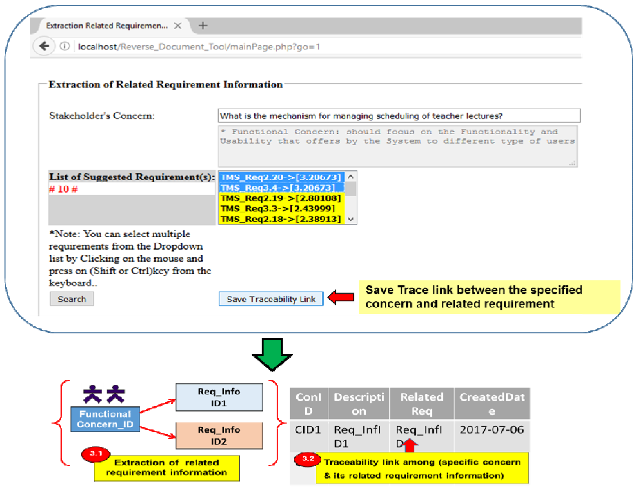
The extraction process starts by accessing the requirement repository and filtering all relevant information related the specified concern. Furthermore, the extraction process is achieved using the Full-Text indexing and searching mode technique as described in [19, 20].

The Full-Text indexing and searching technique allows to implement keyword based filtering and sorting through several searches mode. The searching techniques is achieved using natural language searching mode which interprets the search for specific functional concern (in form of user query); then performs filtering process and ranking of the relevant information related to the specified concern. The main results are displayed in a dropdown menu and sorted into three categories:

- *High weight:* appears in green color and represents highly relevant requirement information related the specified functional concern,
- *Medium weight:* appears in yellow color and represents the medium relevance requirement information related the specified functional concern,
- *Low weight:* represents low relevance values of requirement information, and appears in red color.

3.2) Traceability among specific concern and its related requirement information:

The traceability process is performed after the extraction process. The main idea is to create a trace link among the extracted concerns and its relevant information using the tool as shown in Figure 6


Extraction Related Requiremen...
localhost/Reverse_Document_Tool/mainPage.php?go=1

Extraction of Related Requirement Information

Stakeholder's Concern: What is the mechanism for managing scheduling of teacher lectures?

* Functional Concern: should focus on the Functionality and Usability that offers by the System to different type of users

List of Suggested Requirement(s):
10

TMS_Req2.20->[3.20673]
TMS_Req3.4->[3.20673]
TMS_Req2.19->[2.80108]
TMS_Req3.3->[2.43999]
TMS_Req2.18->[2.38913]

*Note: You can select multiple requirements from the Dropdown list by Clicking on the mouse and press on (Shift or Ctrl)key from the keyboard..

Search
Save Traceability Link
Save Trace link between the specified concern and related requirement
Functional Concern_ID
Req_Info ID1
Req_Info ID2
3.1
Extraction of related requirement information
3.2
Traceability link among (specific concern & its related requirement information)

ConID	Description	Related Req	CreatedDate
CID1	Req_InfID1	Req_InfID	2017-07-06

Fig.6 Tracing Specific Concern to its Related Requirement Information

4) RE for extracting particular architectural information

The final phase, “Phase(4)” is based on using the extracted requirement information produced from the previous. This phase includes two key activities as follows:

- ➢ *RE process for extracting specific source code files,*
- ➢ *Representation of the particular architectural information based on the extracted code files.*

4.1) RE process for extracting specific source code files:

This RE process is achieved by applying a code analyser process which performs static analysis on source code files to determine and trace which set of code files are used to implement specific functionality reflected by the extracted requirement information in *Phase*(3). The code analyzer process includes three key steps as shown in Figure 7. We describe these steps in the following paragraphs:

- ➢ Select the starting point for tracking the execution of a specific functionality represented by extracted requirement information. For examples: page file, class, method or function from code elements. Notably, the selection of a starting point can be performed by using references from existing documents such as the user manual, or the software testing document.
- ➢ Track the execution of selected starting code element and analyze the code extraction contents and gather all related code elements.
- ➢ Extract related code elements in form of main code element and its related elements. The relation between code elements can be describes as:
  - ▪ *require relation* is used to describe the relations between code files and show the dependences of these files within the software system, or
  - ▪ *contain relation* is used to describe that code file contains a set of functions that are used to execute specific functionality of the system, or
  - ▪ *Call relation* is used to describe the relation between code elements and how different functions interact with each other.

As summarized; the whole process of code analyzer is achieved by using a static analyser tool called doxygen tool [21]. The doxygen tool is used to extract code structure from the existing source code files, and visualize the relations between various code elements according the type of source code of target software system in the form of function call graphs, or dependency graphs, or inheritance diagrams, or collaboration diagrams, which are all generated automatically by the tool [21].

