JOURNAL OF SOFTWARE ENGINEERING & INTELLIGENT SYSTEMS
ISSN 2518-8739
30th April 2018, Volume 3, Issue 1, JSEIS, CAOMEI Copyright © 2016-2018
www.jseis.org

A framework for software re-documentation using reverse engineering approach

[1]Nasrin Ismail Mohamed, [2]Nisreen Beshir Osman
College of Computer Science and Information Technology, Sudan University of Science and Technology
Department of Computer Science, Bayan College for Science and Technology, Khartoum, Sudan
E-mail: [1]nesreenimmg@gmail.com, [2]nisreenbeshir@gmail.com

ABSTRACT

During software evolution, programmers spend time and effort in the comprehension of programs and that is due to the fact that the documentation is often incomplete, inconsistent and outdated. In order to avoid these problems, software could be re-documented. Software re-documentation enables the understanding of software that aids the support, maintenance and evolution of software. Re-documentation is implemented by different approaches. Reverse engineering is one of these approaches that provide a better understanding of an existing system by maintainers and developers, especially when faced by a large and evolving legacy system. This study proposes a framework for systems re-documentation based on reverse engineering approach. The re-documentation is done using a reverse engineering tool that generates graphical representations of a system which are then used to produce documentation in the form of a standard document UML notation. Since, the quality of the generated documentation is important for program understanding and software evolution, the study also proposes a model for evaluating the quality of the generated documentation. The Documentation Quality Model (DQM) was validated and result of the evaluation showed that the documentation generated using reverse engineering was usable, up-to-date and complete.

Keywords: reverse engineering; software re-documentation; code understanding; software maintenance; documentation quality model;

1. INTRODUCTION

Software documentation is an important aspect of both software projects and software engineering in general. Unfortunately, the current perception of documentation is that it is outdated, irrelevant and incomplete. For the most part, this perception is probably true. A key goal of re-documentation is to provide easier ways to visualize relationships among program components so that we can recognize and follow paths clearly. There are many approaches that were used to implement the re-documentation process; one of these is using the reverse engineering approach. Developers tend to be focusing on source code since code is the most reliable source to be referred to as the system representation. Therefore, reverse engineering could be used to generate the documentation directly from the source code.

The quality and usefulness of documentation is one of the key factors that affect the quality and consistency of software. Accordingly, generating quality documentation through re-documentation process is important for program comprehension and software evolutions [8]. The study presented in this paper proposes a framework for system documentation based on reverse engineering approach. It also proposes a model for evaluating the quality of the generated documentation.

2. BACKGROUND

2.1 Software re-documentation

Re-documentation is the process of analyzing a software system to represent it with a meaningful alternate view intended for a human audience. Tilley [10] defined re-documentations as follows:

"Program re-documentation 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".

The main purpose of re-documentation is to make sure the software teams understand the legacy system [9]. There are two steps that are necessary to re-document a system. These are:

- Extract facts about the system which can be done starting from a static representation of the system (e.g., the source code).

- These extracted facts need to be combined and transformed into the correct documentation format (e.g., UML diagrams). [5]. Figure (1) shows the steps of the re-documentation process.

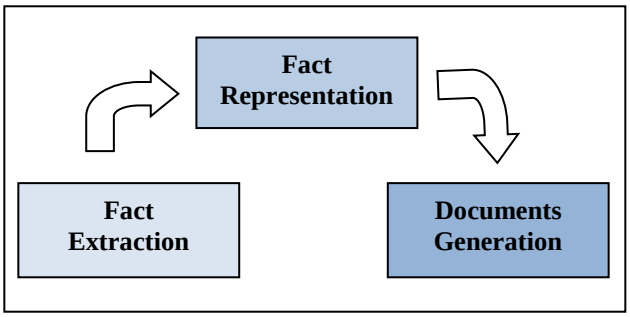

Figure.1 Re-documentation process

There are many re-documentation techniques in industry and research. Most of these techniques can be categorized to few approaches. Some of the significant approaches and tools which can contribute to the development of quality documentation are listed below:

I. **XML Based Approach**: XML based approach is one of the common re-documentation approaches. By using XML the technical writer or software engineer can create their own format such as <CONSTRANT>, <TASK> ,<FILE>, ...etc .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 [5].

II. **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). The generated TSs are stored in repository and produced documentation in a uniform way [5].

III. **Incremental Re-documentation:** One of the common issues in maintaining the system is to record the changes requested by customer or user the source code. The Incremental Re-documentation approach rebuilds the documentation incrementally after changes are done by the programmer [5].

IV. **The Ontology Based Approach:** It produces a schema from the legacy system to describe the context of the software system. The schema should be able to capture the artifacts from the latest version of the software system by establishing a reverse engineering environment [6].

