Chapter 1: Introduction

1.1 Introduction

Re-documentation is part of software engineering, it recovers the understanding of the software and records it, thus making future program understanding easier (E.Chikofsky and J.Cross,1990).

A key goal of re-documentation is to provide easier ways to visualize relationships among program components so that recognize and follow paths clearly. One of the ways to implement the re-documentation process is using the reverse engineering approach .

Reverse Engineering (RE) captures the information from the existing source code, it is the opposite of forward engineering. Also it is the process of analyzing a subject system to identify the system's components and their interrelationships.(Tung Doan , 2008).

Generating quality documentation through re-documentation process is important for program comprehension and software evolutions (N.Sugumaran1, S.Ibrahim2 , 2011) .

This study proposes **a framework for system re-documentation based on reverse engineering approach**. It also **proposes a model for evaluating the quality** of the generated documentation.

1.2 Statement of the Problem

Documentation is a missing item in the maintenance legacy software systems . As these old systems evolve, there is a need for the corresponding documentation and an understanding of the original design so that modifications to the software can be made properly.

Due to this lack of up-to-date documentation, some times maintainers must often work from the source code to the exclusion of any other source of information. For example, a study reports that from 40% to 60% of the maintenance activity is spent on studying the software to understand it and how the planned modification may be implemented .

The key point solution to the above problems is the software re-documentation. Reverse engineering provides a better understanding of an existing system , by focusing on the process of reverse engineering to extract related information from source code then generating new documents .

1.3 Motivation and Objectives

Legacy software systems are generally poorly documented. This makes it difficult to understand and maintain such systems. Re-documenting them could be a great help to keep them “alive”.

Poor and un-useful document is the **main problem** we search to find solution for it . Are the **Reverse Engineering** is the good approach to re-document a legacy software ? , are the document we

generated from RE tool was high quality , How could we measure it ? .

Because high quality and useful documentation is one of the key factors in producing quality and consistent software .

Based on the observations above, our research objectives are summarized as follows:

- Define re-documentation process to explain the role of reverse engineering in it .
- Built a Document Quality Model , to terminate the quality document attributes then measure the quality of document generated from the case study .

1.4 Research steps

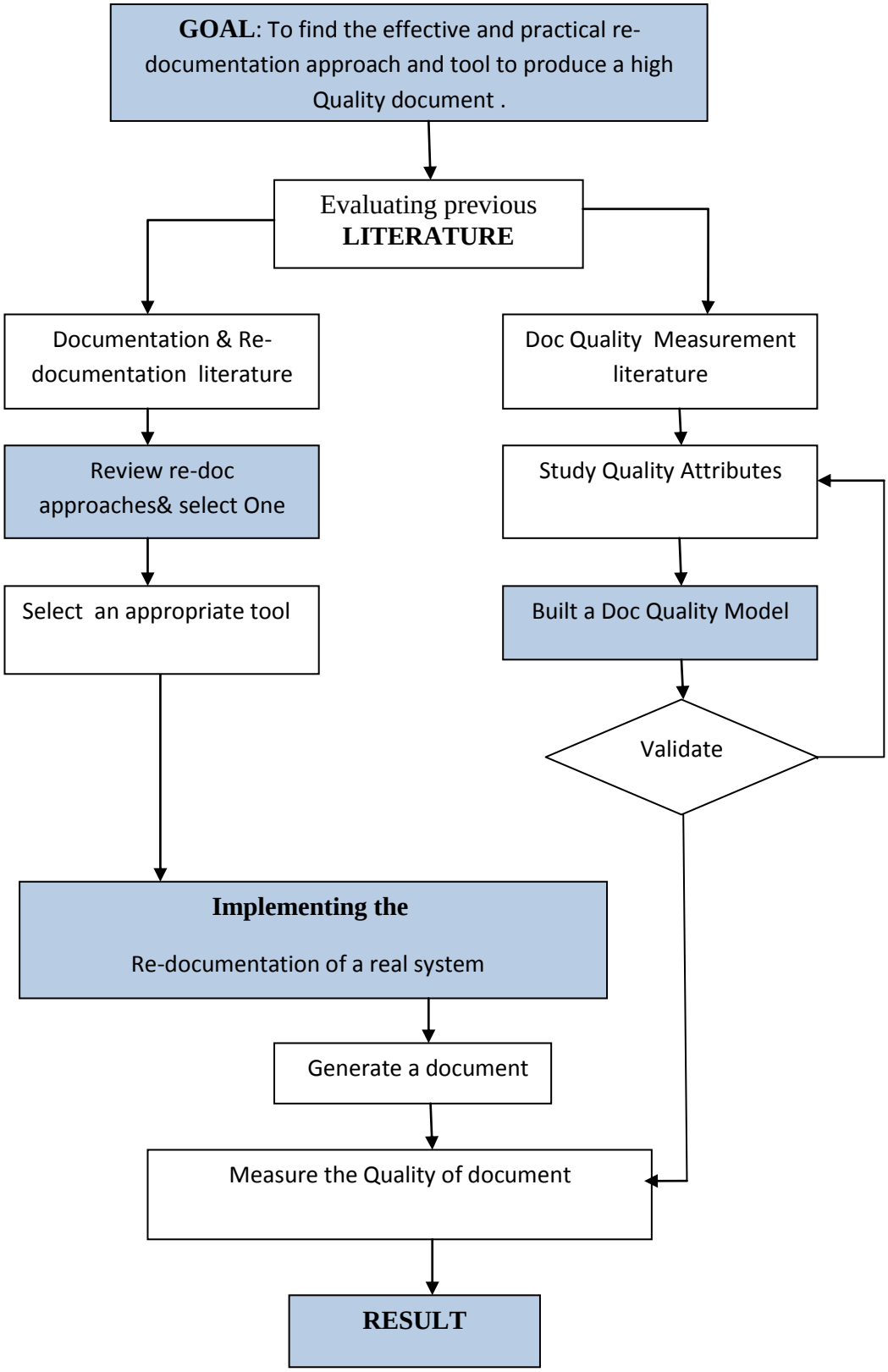


Figure 1.1: Research steps

1.5 Research Organization

The thesis is structured as follows :

Chapter 2 : Provide background and literature review of the thesis

Chapter 3 : Contains the research methodology .

Chapter 4 : Shows The Result and discussion.

Finally, **Chapter 5** : Concludes the thesis .

Chapter2 :literature review

2.1 Overview

The purpose of this survey is to identify the historical background and current understanding of the subject, and to establish the state-of-the-art. The content of this chapter defines the knowledge base upon which work presented in this thesis .

Various research related to software re-documentation, has been carried out at universities and institutes around the world. As a natural starting point in the development of this thesis, I started looking for published academic and research papers with topics and aims similar to this thesis.

The literature focuses on the following areas of study :

- **Software Re-documentation**: Literature the past research that helps define the re-documentation in a software environment as well as providing an analysis of re-documentation process , approaches of re-documentation.
- **The Reverse engineering** as one of the most common approaches of re-documentation.
- **Quality of Software documentation** :to understand the reasons behind re-documentation the aspects of quality document should be define .

2.2 Software Re-documentation

Re-documentation is in general the process of analyzing a software system to represent it with a meaningful alternate view intended for a human audience. Usually, this task is supported by tools that generate views and diagrams really helpful for software engineers during maintenance and quality assurance(M. Torchiano et al , 2009) .

Researchers and practitioners also have looked at the uses of software documentation as just to compile a few notes , but documentation can also be used for learning , testing and working with software system, also solving problems and keeps software-quality at high levels.

Re-documentation addresses that problem by recovering knowledge about the system and making it explicit in documentation .

2.2.1 Re-documentation definition

The intent of re-documentation is to recover documentation about the subject system .Software re-documentation is part of software engineering. It only recovers the understanding of the software and records it, thus making future program understanding easier . (N.Sugumaranet al , 2009) .

" Software re-documentation is one of the approaches used as an aid for program understanding , to support the maintenance and evolution ".(SugumaranNallusamy , Suhaimi Ibrahim, 2011). Another definition explained by Tilley as follows. *"Program redocumentation is one approach to aiding system understanding in order to support*

maintenance and evolution. It is the process of retroactively creating program documentation for existing software systems. It relies on technologies such as reverse engineering to create additional information about the subject system. The new information is used by the engineers to help make informed decisions regarding potential changes to the application".(Tilley, S. 2008) .

From these definitions it is clearly stated that re-documentation is to recreate a documentation of existing system from the available resource. However, Software improvement by updating documentation it is a (re-documentation).

2.2.2 Goals of the software Re-documentation

The purpose of re-documentation to make sure the software teams understand the legacy system (Sugumaranet al , 2009). The main goals of the software re-documentation process are : It is **firstly** used to create alternative views of the system so as to enhance the understanding .**Secondly**, it is also used to improve the current documentation. Ideally, such documentation should have been produced during the development of the system and updated as the system changes. This, unfortunately, is not usually the case. **Thirdly**, it also generates documentation for a newly modified program (Sugumaran Nallusamy , Suhaimi Ibrahim, 2011).

2.2.3 Re-documentation Categorization

Software re-documentation can be categorized into Textual and Graphics.

Textual documents include textual information and description about the software system that could be in different formats such as Word, PDF, HTML and XML files .**Graphical** documentation includes charts and graphs that rely on a variety of software visualization techniques to make complicated information easier for the designers, developers and testers to understand (e.g., MS Visio, UML diagrams).(Joris Van et al, 2010) .

Each of this can be classified in many ways. On of the way is to classify by their presenting mode , as shown in Figure 2.1.