4.2) Representation of the particular architectural information:

Generally; the representation process includes two key steps; mapping the extracted code elements into a component model; and visualizing the architectural information using architecture styles. The following paragraphs describe the details of these steps:

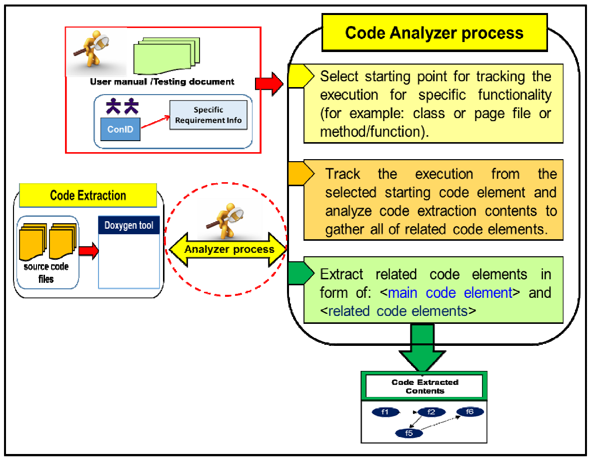

Code Analyzer process
User manual /Testing document
ConID
Specific Requirement Info
Select starting point for tracking the execution for specific functionality (for example: class or page file or method/function).
Track the execution from the selected starting code element and analyze code extraction contents to gather all of related code elements.
Extract related code elements in form of: <main code element> and <related code elements>
Code Extraction
Doxygen tool
source code files
Analyzer process
Code Extracted Contents
f1
f2
f6
f5

Fig.7 Code Analyzer Process

- ➢ *Mapping the extracted code elements into a component architecture model*: This step involves the process of organizing the extracted code elements into a component model to make an explicit mapping between software architecture and the code elements of the target system.

  It is important to note that this process assumes that the term “*component*” can be associated with a code element such as a code file, a webpage file, a class, a class method, a function, or either as a group of related methods or functions which are used frequently together in the execution of specific system’s functionality.

  For example, suppose the given code element is a webpage source file called *page_Layout.php*, this webpage file can be mapped into a “*Page Layout*” component which contains the set of functions or methods that are used to execute specific system’s functionality as in the following example shown in Figure 8.

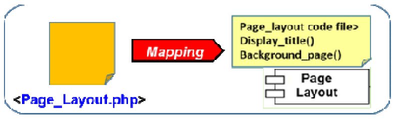


Fig.8 Example of Mapping Code’s Element into Component

- ➢ *Visualizing architectural information using architecture styles:* The whole purpose of this process is to create a logical model, so that the architectural information is visualized and represented in the form of logical component model which helps the stakeholders to gain insight of the architecture information related to their functional concerns about a target system.

  The visualization process starts by selecting the structure of the architecture which is mainly based on the application’s type as introduced in [20] called archetypes. The Microsoft guide for application architecture defines these archetypes as shown in Table 4 [22].

  The application archetypes includes the architecture’s structure for common types of applications such as web applications, rich client applications, rich internet applications, service applications and mobile applications as summarized in Table 4. However, beside these archetypes, the Microsoft's guide also contains details of some specialized application types such as hosted and cloud services, and office business applications.

TABLE 4
APPLICATION ARCHETYPES SUMMARY [22, P. 226]

Application Type	Description
Web applications	The applications of this type are typically support connected scenarios and can support different browsers running on a range of operating systems and platforms.
Rich client applications	The applications are usually developed as standalone applications with a graphical user interface that displays data using a range of controls. Those applications can be designed for disconnected and occasionally connected scenarios if they need to access remote data or functionality.
Rich Internet applications	The applications of this type can be developed to support multiple platforms and multiple browsers, displaying rich media or graphical content. Rich Internet applications run in a browser sandbox that restricts access to some features of the client.
Service applications	The services expose shared business functionality and allow clients to access them from a local or a remote system. Service operations are called using messages, based on XML schemas, passed over a transport channel. The goal of this type of application is to achieve loose coupling between the client and server.
Mobile applications	Applications of this type can be developed as thin client or rich client applications. Rich client mobile applications can support disconnected or occasionally connected scenarios. Web or thin client applications support connected scenarios only. Device resources may prove to be a constraint when designing mobile applications.

The architecture of each of the archetype application can be defined using architecture styles. For example, the guide in [22] describes a layered architecture style for web applications. The visualization process we adopt is performed using these architectural styles. This is based, for example, on grouping related components in web applications as a three-layered architecture which consists of a presentation layer, business layer and data layer as shown in Figure 15. Each layer should include specific components described as follows:

- *Presentation Layer:* responsible for managing user interaction with software system, and generally consists of components that provide a common bridge into the core business logic that encapsulated in the business layer.
- *Business Layer*: which implements the core functionality of software system, and encapsulates the relevant business logic. It generally consists of components, some of which may expose service interfaces that other callers can use.
- *Data Access Layer*: provides access to data hosted within the system, and data exposed by other networked systems; perhaps accessed through services.

To summarize; *Phase* (4) includes two key steps. The first step deals with organizing the extracted code elements into a component model to make an explicit mapping between the system's architecture and code elements. The second step deals with using archetypes and architecture styles to visualize the architecture model. We give the example of a layered architectural style for web applications. The visual model represents the extraction of the partial architectural information in the form of a logical model. This architectural information helps stakeholders to answer their architectural concerns about a target system. The next section describes how to apply the methodology phases to a practical case study.

III. APPLY RE METHODOLOGY TO A CASE STUDY

The following sections describe how to implement the RE methodology phases using a legacy web application as a practical case study. The section starts by giving an overview of the selected software system, and describes the main reasons for selecting this system. Then we describe the details of applying each phase of the methodology to the case study.

A. *Selecting Software System for a Case Study*

The case study selected is a web application system called Timetable Management System (TMS). TMS was developed by the Computer Center at Sudan University of Science and Technology (SUST) in 2008.