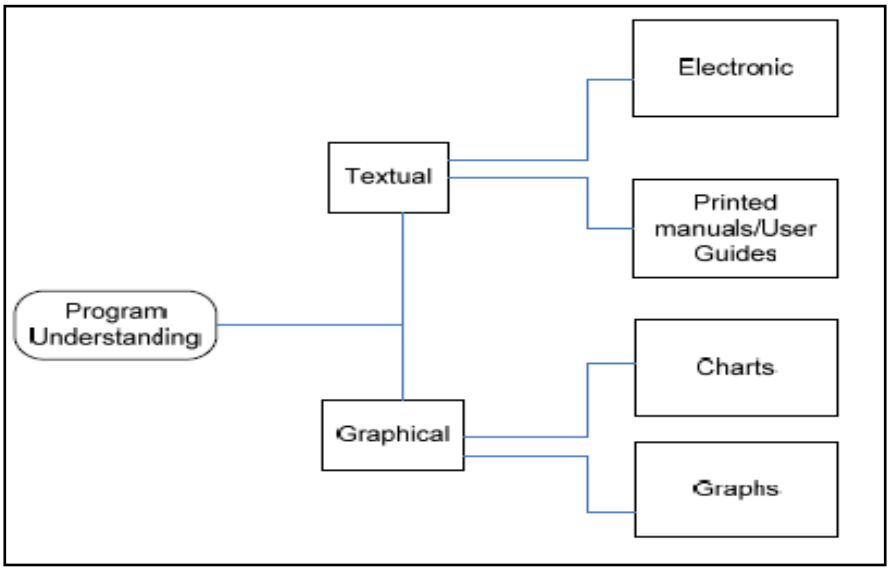


Figure 2.1: A Classification of software re-documentation

Both textual and graphics become more reliable and make complicated information easier for the developer to understand by representing into diagram .

2.2.4 Re-documentation Process

From a high-level perspective, two steps are necessary to re-document a system.

Firstly, one needs to **extract facts about the system**. That can be done starting from a static representation of the system (e.g., the source code), from already available documentation (text documents, spreadsheets, . . .etc,) or from the people working with the system.

Secondly, these **facts need to be combined and transformed into the correct documentation format** (e.g., UML diagrams or hyper documents**) .**(Joris Van et al, 2010)

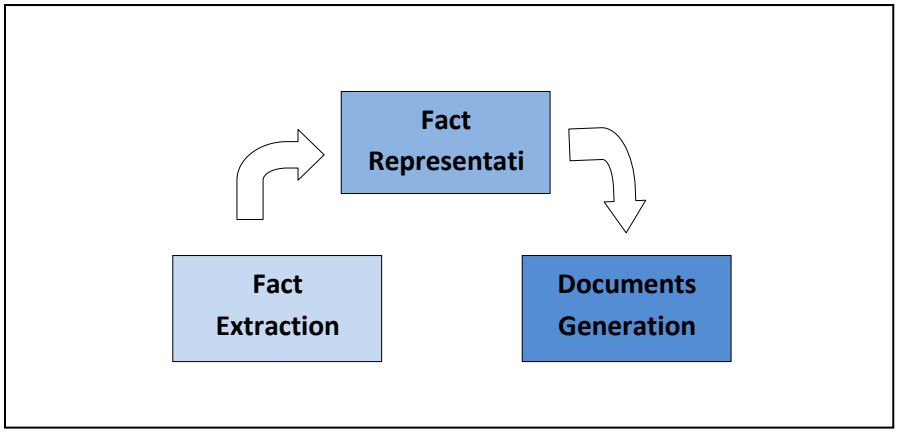


Figure 2.2: Overview of the re-documentation process

2.2.5 Re-documentation Approaches

There are many re-documentation solutions in industry and research. Most of the solutions can be categorized into few approaches.

We have identified some significant approaches and tools which can contribute to further development of the quality documentation from the re-documentation process :

1. **XML Based Approach:** XML based approach is one of the common re-documentation approaches . It contains structured information that extracts the content and the meaning of the documentation **.**

   By using XML the technical writer or software engineer can create their own format such as <CONSTRANT>, <TASK>،<FILE>, <VARIABLE> and<FUNCTION> . The nature of XML shows that the information in hierarchical help to understand the program more easily. It also validates the data captured from the program (Jochen et al. 2001)

2. **Model Oriented Re-documentation (MOR):** The first step in Model Oriented Re-documentation approach is to transform the legacy system into formal models. These formal models are written using a formal language and transform into TSs(Technological Spaces). Are Generated TSs are stored in repository and produced documentation in a uniform way.

3. **Incremental Re-documentation:** One of the common issues in maintaining the system is to record the changes requested by customer or user to occur in the source code. the Incremental Re-documentation approach to rebuild the documentation incrementally after the changes are done by the programmer **.**

4. **The Ontology Based Approach :**It produce a schema from the legacy system to describe the context of the software system . The schema should able to capture the artifacts from the latest version of the software system by establishing a reverse engineering environment.

5. **Reverse Engineering :** Reverse engineering captures the design information from the existing source code .The purpose of re-documentation to make sure the software teams understand the legacy system. Sometimes, the output of reverse engineering is thought to be the same as re-documentation.

2.3 Reverse Engineering (RE)

The most common approaches of re-documentation is software reverse engineering , it is the concerned with the analysis .

Reverse engineering as the inverse procedure of forward engineering , a forward software engineering process goes from certain specification or target architecture toward building the system .

Therefore , the main difference between reverse and forward engineering is that the reversing process goes from low abstract level to higher , figure 2.3

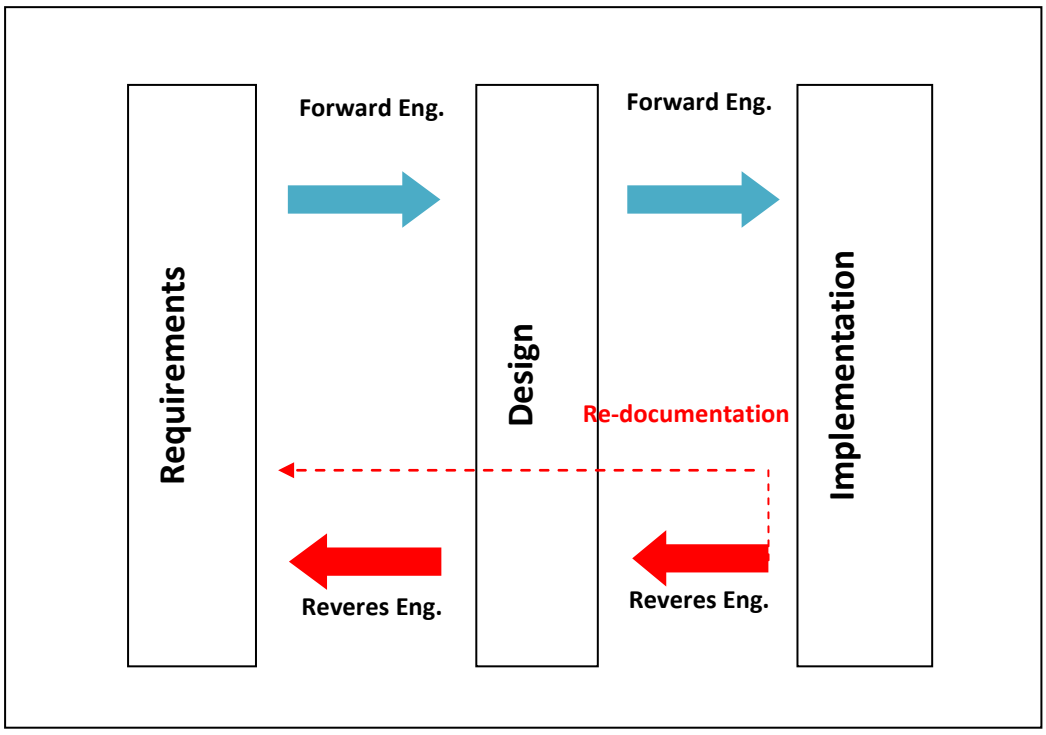


Figure 2.3: Difference between Forward and Reverse engineering

2.3.1 RE Definition: The IEEE Standard for Software Maintenance defines reverse engineering as :"*the process of extracting software system information (including documentation) from source code*. "(IEEE Std 1219-1993 , (1998)

In the context of software engineering, as defined by Chikofsky and Cross , reverse engineering is : *" the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction ."*(E. Chikofsky and J. Cross,1990)

2.3.2 RE Tasks :A past investigation detected five major tasks that an RE tool should support . These tasks are, in increasing abstraction level order: program analysis, plan recognition, concept assignment, re-documentation, and architecture recovery (see also Figure .2.4) . Program analysis is the basic task that any RE .

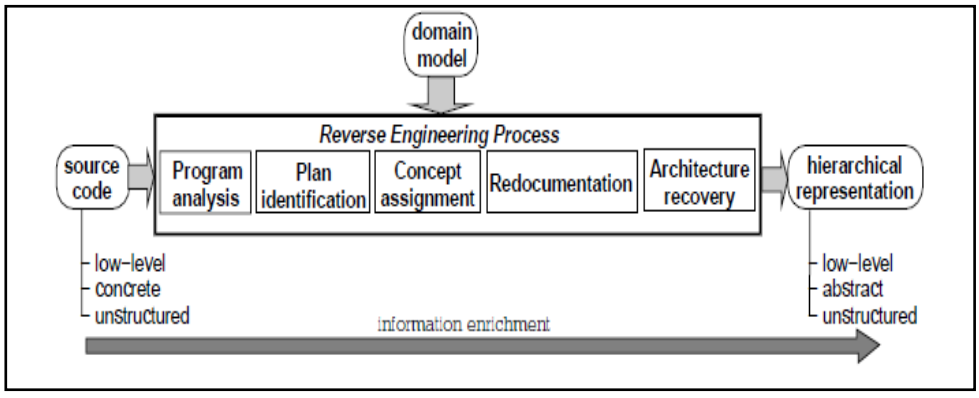


Figure 2.4 : Reverse engineering Tasks

Software systems that are targets for reverse engineering, such as legacy applications, are often large, with hundreds of thousands or even millions of lines of code . As a result, it is highly desirable to automate reverse engineering activities.