TMS is a Web-based open source system which was built for Sudanese Universities using MySQL database and PHP web page language with Arabic interface; and it provides high flexible features for managing and controlling the scheduling of lectures' times for students at Sudanese universities.

Moreover; TMS is flexible to accept changes that occur in schedules for all colleges at the university during the academic year without an overlap in specified slot times between these colleges.

We chose this system for the following reasons. TMS software is a diverse software implemented as a combination of both front-end PHP, JavaScript and HTML code plus a back-end MySQL database. It is an example of an application with multiple components implemented with different technologies. TMS is considered to be a legacy system implemented with more than 10 years old technologies since 2008. The documentation of TMS's architecture is missing, and the system documentation needs to reflect its current architectural representation in order to be reengineered with new technologies. Recovering the particular architectural information of the system is essential to support the system's understand-ability and maintainability.

Table 5 represents the general description about the TMS's source code contents.

TABLE 5
TMS SOURCE CODE OVERVIEW

System Name	**T**imetable **M**anagement **S**ystem(**TMS**)
Description	The core of source code is mainly PHP webpage source files (written with PHP procedural function code style, and its non-object oriented code style).
PHP Source Files	110
Total LOC	30364
Number of Functions Code	148

B. *Applying RE Methodology Phases to the Case Study*

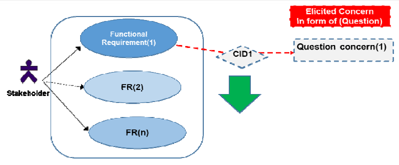
The following paragraphs elaborate on the details of applying each phase of RE methodology:

1) Define a set of Stakeholders Concerns: *we define a set of* stakeholders' concerns base on the "*Functional viewpoint*" of the TMS system. The primary TMS's stakeholders are:

- *End-User*: who defines the system's functionality and ultimately make use of it. TMS has three end-users (*College Admin*, *Teachers*, and *Students*).
- *Maintainer*: who manages the reengineering and improvements of the system.

2) Elicit a specific Stakeholders Concern: The elicitation process is focused on a selecting a particular functional concern related to a use-case or a major functionality offered by the system to different type of users. The main idea is to elicit a specific concern such as "CID1" shows in Figure 9 bellow.

General Concern: The main functionalities that offers by TMS system to different types of end-users (College Admin, Student , Teacher)?

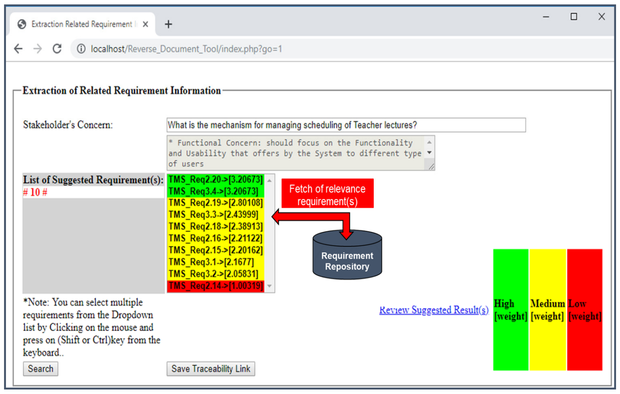


CID	Elicit Specific Concern (as Question Format)	Related Stakeholder
✓ CID1	What is the mechanism for managing scheduling of Teacher lectures?	**End-user** (Teacher and College Admin)

Fig.9 Elicit a Specific Stakeholder Functional Concern

3) Extract related requirements information based on the elicited stakeholder's concern: TMS has 34 functional requirements; this phase assumes that all of TMS's functional requirements are already existed in a "requirement repository". The extraction process starts by accessing the requirement repository and filtering all of relevant information based on the elicited concern. Then create a trace link between its relevant information. The phase is achieved by using the tool as described in section 3.1 and section 3.2.

Using the tool we obtain the results shown in Figure10. The results of the search shows *ten requirements* information displayed in a dropdown menu and sorted by ranking using *three categories* as follow: *High weight*(2) appears in green color, *Medium weight*(7) appears in yellow color, and *Low weight*(1) appears in red color.


Extraction Related Requirement
localhost/Reverse_Document_Tool/index.php?go=1
Fetch of relevance requirement(s)
Requirement Repository
Review Suggested Result(s)
High [weight]
Medium [weight]
Low [weight]
Search
Save Traceability Link

Extraction of Related Requirement Information

Stakeholder's Concern: What is the mechanism for managing scheduling of Teacher lectures?

```
* Functional Concern: should focus on the Functionality
and Usability that offers by the System to different type
of users
```

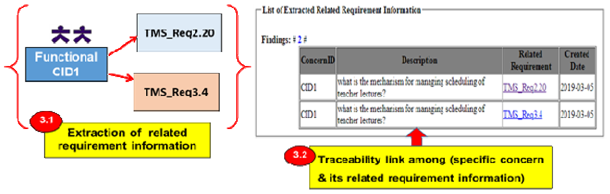
List of Suggested Requirement(s): # 10 #

```
TMS_Req2.20->[3.20673]
TMS_Req3.4->[3.20673]
TMS_Req2.19->[2.80108]
TMS_Req3.3->[2.43999]
TMS_Req2.18->[2.38913]
TMS_Req2.16->[2.21122]
TMS_Req2.15->[2.20162]
TMS_Req3.1->[2.1677]
TMS_Req3.2->[2.05831]
TMS_Req2.14->[1.00319]
```

*Note: You can select multiple requirements from the Dropdown list by Clicking on the mouse and press on (Shift or Ctrl)key from the keyboard..

Fig.10 Extraction of Related Requirements Information

Additionally, the creation of a trace link is performed in order to link the elicited concern with its relevant information produced from the extraction process as shown in Figure 11.


Functional CID1
TMS_Req2.20
TMS_Req3.4
3.1
Extraction of related requirement information
3.2
Traceability link among (specific concern & its related requirement information)

List of Extracted Related Requirement Information

Findings: # 2 #

ConcernID	Description	Related Requirement	Created Date
CID1	what is the mechanism for managing scheduling of teacher lectures?	TMS_Req2.20	2019-03-05
CID1	what is the mechanism for managing scheduling of teacher lectures?	TMS_Req3.4	2019-03-05

Fig.11 Traceability among specific Concerns and their related Requirement Information.

4) Extracting architectural information: This final phase is achieved by applying the RE process at code level to perform following key steps:

4.1) Extracting specific source code files:

The code extraction process is performed by using a static code analyzer as described in section 4.1. Using the existing TMS source code file, we determine which set of source code files are used to implement the specific functionality of the system specified in the previous steps. Notably, the selection of a starting point for the extraction process is performed by returning to TMS's user manual in order to track the starting point for "*TMS_Req2.20*" execution. The main output of this process is to extract the call graph to obtain and visualize the dependencies between the function elements which are used to execute specific functionality in the system as described in Figure12 and Figure13.

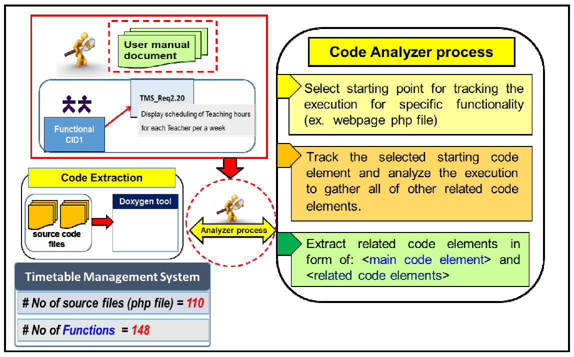

User manual document
TMS_Req2.20
Display scheduling of Teaching hours for each Teacher per a week
Functional CID1
Code Extraction
source code files
Doxygen tool
Analyzer process
Timetable Management System
# No of source files (php file) = 110
# No of Functions = 148
Code Analyzer process
Select starting point for tracking the execution for specific functionality (ex. webpage php file)
Track the selected starting code element and analyze the execution to gather all of other related code elements.
Extract related code elements in form of: <main code element> and <related code elements>

Fig.12 Applying Code Analyzer Process

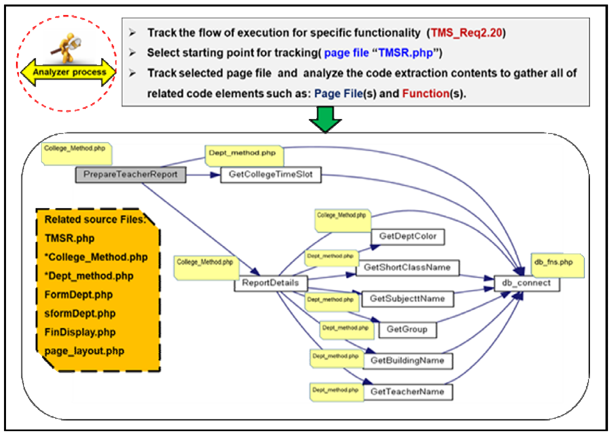


Fig.13 Extracted Call Graph for Executing "TMS_Req2.20" Functionality

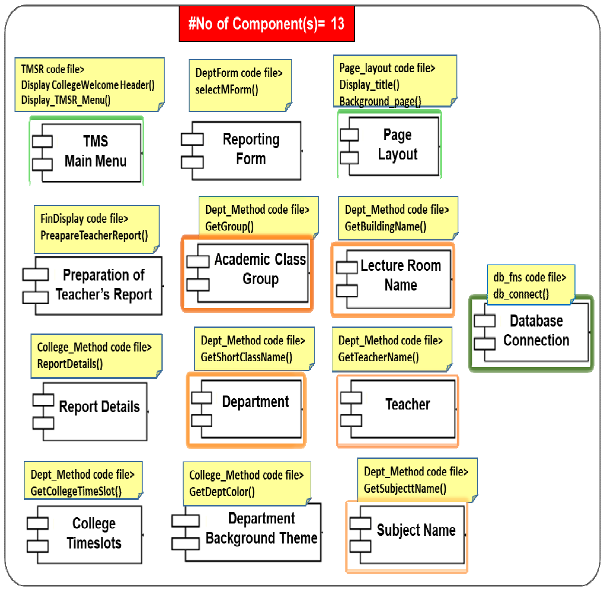
4.2) Representation and Visualization of architectural information:

This process includes two steps: The first step deals with mapping the extracted code elements into architectural components. The selected code elements in Figure 14 (webpages and functions) are mapped into *thirteen components* architecture. The second step is visualizing and representing particular architectural information using a web application layered architecture style.