The Rigi environment provides such tool support , the reverse engineering of (industrial) systems with Rigi has generated valuable experiences. These experiences have also shaped Rigi's methodology of how to reverse engineer or re-document (legacy) systems.(IzzulHidayatNaisan and Suhaimi Ibrahim .2010) .

2.4 Quality of Documentation

The quality of a software product is a significant driver for its success. However, the majority of the applied quality assurance methods mainly focus on the executable source code. Quality reviews of the software documentation are often omitted.

Software documents such as requirements specifications, design documents, or test plans represent essential parts of a software product. Therefore, the quality of such documents influences the overall quality of a software product considerably.

That means , Documentation is a key component in software quality and improving the documentation process will have considerable impact on improving the quality of software.

Documents describe the product at all levels of development including the finished product. therefore, **documents need to be up-to date, complete, consistent and usable**.(Kipyegen, William P. K.2013) . **Capri** defines a **successful documentation** as one that makes information easily accessible, provides a limited number of user entry points, helps new users learn quickly, simplifies the product and helps cut support costs.

Poor documentation is the cause of many errors and reduces efficiency in every phase of a software product's development and use .

Quality of software is not 'just an IT problem'; it is a business problem if the software affects the business. The goals for all Software Engineering research are improvements in productivity and quality.

2.4.1 Quality definition

A number of non-equivalent definitions of ‘quality’ exist, indicating that different understandings of the concept of ‘quality’ exist ; therefore numerous approaches to quality management have been proposed which aim to consider quality in a generic context.

According to Kitchenham [KIT96] ‘Quality’ means different things to different people; it is highly context dependent.

As there is no universally accepted definition of quality there can be no single, simple measure of software quality that is acceptable to everyone.

However, defining quality in a measurable way makes it easier for others to understand a given viewpoint .

2.4.2 Measuring the Quality

To understand and measure quality, researchers have built models of how quality characteristics relate to one another . Software product quality model is to make clear and direct links between high-level quality attributes and explicit product characteristics at all levels.

To understand and measure quality, many models of quality and quality characteristics have been introduced.

Jim McCall et al. (1977) introduced the model with the following attributes Correctness, Reliability, Efficiency, Integrity and Usability.

Barry W. Boehm used As-is Utility (Reliability, Efficiency ,Portability, Human Engineering) Maintainability (Testability, Understandability , Modifiability) and Portability to determine the quality.

The ISO 9001 and **The IEEE Std 1061-1998** used sets of factors such as Functionality ,Reliability, Efficiency, Maintainability, Portability and Usability to distinctively assess the quality.

The Hybrid Methodology for Software Documentation Quality provide a documentation quality **meta-model** presented by **(GolaraGarousi, 2012)**, that modeled the quality of content using Content Quality, which has several attributes as its subclass: Accessibility, Accuracy, Author-related, Completeness, Consistency, Correctness, Information organization, Format, Readability, Similarity, Spelling and grammar, Traceability, Trustworthiness, Up-to-dateness as shown in figure 2.5 :

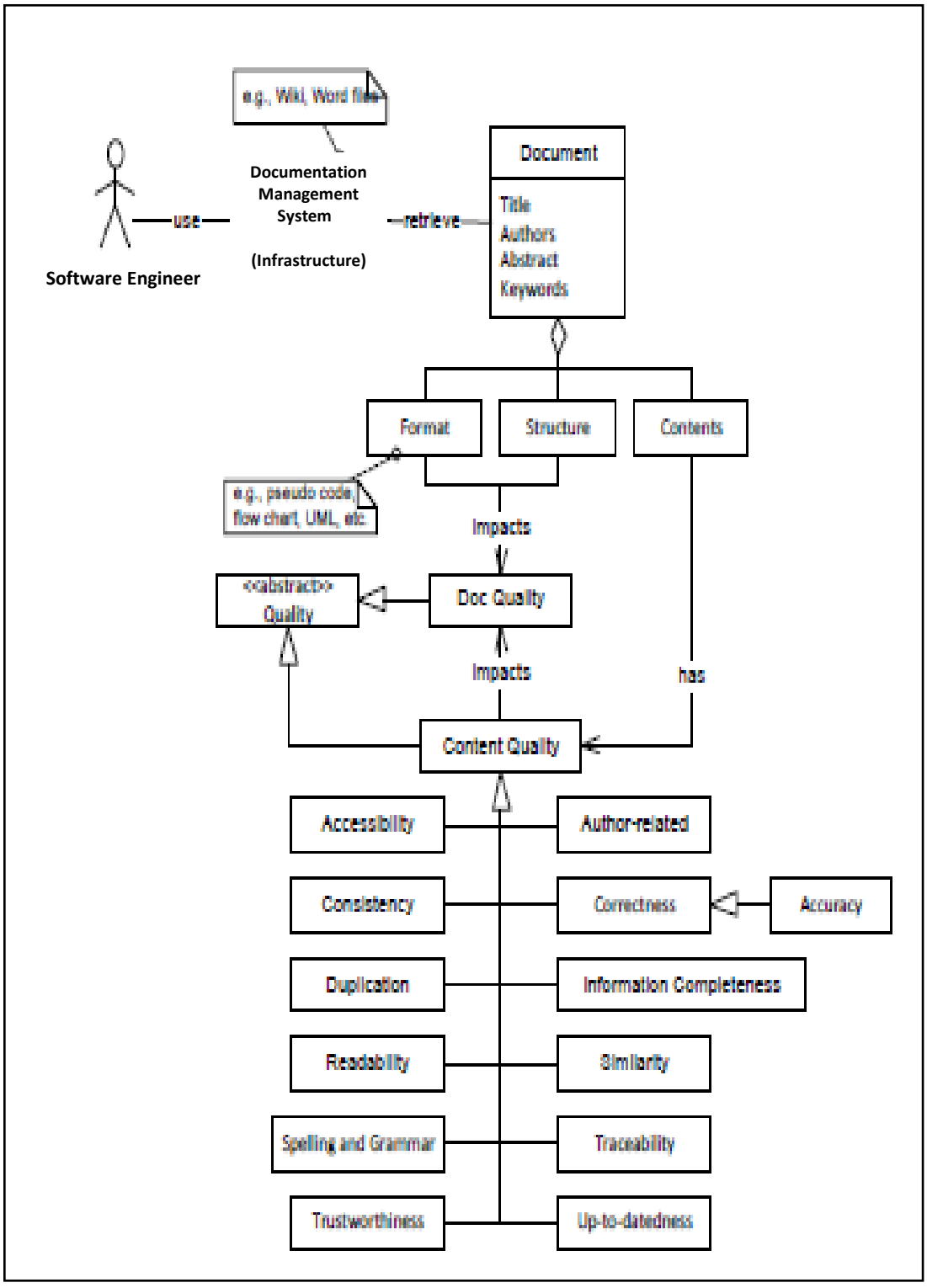


Figure 2.5 : Documentation quality meta-model overview ,

(GolaraGarousi, 2012)

The **key performance indicators** (KPIs) framework , it was also created by focusing on quality attributes of the of the document . (BISHARE S.A , et al., 2013). Shown in Table 2.1 below .

Table 2.1 : KPIs Framework

KPI	Quality Attributes
Structure	Understandable , well-presented, well-documented ,concise representation , consistency , interpretability .
Contextual	Value –added, appropriate amount of data , completeness .
Accuracy	Accuracy, believability , objectivity .
Accessibility	Accessibility ,easily traced, user friendly, ease to retrieval .

The models presented herein are focused around a set of attributed/metrics used to distinctively assess quality by making quality a quantifiable concept.

A common approach to formulating a software product quality model is to first identify a small set of high-level quality attributes and then, in a top-down fashion, decompose these attributes into sets of subordinate attributes.

Chapter 3 : Research Methodology

3.1 Introduction

A case study : Re-documented SQL/DB System **: ShM** using **Rigi Reverse Engineering tool .**

To validate the usefulness of the reverse engineering approach in re-document , a target ShM software system was examined . Using Rigi tool to understand the real-word system to re-documented it . In this chapter we present the result which was carried out by implementing the following steps .

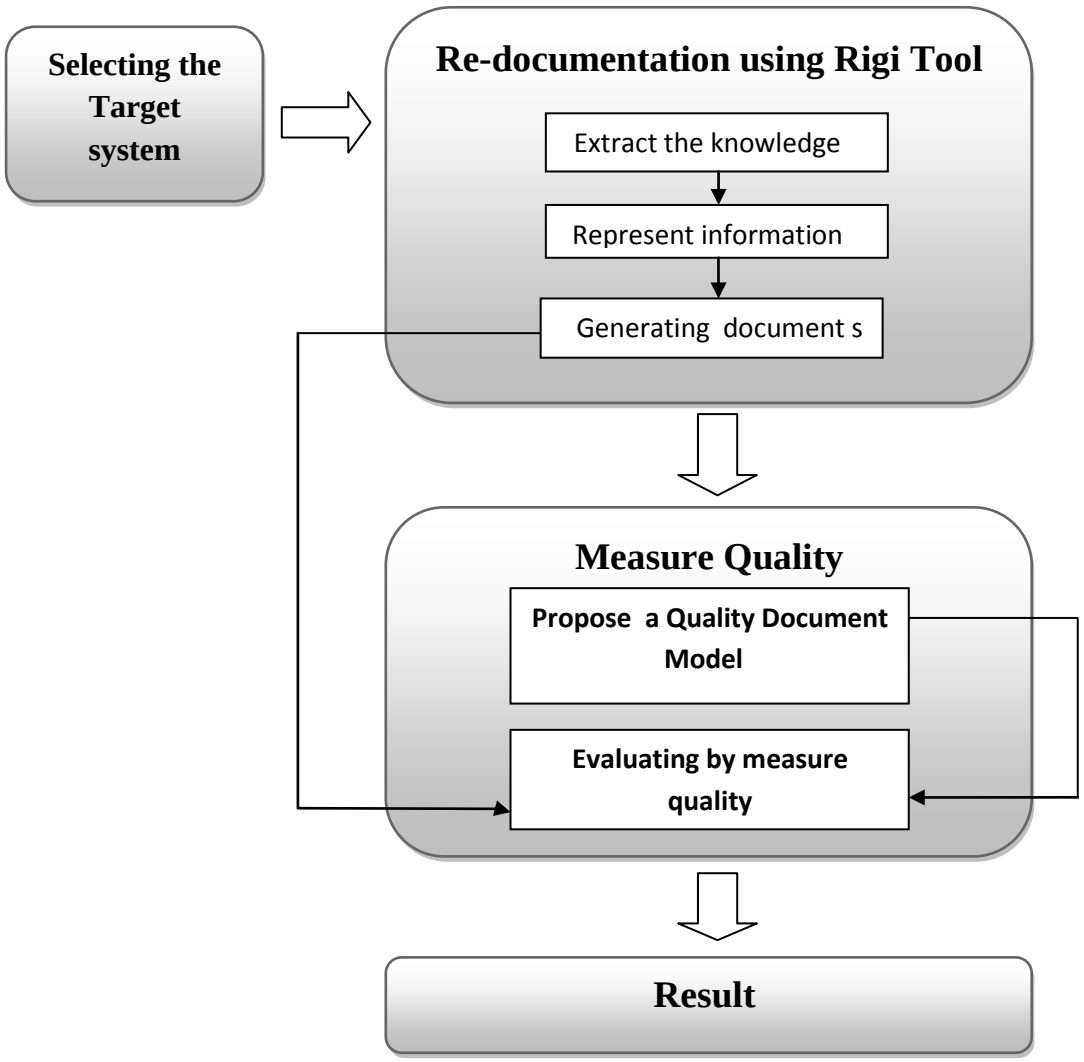


Figure 3.1 : Research Methodology steps.

The first step in research was select the target system (undocumented or poor documented system). Then using Rigi tool to **implement the reverse engineering process**, the extracted information from this step is visualized as a directed graph or as text information for the selected subsystem in report.

Then we proposed or built a **Document Quality Model– DQM** by selected aspects and attributes to evaluate and measure the quality of the document generated from the previous step.

We summarized the two practical steps as follows:

Step 1 : Re-documented the selected system , using Rigi tool ,

Step 2 :Measure Quality of the document generated .

3.2 The Target SQL/DB system : ShM

The selected target system **ShM** (Shopping Management System) is Service-Oriented system , that provides services for purchasing items such as books or clothes , based on large data base . Details shown in appendix B .

3.3 Overview to Rigi Reverse Engineering Tool

Rigi allows the visualization of software in the form of graphs and supports a reverse engineering methodology called structural re-documentation .

Rigi is used for program understanding , it is an interactive, visual tool designed to help developers better understand and re-document their software . The describe of Rigi's main components and functionalities look at appendix A .

3.4 Re-documentation using Rigi tool

Graph editor *rigiedit* whose user interface is based on windows , when you run it , you initially see the Rigi *Workbench* window and a *root* window shown in figure 3.3

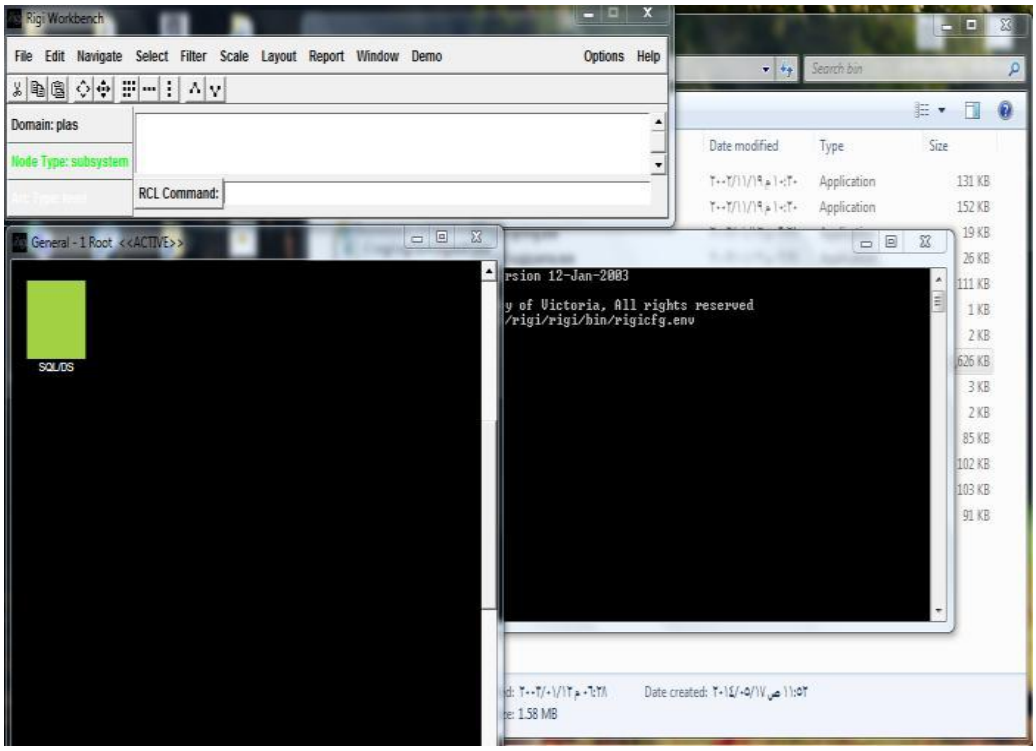


Figure 3.2: The main windows of *Rigiedit* .

3.4.1 Read the source code

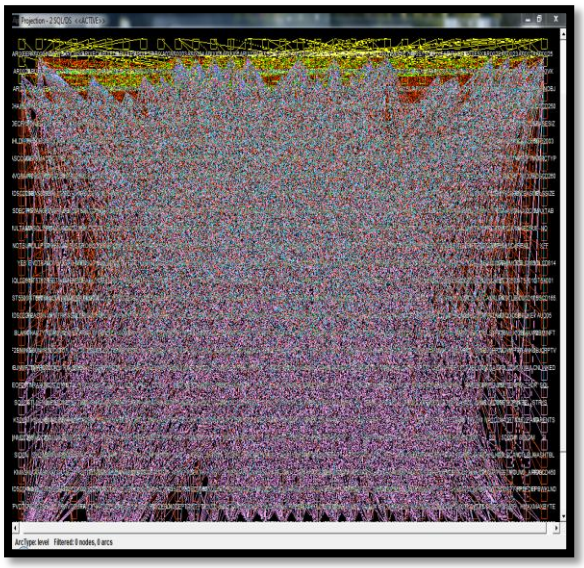
The source code represented in the input file ,Rigi includes parsers to read the source code of the subject system , then it is able to view information stored in RSF files .

3.4.2 Load initial Graph , the Root window

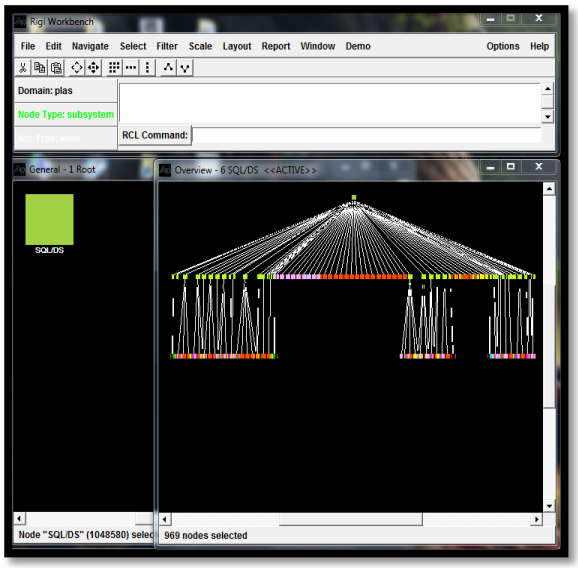
From the *File* menu Choose *Load Graph* that presenting a view of the current directory contents , select the file called ***rsf***, and click *OK*. The initial window titled *Root* is used to display the parent(s) of the subsystem hierarchy (*SQL/DS node*).