The selection of the architecture type is based on Web Application Archetype which is applicable with the TMS system. The core of the Web application is the server-side logic which is visualized in a three-layer architecture.

Figure 15 shows the main components in each layer that are used to describe and represent "*TMS_Req2.20*" functionality as following:

- *Presentation layer* includes *three components* such as (TMS Main Menu, Reporting Form and Page Layout component). These components are responsible for managing the end-user interaction with TMS system.
- *Business layer* includes *nine components* which implement the core functionality of TMS system. The first *four components* such as (*Preparation of Teacher Report*, *College Timeslots*, *Report Detail* and *DeptBackground Theme* component). These Components are concerned with the retrieval, processing, transformation, and management of TMS's data; business rules and policies. The others *five components called* "business entities" which encapsulate the business logic and data necessary to present the real world elements within TMS system, such as (*Academic Class Group*, *Lecture Room*, *Teacher*, *Subject* and *Department*).
- *Data access layer* consists of the *database connection component* which provides access to the data hosted within TMS system



#No of Component(s)= 13
TMSR code file> Display CollegeWelcomeHeader() Display_TMSR_Menu()
TMS Main Menu
DeptForm code file> selectMForm()
Reporting Form
Page_layout code file> Display_title() Background_page()
Page Layout
FinDisplay code file> PreapareTeacherReport()
Preparation of Teacher's Report
Dept_Method code file> GetGroup()
Academic Class Group
Dept_Method code file> GetBuildingName()
Lecture Room Name
db_fns code file> db_connect()
Database Connection
College_Method code file> ReportDetails()
Report Details
Dept_Method code file> GetShortClassName()
Department
Dept_Method code file> GetTeacherName()
Teacher
Dept_Method code file> GetCollegeTimeSlot()
College Timeslots
College_Method code file> GetDeptColor()
Department Background Theme
Dept_Method code file> GetSubjecttName()
Subject Name

Fig.14 Mapping Extracted Code Elements into Components Architecture

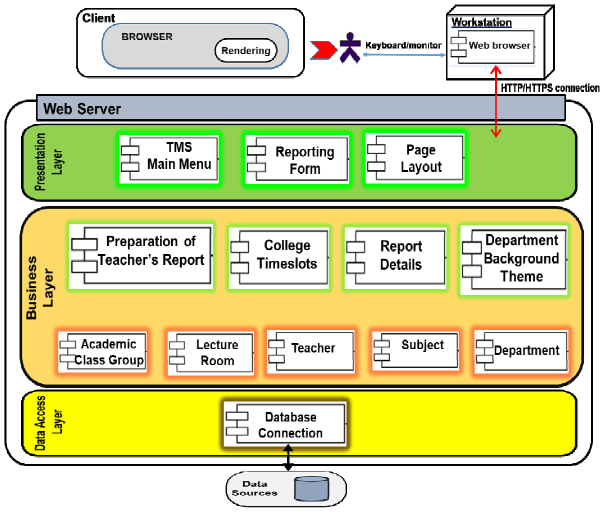


Fig. 15 Visualizing Particular Architectural Information using Layered Architecture Model

To summarized, the layered architecture model is used to visualize and represent the extraction of particular architectural information into a graphical model for stakeholders which helps to answer their architectural concerns about specific functionality of the TMS system. Moreover, this architectural model provides an abstract level of architectural representation for stakeholders which highlights which set of components are needed to execute specific functionality of the system. This is shown here as the functionality of the mechanism for managing the scheduling of Teachers lectures as shown in Figure 16.

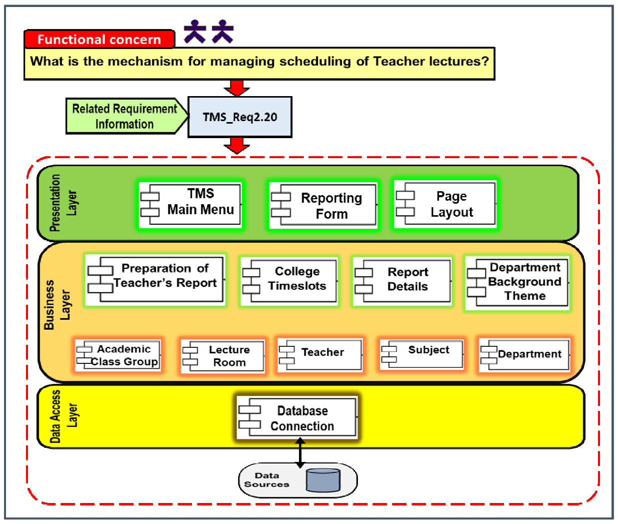


Fig.16 Representation of Particular Architectural Information Based on Stakeholder's Functional Concern

IV. COMPERSION WITH RELATED WORKS

RE has become one of the major engineering trends for software evolution. The core of RE consists of extracting information from the available software artifacts such as source code and translating it into abstract representations to be understandable by the stakeholders [2],[3],[5]. Accordingly; C.Stringfellow et al. discussed that reverse architecting is a specific type of reverse engineering, and stated that the RE process should consist of three phases starting with an extraction phase where we extract information from the source code and document it in documentation, and documented system history. The process also include an abstraction phase which abstracts the extracted information based on the objectives of RE activity, then elicits the extracted information into a manageable amount of information. And finally a presentation phase that represents the abstracted data in a way suitable for the stakeholders [23].