3.4.3 Visualization , The Graphical Representation

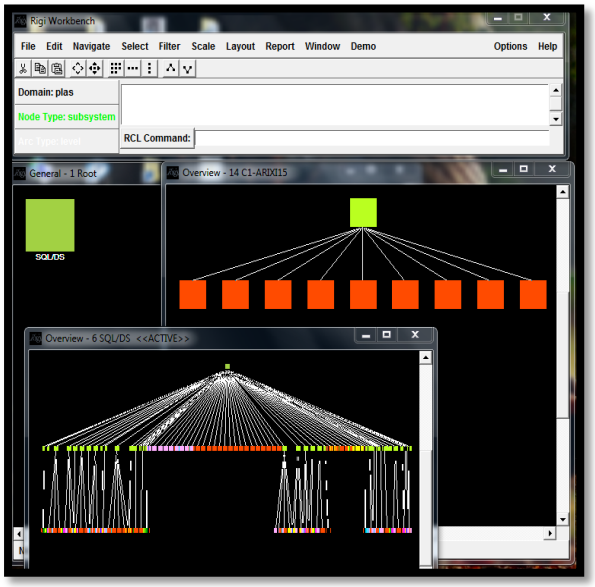
To represent and visualize extracted information as directed graph should implement the four step below , as shown in figure 3.3 in (A , B , C , D) .



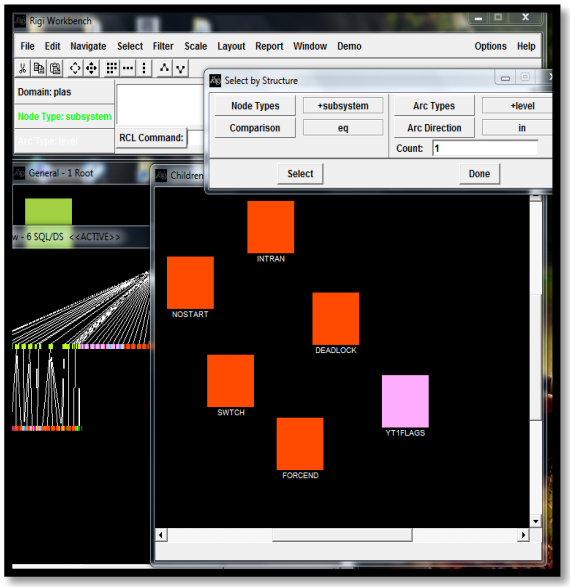
A) Rigi initial graph for whole system



B) The Tree-like structure



C) Identify a subsystem



D) Nodes in Object level

Figure 3.3 : The Graphical Representation

A) To load an **initial graph into Rigi** from a file generated by the source code . Choose *Projection* from the *Navigate* menu .

The **reverse engineering** is often presented, a complex software systems, with many more nodes and arcs, the resulting visual clutter can be confusing.

We see a graph that contains all the artifacts in the system. The arcs in the graph describe dependences among the artifacts .

B) To Manage the complexity of the graph for large information spaces, we use **presenting the tree-like structure** .

A new Overview window appears, to perform an overview of the subsystem hierarchy descending from the SQL/DA node : Choose *Overview* from the *Navigate* menu .

That presents a vertical "slice" of the hierarchy.

C) Identify clusters of related nodes and collapse them into nodes that **represent subsystems** . By visually inspecting the subsystem graph, you get a high-level summary of the major components of the program.

The simplest traversal technique is to open a node and traverse down in the hierarchy. *Double-left-click* on the selected subsystem node .

D) A Child window typically presents the structure of a single node, by choose *Children* from the *Navigate* menu .

At the **object level** the name of the node appears .This portrays part of a level in the hierarchy, in a kind of horizontal "slice".

3.4.4 The Textual Representation

To **view information** on the immediate subsystem node Choose *View Information* from the node *menu* right-click on the node as it is presented within a window , in figure 3.4 :

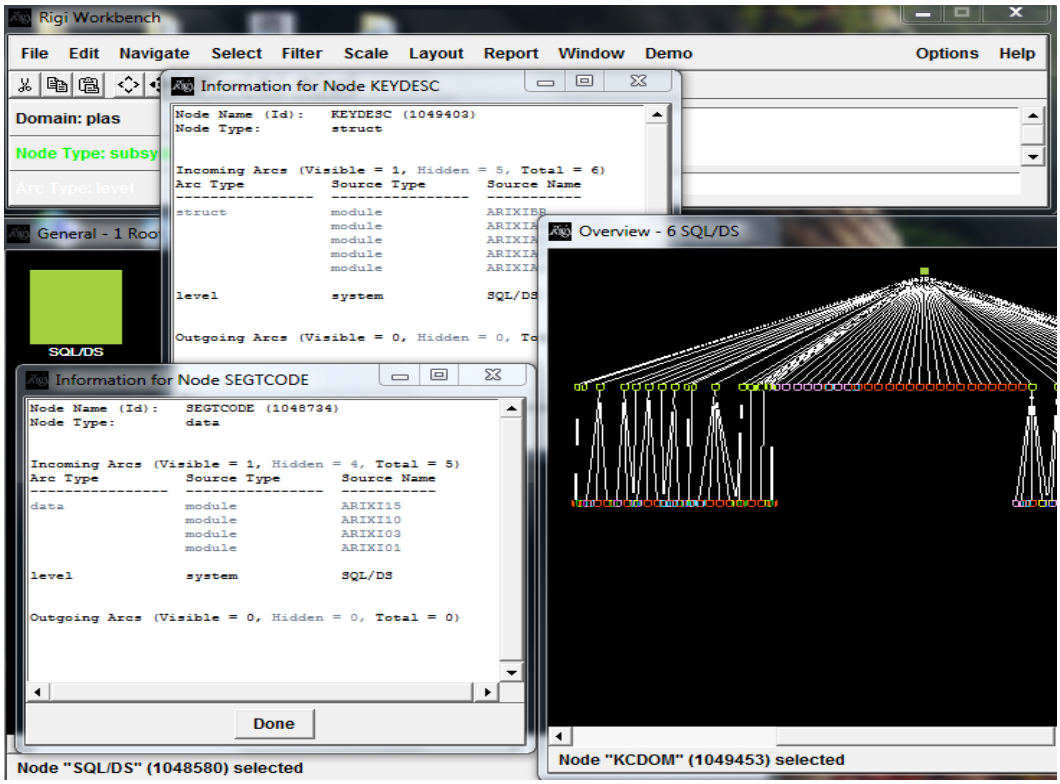


Figure 3.4 : view information from selected node .

A textual Information window appears . This information includes the node's:

- internal node ID ,
- node type,
- incoming and outgoing arcs by arc type,
- and neighboring nodes along these arcs .

Produce an Exact Interface Report for the whole system , Choose *Exact Interface* from the *Report* menu .

3.5 Generating Documents :

To generate documentation, it is not possible to do everything automatically. Some parts still require human interpretation to complete the work .We chose a UML notation to generate a standard graphical documentation .

UML class diagrams are widely used for modeling the static structure of software system in both forward and reverse engineering .

In this research ,Rigi has been used for static reverse engineering .the extract static information is viewed as a directed graphs. The static dependency graph contains approximately the same information as a class diagram .

In Rigi, classes and interfaces have their own node types , methods ,constructors and variables given inside a class in UML class diagram .

The Table 3.1 enumerates the main UML class diagram constructs and the constructs that can be used in Rigi for expressing the meaning of the UML class diagram .The correspondence is characterized as replacing if such a Rigi construct exists **.**

Table 3.1 : Class diagram construct VsRigi graph construct

A UML class diagram construct	A Rigi static dependency graph construct	Correspondence
Class	Class (node type)	Replacing
Interface	Interface (node type)	Replacing
Method	Method (node type)	Replacing
Variable	Variable (node type)	Replacing
Generalization	Inherit (arc type)	Replacing
Association	(arc type)	Replacing

For capturing high-level information about the model-view , we made a static abstraction a ShM class diagrams shown in figure 3.5 (a-b) below :

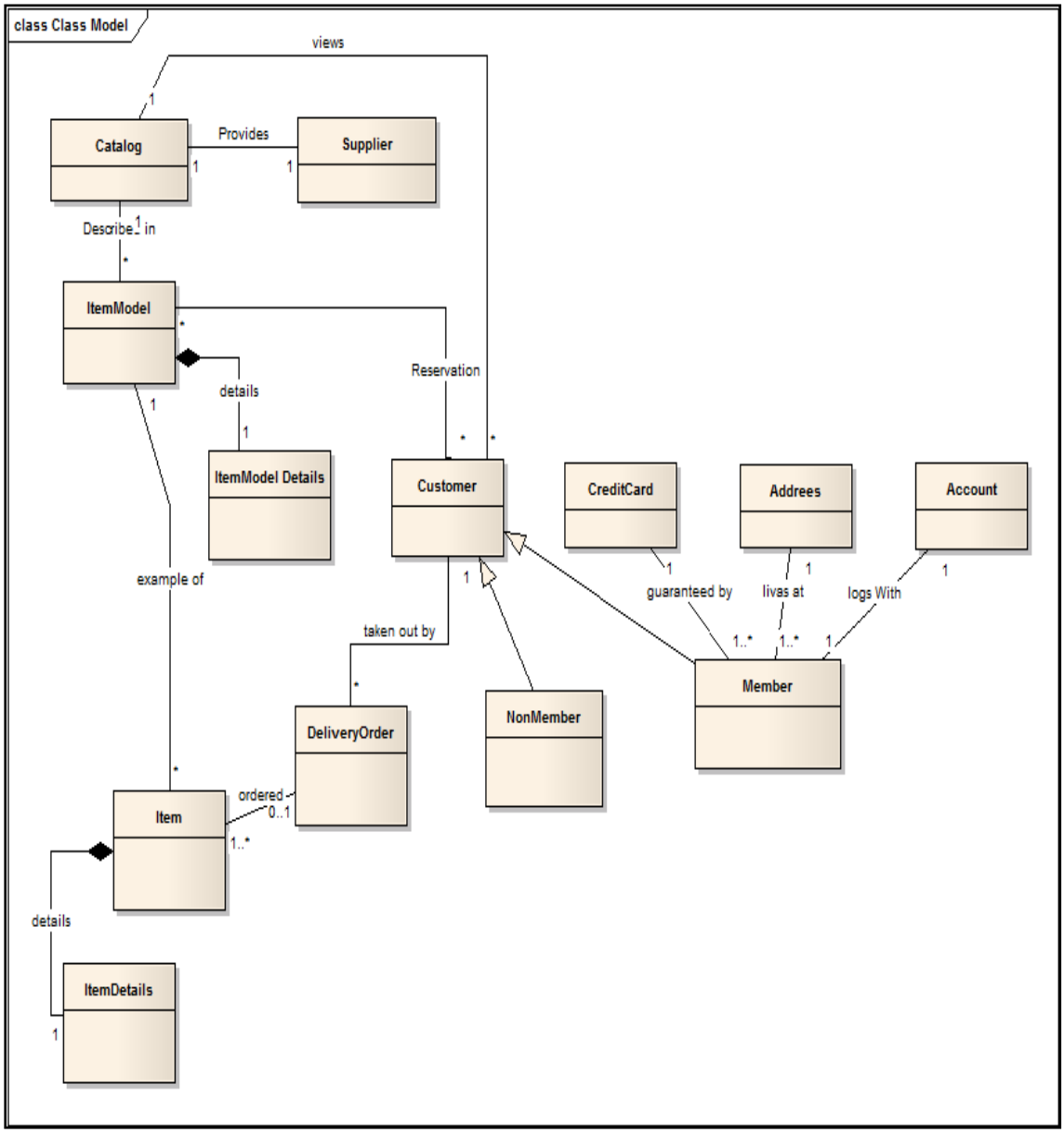


Figure 3.5-a :ShM class diagram

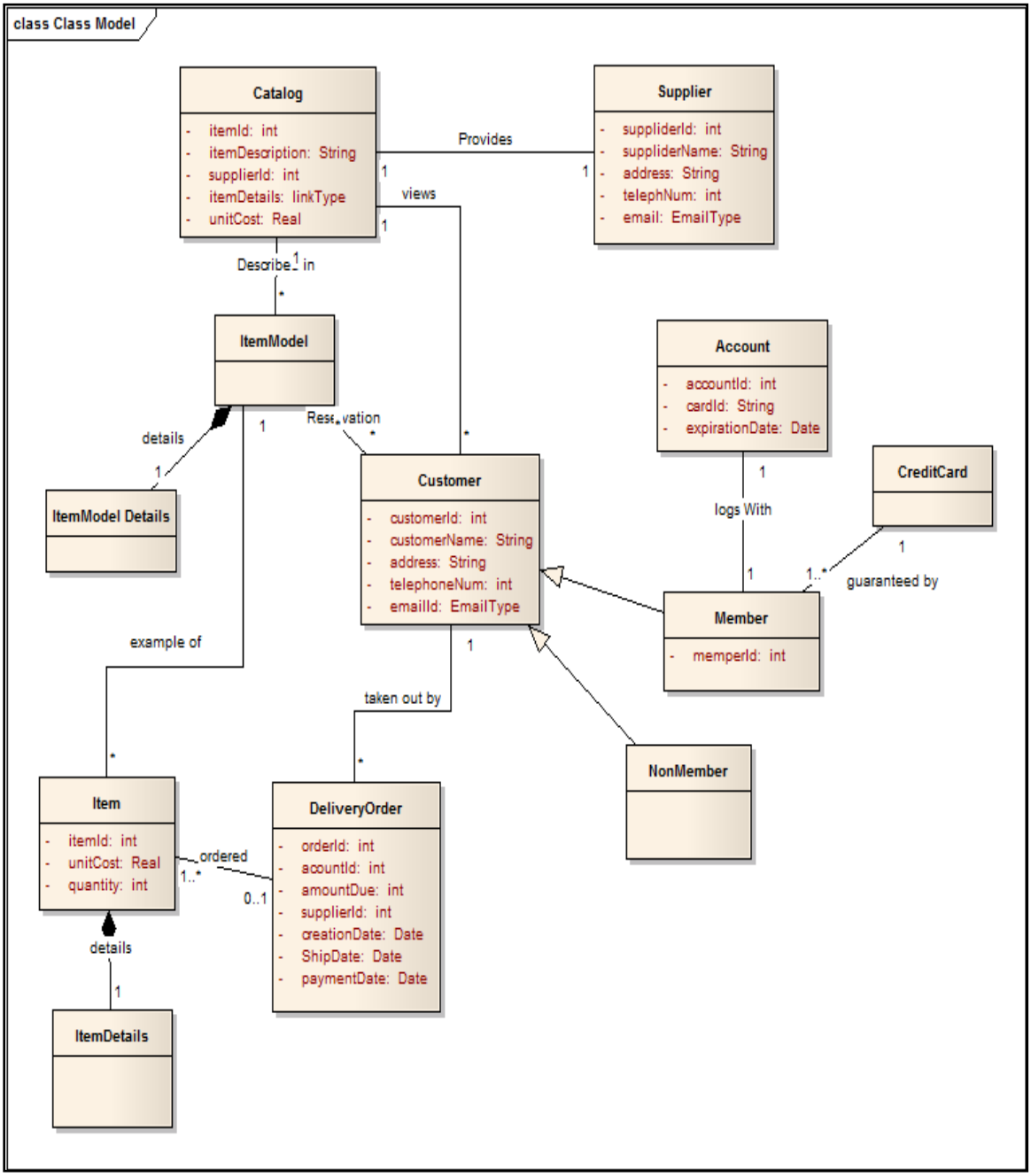


Figure 3.5-b :ShM class diagram with details

The static entity class model shows the entity classes and the relationships among these classes.

And the Component ports and interfaces for services in figure 3.6 .

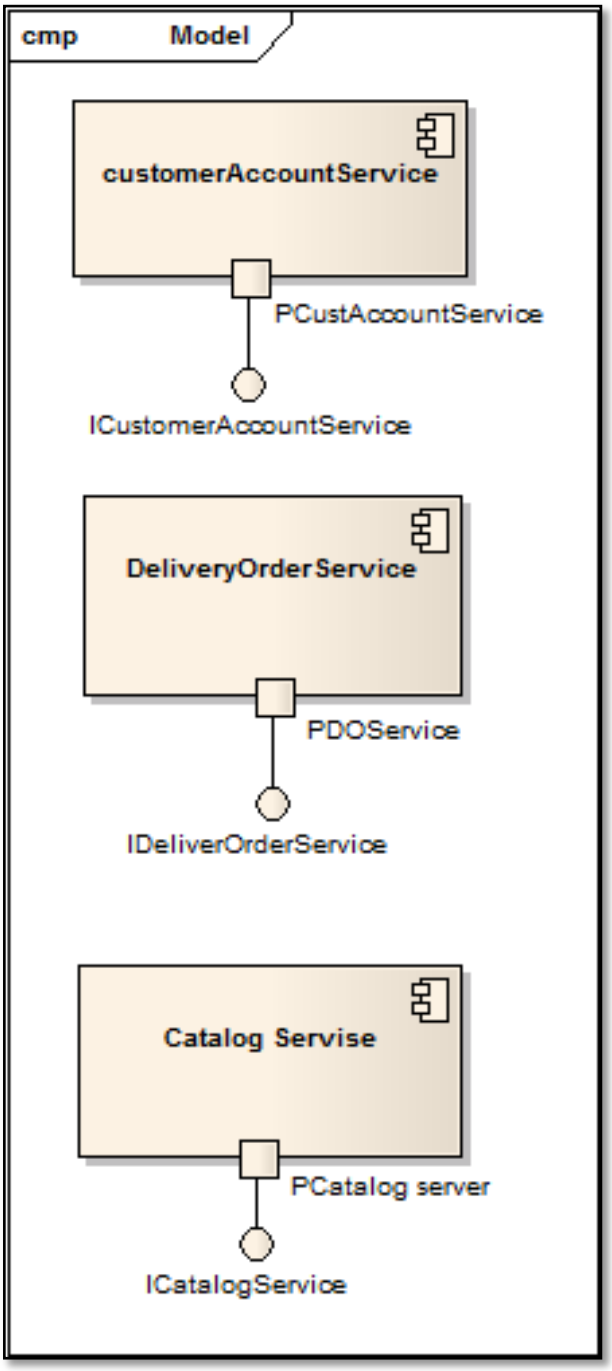


Figure 3.6 :Component ports and interfaces for services .

3.6 Proposed Document Quality Model

A number of quality models for software processes havebeen introduced, each of which is intended to encompass the totality of quality factors and issues. These models may be used to develop , measure or guide improvement of quality .