Software architecture consists of the description of components and their relationships and interactions, both statically and behaviourally as described in [6],[15],[23],[24]. Chikofsky et al. discussed that the RE process helps to generate the documentation to recover the design information of the system by analyzing the software to identify the components and the interrelationships between these components, and to create a representations of the software system [2].

Previous research made great strides to overcome the problem of documenting and recovering the software architecture to reflect the system's changes. Therefore, several approaches, methods, frameworks and RE methodologies have been proposed form different perspectives [4-6],[15],[22-37]. The most important of these proposed approaches were based on the concept of architectural knowledge [24],[31]. They promote the interactions between the stakeholders to improve the architecture of the software system.

Moreover; some of the recent approaches and techniques considered the perspective of getting the executable architecture from existing source code of software system as in [26],[34],[35]. These techniques considered every line of code for extracting the architecture of a target system. However, these extracted architecture were reflected every functionality exists in the original source code. For example; R.Arshad et al. proposed a RE model called (X-MAN) for extracting executable architecture in form of component model based on object oriented source code [34]. The executable architecture contains structural and behavioural aspects of software system in analyzed manner, and the extracted components can be used to support the re-usability of component and integrated them with other systems as described in [26],[34].

For further information; we presented a survey paper indicated in [4]. This survey paper reflects the current state of art in documenting and recovering software architectures using RE techniques. We highlighted and compared set of existing RE methods and approaches based on their findings and limitations. However, the main observation indicates that most of these existing methods and approaches are mainly focused on the developer viewpoint as the main stakeholder; and based to reflect the whole architecture of software system [4]. The recent approaches and methods discussed the need for alternative solutions to extend additional stakeholders. The solutions should focus to communicate with the stored architectural information by applying the scenario based documentation through stakeholders' scenarios and managing the architecture's documentation of software system. However; these issues should simplify and classify the architectural information based on identifying stakeholders' concerns and viewpoints about the target system, and visualize the architectural information in a proper level of abstractions based on these stakeholders' concerns.

In this paper we present a RE methodology for visualizing architectural information for multiple stakeholders and viewpoints based on applying the RE on specific parts of the source code. The process is driven by eliciting stakeholders' concerns on specific architectural viewpoints to obtain and visualize architectural information related these concerns.

The main idea of the methodology integrates the RE technology and the representation of software architectural information. The extraction process of RE methodology is driven by addressing the specific concern by stakeholder(s) for extracting only partial architectural information. Therefore, it's doesn't address RE of the whole architecture of a target system. Moreover; the representation process includes two key steps; mapping the extracted code elements into a component model; and visualizing the architectural information using the architecture styles. This visualized architectural information indicates the architecture for particular part of software system which support the understand-ability and maintainability process for legacy software system.

Respecting and comparing with some of the related works as summarized in Table 6; our main contributions are three fold: (1) The RE methodology is based on the IEEE 1471 standard for architectural description and supports concerns of stakeholder including end-user and maintainer; (2) RE methodology supports the visualization of a particular part of the target system by providing a visual model of the architectural representation which highlights the main components needed to execute specific functionality of the target system, and (3) The methodology uses architecture styles to organize the visual architecture information. We illustrate the methodology using a case study of a legacy web application system

As a result of these contributions, the visualization of a particular part of the target system highlights the main components needed to execute specific functionality which can be used to support the understand-ability and maintainability of the legacy software system (by putting the stakeholder in the maintenance loop; so that stakeholder can give feedback on the information related the target system).

V. Conclusion And Future Work

The main contributions drawn from the proposed RE Methodology are: firstly; a new RE Methodology follows IEEE 1471 standard of architectural description and support concerns of stakeholder including end-user and maintainer. Secondly; GUI prototype tool to support the steps of Methodology. It supports the visualization of a particular part of the target system by providing a visual model of the architectural representation which highlights the main components needed to execute specific functionality of the target system. Finally; the verification of the methodology using legacy web application system.

TABLE 6
SUMMARIZATION OF SOME RELATED APPROACHES AND METHODOLOGIES

Author (year)	**General Description**	**Documenting Architecture Whole/*Particular***	**Addressing stakeholder concern**	**Organizing Extracted information**
Kumar (2013)	RE methodology for understanding software artifacts.	Whole Architecture	*Developer concern*	UML models (state diagram and communication diagram).
Hugo et al. (2014)	Framework for understanding the contents of legacy systems using model driven RE.	Whole Architecture	*Developer concern*	By three layers and the components of each layer are based on the nature of legacy system technologies.
Che et al. (2011)	An approach for collecting the architectural design decisions (ADDs)	Whole Architecture	*Developer and architect concern*	Using triple view model framework (TVM) which includes three different views for describing the notation of ADDs.
Che et al. (2012)	An approach for managing the documentation and evolution of the architectural design decisions	Whole Architecture	*Developer and architect concern*	TVM framework for specifying its views through end-user scenario(s).
Che (2013)	Methodology for documenting and evolving the architectural design decisions	Whole Architecture	*Developer and architect concern*	UML metamodel for TVM framework, each view of TVM specified by classes and a set of attributes for describing ADDs information.
Riva et al. (2002)	Approach for generating the architectural documentation	Whole Architecture	*Developer concern*	Using XML notation for representing the architectural documentation.
C.Meiru (2013)	Approach for documenting and evolving architectural design decisions	Whole Architecture	*Developer and architect concern*	Using textual format for representing the architectural design decisions.
Panas et al. (2013)	RE approach for unified recovery architecture	Whole Architecture	*Developer and architect concern*	Unified recovery model for documenting the architecture of software.
Arshad et al. (2017)	RE model for extracting the architecture of object oriented source code.	Whole/ Particular Architecture	*Developer concern*	Component model for representing the architecture.
Starke et al. (2017)	Arc24 Template for documentation of software and system architecture	Whole Architecture	*Developer and architect concern*	Textual document includes several sections: underlying business goals, essential features and functional requirements for the system, quality goals, the relevant stakeholders and their expectations.
Maras et al. (2009)	PHPModeler tool for legacy PHP Web applications	Whole Architecture	*Developer concern*	Static UML diagrams (such as: dependency models for representing resources of the current page, its functions and dependencies).
Razavizadeh et al. (2009)	Framework for extracting the architectural views from object-oriented source code.	Whole Architecture	*Developer concern*	Conceptual model for representing the architectures' viewpoints.