We applied an iterative process to derive quality attributes of the documentation as well , we ensured not to introduce any self-invented attributes other than those proposed by the authors of the primary studies. To classify quality-related attributes we constructed a unified model , as shown in Figure 3.7 :

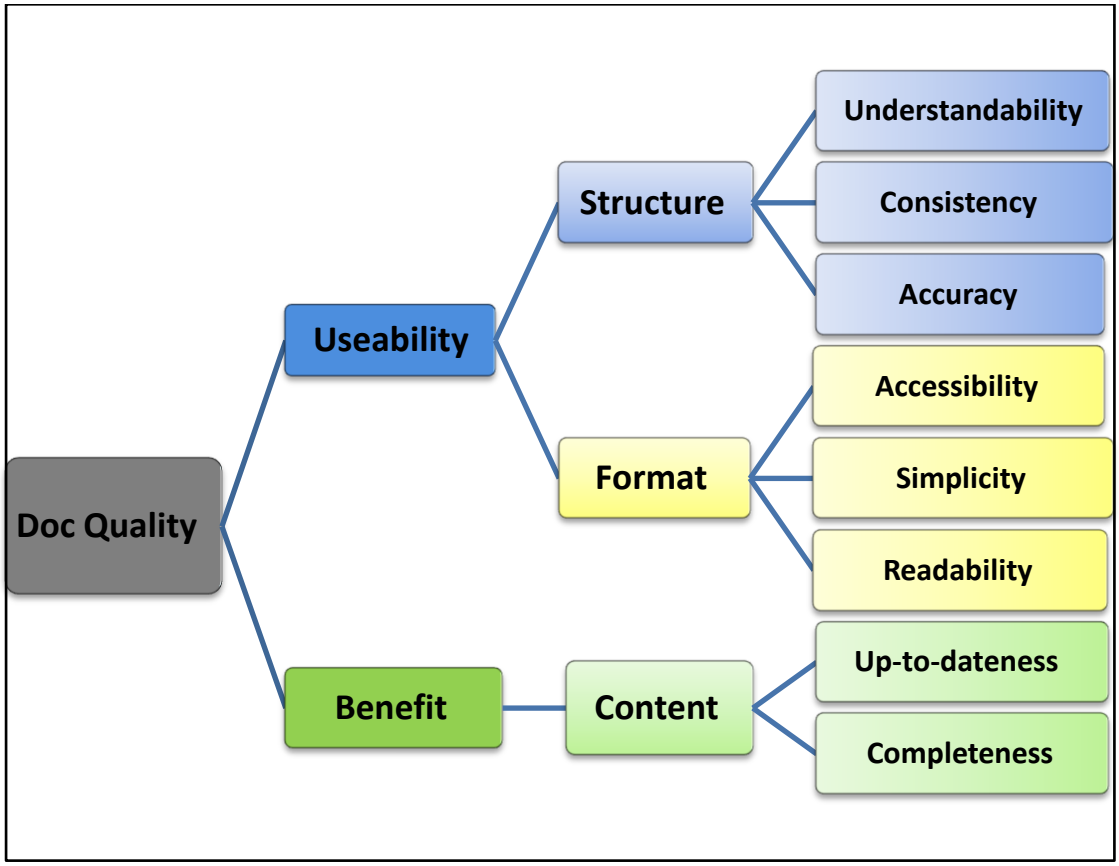


Figure 3.7 :Document Quality Model

The Document Quality Models DQM contain two main criteria **Usability** and **Benefit** . The quality criterions are depend on one or more aspects described by set of quality attributes.

The benefit of quality models is that they are simpler to use in proving the quality of document we have in re-documented process by Rigi tool.

Chapter 4 :Results And Discussion

4.1 Results

This research focuses on reverse engineering process to generate documentation from source code . The graphical tool (Rigi) we used allow manipulations of view , and give support for building high level model of target software to facilitate program comprehension . We chose UML notation to generate documents .

To Prove the Re-documentation through reverse engineering is very important and effective, empirical evaluation procedures are developed to measure quality of document was generated in the case study .

Using Document Quality Models DQM shown in previous chapter measuring main criteria **Usability** and **Benefit**.

Table 4.1 : Documents Evolution Result

Quality Attributes		**Documents Evaluate**
Usability	Understandability	√
	Consistency	√
	Accuracy	√
	Accessibility	√
	Simplicity	√
	Readability	√
Benefit	Up-to-dateness	√
	Completeness	√

4.2 Discussion

Several tasks were set for the case study. In what follows we discuss how well these tasks were achieved .

As we see in Table 4.1 ,**Usability criteria** measure by set of attributes : understandability ,consistency , accessibility … etc . All that depend on documents format choosing . The UML notation we choose it has accepted as standard for visualizing , understanding and documenting software systems ,it provides several diagram types support all lifecycle stage of forward engineering process . The same diagram types used for reverse engineering purposes as well.

UML brought forth a unified standard modeling notation that IT professionals had been wanting for years . So UML document format have all usability aspect we need .

The DQM measured **Benefit criteria** by content of documents, it should be up-to-date and complete . As shown in result Table 4.1 , the documents we generated in the case study is up-to-date it depend on the latest version of the software system , also it complete and accurate because it is derived from the actual source cod .That explain effective of the reverse engineering technique in software re-documentation . This is the aim we wont to achieve it .

Most of the researches published in this field , they used others re-document approaches with in different tools and achieved a several results shown different quality measurement of document .

However , as we seen in literature review most of the approaches and tools are developed for reverse engineering which are generally compared to the re-documentation process .

Chapter 5: Conclusion and Recommendations

5.1 Conclusion

Legacy software systems have a different approach to software re-documentation than has traditionally been used , one of them is reverse engineering .

Documentations made manually by developers in some cases are inconsistent. Some change requests, updates, or bugs fixing somehow are not included in the documentation as the software evolves. Developers tend to be focusing on source code rather than the documentation. Consequently, code is the most reliable source to be referred as the system representation .

Generating the documentation directly from the source code makes the result consistent with the code at all times. Therefore, reverse engineering is very important and very effectively to understand large software systems then re-documented .

This research discusses a reverse engineering process implemented by Rigi tool to generate a standard software documentation.

5.2 Recommendations and Future work

Recognizing abstractions in real-world systems is as crucial as designing adequate abstractions from scratch, especially for obsolete system which was written 10 to 25 years ago. Reverse engineering is very significant to support software maintenance in order to maintain existing system .

However, Some parts still require human interaction to complete all information in the generated documentation. To do everything automatically will be part of the future work .

Appendix A

Overview to Rigi Reverse Engineering Tool

Rigi allows the visualization of software in the form of graphs and supports a reverse engineering methodology called structural re-documentation .

Rigi is used for program understanding , it is an interactive, visual tool designed to help developers better understand and re-document their software. It includes **parsers** to read the **source code** of the subject software and produce a graph .It helps to understand legacy software systems where the existing documentation may be missing or lacking.

In this section we describe Rigi's main components and functionalities, and assess its impact on re-documentation using reverse engineering in research and practice .

1. The Rigi. Environment

Rigi environment provides tool support automate reverse engineering activities. It is composed into three main entities that provide functionalities for :

- fact extraction,
- information representation, storage, (repository),
- and interactive graphical manipulation (editor) .

Rigi architecture shown in figure 1A exposes the main functionalities: **Extraction** of facts from software systems, a **Repository** to represent and store facts , and Analyses and **Visualization** of facts .
The facts that are stored in RSF Rigi Standard Format adhere to a certain data model (or schema). A RSF data model is explicitly defined with a simple specification language .

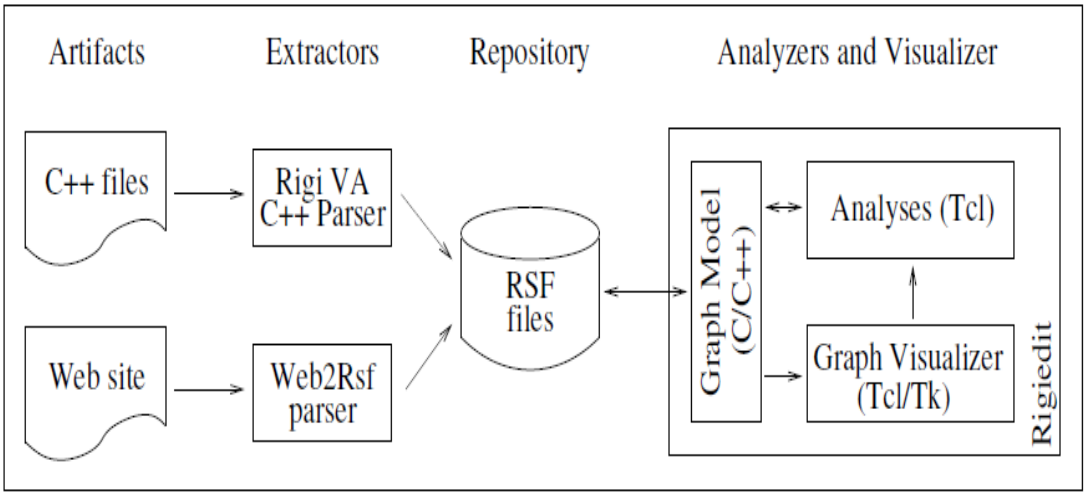


Figure 1A: Rigi's conceptual architecture

2. Reasons for choosing Rigitool

In this research , we had several reasons for choosing Rigi : **First** , Rigi scales up. **Second** ,Rigi support building high-level views of software by constructing hierarchical structures .

Third ,Rigi provides a large and extensible set of slicing mechanisms . they help the user to focus on a desired aspect of the software. **Fourth**, rigi is easy to customize and extend ,hence providing a good environment for the experiment by making the system end-user programmable RCL-Rigi Scripting language . Finally ,Rigi was flexibility and scalability **.**

Appendix B

The Target SQL/DB system : ShM

The selected target system **ShM** (Shopping Management System)is Service-Oriented system , that provides services for purchasing items such as books or clothes , based on large data base .

In the Shopping System, customers can request to purchase one or more items from the supplier.

The customer provides personal details, such as address and credit card information. This information is stored in a customer account. If the credit card is valid, then a delivery order is created and sent to the supplier. The supplier checks it, confirms the order.

In the following the main screens of system :

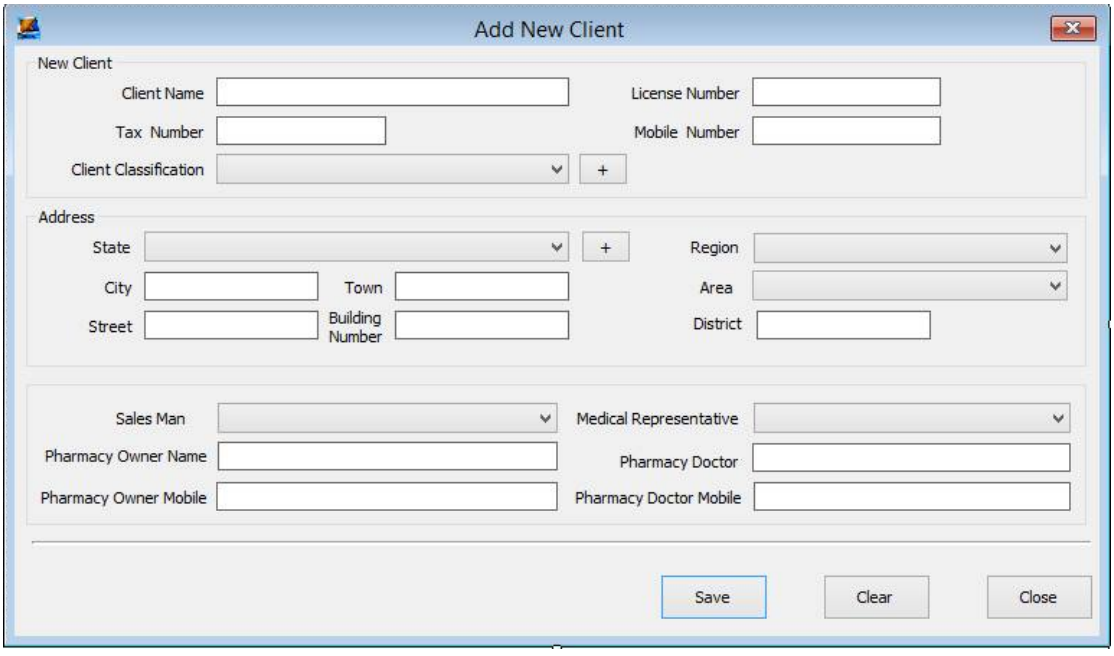


Figure 1B: Add New Client

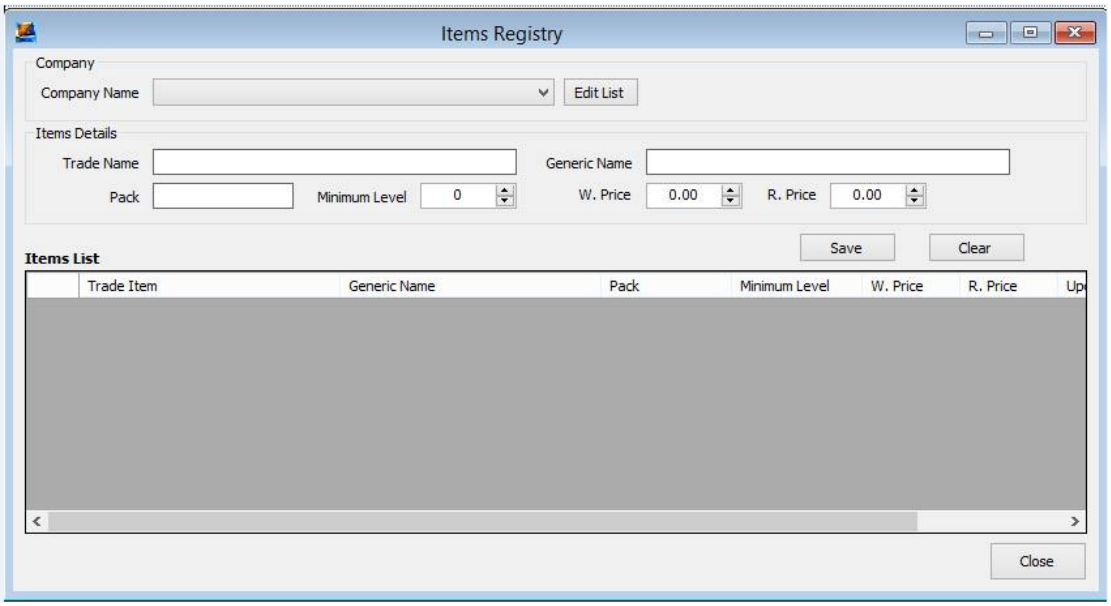


Figure 2B: Items Registry

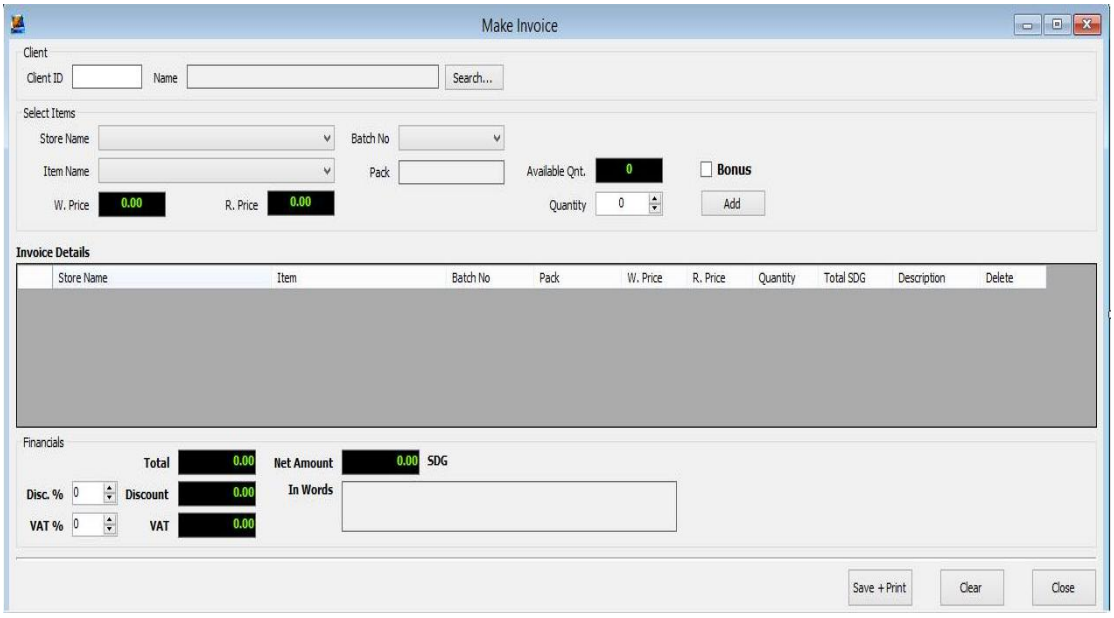


Figure 3B: Make Invoice

The system contains large number of modules , variables and structures and also a large number dependencies . To understand and analyzed the overall structure of system .

Static information is generated for whole software and visualized using Rigi. Tool . The visualized graph shows the structure of ShM system, which is over one million lines of code .

References

BISHARE SUFI ABDI **, "Framework for Measuring Perceived Quality in Technical Documentation " ,** University of Gothenburg Chalmers University of Technology , February 2013.

E. Chikofsky and J. Cross, "**Reverse Engineering and Jan. Design Recovery a Taxonomy**", *IEEE Software*, vol.7(1),1990, pp. 13-17

GolaraGarousi , " **A Hybrid Methodology for Analyzing Software Documentation Quality and Usage "** , UNIVERSITY OF CALGARY , September 2012

Institute of Electrical and Electronics Engineers.*Standard for Software Maintenance*. New York, IEEE Std. 1219-1998.

IzzulHidayatNaisan and SuhaimiIbrahim , "**Reverse Engineering Process to Support Software Design Document Generator "** , 2010

Joris Van Geet , Peter Ebraert , Serge Demeyer , "**Redocumentation of a Legacy Banking System"** , 2010

M. Torchiano1 F. Ricca2 P. Tonella3 **" Empirical comparison of graphical and annotation-based re-documentation approaches"** , 2009

N. Sugumaran1 and S. Ibrahim2 ,**"An Evaluation on Software Redocumentation Approaches and Tools in Software Maintenance**" http://www.ibimapublishing.com/journals/CIBIMA/cibima.html Vol. 2011 (2011), Article ID 875759

NoelaJemutai Kipyegen1 and William P. K. Korir2 ," **Importance of Software Documentation "** ,IJCSI International Journal of Computer Science Issues,Vol. 10, Issue 5, No 1, September 2013 , ISSN (Print): 1694-0814 | ISSN (Online): 1694-0784 www.IJCSI.org

SugumaranNallusamy and SuhaimiIbrahim **," A Review of Redocumentation Approaches " ,**UniversitiTeknologi Malaysia ,2009

SugumaranNallusamy ,Suhaimi Ibrahim , **"A Software Redocumentation Process Using Ontology Based Approach in Software Maintenance "** , *International Journal of Information and Electronics Engineering, 2011*

Tung Doan :**" An evaluation of four reverse engineering tools for c++ applications "**. University of TamperM.Sc . thisis, 75 page +1appendix , 2008

Tilley, S. 2008. Three Challenges in Program **"Redocumentation for Distributed Systems."** In Proceedings of IEEE Conference 2008 .