Further information; the extraction of architectural representation helps stakeholders especially (maintainer, end-user, architect, tester and developer) for obtaining the as built architecture from its implemented source code elements, and supporting the understand-ability and maintainability phase for the target system.

For example; the architectural representation can be used by the maintainer to support the understand-ability for particular part of the system; by tracing the related requirement information through its implemented code elements and highlighted which components were needed to represent specific functionality of the target system as described in Figure 16.

Moreover; in case of improving or re-engineering the legacy software system into new technology such as (object oriented system or cloud based application system); the architectural representation helps the maintainer to identify which set of components that implement the core functionality of legacy system, and encapsulate the relevant business logic, or either to decide how to manage and migrate the executable components into cloud based environment.

Additionality, the extracted architectural information can be used by the end-user to support the understand-ability for particular part of the system by providing a proper level of architectural diagram that highlighted which components are needed to describe specific functionality. Actually, this is very important by putting the end-user in the maintenance loop so that end-user can give feedback on the information related the target system, or either to determine and decide in case of re-engineering specific functionality of legacy software system through adding new features for the target system.

The main recommendations for the Future work are highlighted as follow: there is a need to extend RE methodology to support additional architectural viewpoint beside the "Functional viewpoint" based on a given classification of viewpoints catalog (such as: the information viewpoint, the deployment viewpoint, and the operational viewpoint). The development of automated tool is needed to support the whole phases of RE methodology, and apply RE methodology in different application domains such as: the robotics systems and smart object systems to support the understand-ability and maintainability process for particular parts of these systems.

VI. ACKNOWLEDGMENT

This research work was funded in part by Qatar National Research Fund (QNRF) under the National Priorities Research Program (NPRP) Grant No.: 7 - 662 - 2 - 247

REFERENCES

[1] [1] M. Garg and M. K. Jindal, "Reverse Engineering Roadmap to Effective Software Design," International Journal of Recent Trands in Engineering, vol. 1, no. 2, May. 2009.
[2] [2] E. J. Chikofsky and H. C. James, "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, 13-17 Jan.1990.

[3] [3] L. H. Rosenberg and E. H. Lawrence, "Software re-engineering," Software Assurance Technology Center, 1996. [Online]. Available: http://www.scribd.com/doc/168304435/Software-Re-Engineering1.

[4] [4] H. Alamin M. and H. H Ammar, "Reverse Engineering for Documenting Software Architectures, a Literature Review," International Journal of Computer Applications Technology and Research, vol. 3, no. 12, pp. 785 - 790, Dec 2014.

[5] [5] M. Harman, W. B. Langdon, and W. Weimer, "Genetic Programming for Reverse Engineering," in Working Conference on Reverse Engineering (WCRE'13), Koblenz, Germany, 2013.

[6] [6] P. Clements, F. Bachmann, L. Bass and D. Ga, "Prologue: Software Architectures and Documentation," 2010. [Online]. [Accessed 26 April 2014].

[7] [7] R.Hilliard, D. Emery, M. Maier, "All About IEEE Std 1471," 2007. [Online]. Available: http://www.csee.wvu.edu/~ammar/CU/swarch/lectureslides/slidesstandards/all-about-ieee-1471.pdf.

[8] [8] Institute of Electrical and Electronics Engineers, "IEEE Recommended Practice for Architectural Description of Software Intensive Systems," 2000. [Online]. Available: http://cabibbo.dia.uniroma3.it/ids/altrui/ieee1471.pdf. [Accessed 9 July 2014].

[9] [9] P. Kruchten, "Architectural Blueprints: The “4+1” View Model of Software Architecture," IEEE Software, vol. 6, no. 12, p. 42–50, 1995.

[10] [10] N. Rozanski and E. Woods, Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives, 2nd ed., Addison Wesley, 2005

[11] [11] N. Rozanski and E. Woods, "Applying viewpoints and views to software architecture," 2011. [Online]. Available: http://www.viewpointsandperspectives.info/vpandp /wpcontent/themes/secondedition/doc/VPandV_WhitePaper.pdf. [Accessed 14 June 2015].

[12] [12] M. Nicholas, "A survey of Software Architecture Viewpoint Models," in In Proceedings of 6th Australasian Workshop on Software and System Architectures, 2005.

[13] [13] E. Woods, "Experiences Using Viewpoints for Information Systems Architecture: An Industrial Experience Report".

[14] [14] K. Henk and H. V. Vliet, "A method for defining IEEE Std 1471 viewpoints," Journal of Systems and Software, ELSEVIER, vol. 79, no. 1, pp. 120-131, January 2006.

[15] [15] C. Riva and Y. Yang, "Generation of architectural documentation using XML," IEEE Computer Society Press, vol. 9, no. In Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE02), pp. 161-169, 2002.

[16] [16] P. Clements , "Comparing the SEI’s Views and Beyond Approach for Documenting Software Architectures with ANSI-IEEE 1471-2000," Software Engineering Institute, Carnegie Mellon University, July 2005.

[17] [17] N. Rozanski and E. Woods, "Viewpoints and Perspectives Reference Card," [Online]. Available: http://www.viewpoints-and-perspectives.info/home/viewpoints/functional-viewpoint/. [Accessed 10 November 2015].

[18] [18] N. Rozanski and E. Woods, "Viewpoints and Concerns," 2011. [Online]. Available: http://www.viewpoints-andperspectives.info/home/viewpoints. [Accessed 10 November 2015].

[19] [19] MySQL 5.7 Reference Manual Document, "MySQL 5.7 Reference Manual Document/Full-Text Search Functions/Natural Language Full-Text Searches," MySQL, [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/fulltext-natural-language.html. [Accessed 25 March 2017 at 8:00AM].

[20] [20] MySQL Reference Manual, "MySQL Reference Manual/Important Algorithms and Structures/10.7-Full-Text Search," [Online]. Available: https://dev.mysql.com/doc/internals/en/full-text-search.html. [Accessed 1 April 2017 at 05:30PM].

[21] [21] Doxygen Tool website and Doxygen Documentation. [Online]. Available: http://www.doxygen.org/download.html

[22] [22] MI C R O S O F T ® Architecture guide, "MI C R O S O F T ® Application Architecture Guide(patterns & practices Developer Center), Application ArcheTypes, Chapter 20: Choosing an Application Type," 2009. [Online]. Available: https://msdn.microsoft.com/en-us/library/ee658104.aspx.

[23] [23] C. Stringfellow, C. D. Amory and D. Potnur, "Comparison of software architecture reverse engineering methods," In Proceedings of Information and Software Technology, vol. 7, no. 48, pp. 484-497, July 2006.

[24] [24] C. Meiru, "An Approach to Documenting and Evolving Architectural Design Decisions," in International Conference on Software Engineering (ICSE'13), San Francisco, CA, USA, 2013.

[25] [25] R. K. Len Bass and P. Celements, Software Architecture in Practice, 2nd ed., Addison Wesley Professional, 2003.

[26] [26] K. Lau and C. M. Tran, "X-man: An mde tool for Component based System Development," in Software Engineering and Advanced Applications (SEAA), and EUROMICRO Conference, IEEE, 2012.

[27] [27] T. Panas , W. Lowe and U. Aßmann , "Towards the Unified Recovery Architecture for Reverse Engineering," [Online]. Available: https://ai2-s2-pdfs.s3.amazonaws.com/b8e1/c9bd8cf3360b82de68e8049b281a1e2f4a25.pdf. [Accessed 30 October 2017].

[28] [28] G. C. Penta and D. Massimiliano , "Frontiers of Reverse Engineering: a Conceptual Model," pp. 38-47, 2008.

[29] [29] A. Razavizadeh , H. Verjus, S. Cˆımpan and S. Ducasse, "Multiple Viewpoints Architecture Extraction," in IEEE, 2009.

[30] [30] S. Demeyer, S. Ducasse and O. Nierstrasz, "Object Oriented Reengineering Patterns," Switzerland, Square Bracket Associates, 2008, p. 338.

[31] [31] M. Shahin , P. Liang and M. Khayyambashi, "Architectural Design Decision: Existing models and tools," in European Conference on Software Architecture, 2009.

[32] [32] G. Starke and P. Hruschka, "Arc24 Template for documentation of software and system architecture," 3 May 2017. [Online]. Available: http://www.arc24.de.

[33] [33] G. Liang and L. Yu, "Quality Driven Re-engineering Framework," Blekinge Institute of Technology, Sweden, December, 2013.

[34] [34] R. Arshad and K. K. Lau , "Extracting Executable Architecture From Legacy Code Using Static Reverse Engineering," in International Conference on Software Engineering Advances(ICSEA 2017), 2017.

[35] [35] J. Maras, M. Štula and I. Crnkovic, "PHPModeler- a Web Model Extractor," in IEEE/ACM International Conference on Automated Software Engineering, (Nov2009), IEEE Computer Society, 2009.

[36] [36] W. Kim, S. Chung and B. Endicott Popovsky, "Software Architecture Model Driven Reverse Engineering Approach to Open Source Software Development," in The 3rd annual conference on Research in information technology, Atlanta, Georgia, USA, October 15–18, 2014, ACM.

[37] [37] A. Razavizadeh, H. Verjus, S. Cimpan and S. Ducasse, "Multiple Viewpoints Architecture Extraction," 2009 IEEE/IFIP WICSA/ECSA, pp. 329-332, 2009, IEEE..