2.2 Reverse engineering

The most common approaches of re-documenting software are named as reverse engineering. The IEEE Standard for Software Maintenance [3] defines reverse engineering as: "the *process of extracting software system information (including documentation) from source code*." In the context of software engineering reverse engineering is defined by Chikofsky and Cross [1] as: *"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".*

Reverse engineering captures the design information from the existing source code. It is the inverse procedure of forward engineering, as forward software engineering process goes from certain specification or target architecture toward building the system, the reverse engineering process goes from low abstract level to higher as shown in Figure 2.

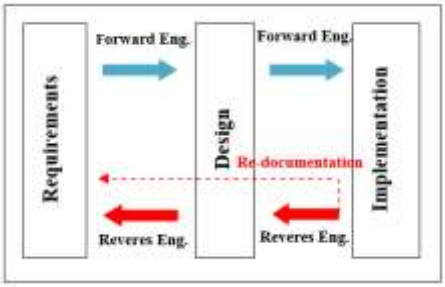

Figure. 2 Difference between forward and reverse engineering

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.

2.3 Documentation quality

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 and therefore should be up-to date, complete, consistent and usable [4].

According to Kitchenham "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. To understand and measure quality, many models of quality and quality characteristics have been introduced. The IEEE Std 1061-1998 [3] used a set of factors such as Functionality, Reliability, Efficiency, Maintainability, Portability and Usability to distinctively assess the quality.

Garousi [2] presented the hybrid methodology for Software documentation quality provide a meta-model 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* [2]. The key performance indicators (KPIs) framework, were also created by focusing on the quality attributes of the document: *Structure, Contextual, Accuracy and Accessibility* [7].

3. METHODOLOGY

This part summarizes the two practical steps used to re-document software using the proposed framework and measures the Quality of the document generated.

3.1 System re-documentation

The selected system was Shopping Management System which is a Service-Oriented system that provides services for purchasing items 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. The system contains large number of modules, variables and structures and also large number dependencies.

3.2 Rigi tool

Rigi tool environment provides automated reverse engineering activities that could be used to understand and analyze the overall structure of a system. Static information is generated for whole software and visualized using the tool. Figure (3) shows the conceptual architecture of the Rigi tool.

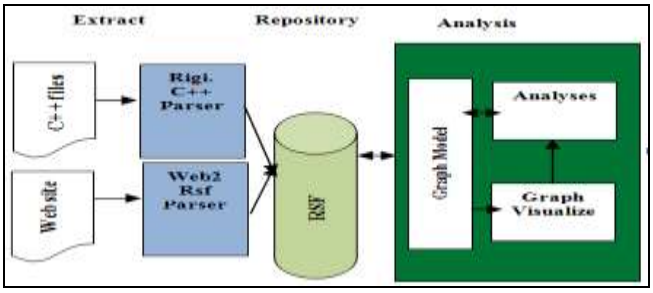

Figure. 3 Rigi conceptual architecture

The Rigi Workbench window and a root window are shown in figure (4).

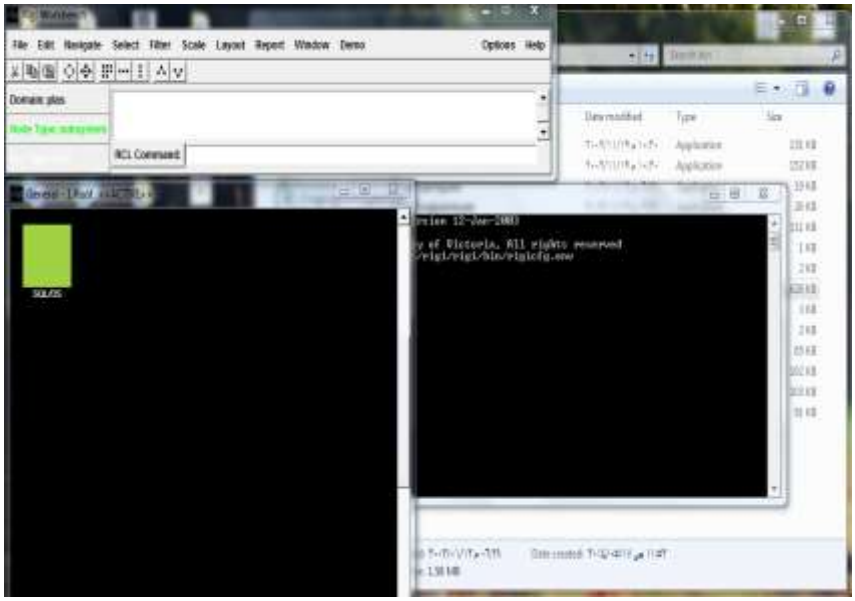

Figure. 4 Rigi workbench window

- **Fact Extraction.** Rigi includes parsers to read the source code of the subject system. It also enables viewing information stored in Rigi standard Format (RSF) files.
- **Fact Representation.** Represent and visualize the extracted information as directed graph, the initial window titled Root in figure 4 is used to display the parent(s) of the subsystem hierarchy (SQL/DS node).

A tree-like structure represented in figure (5) was used to manage the complexity for large information spaces.

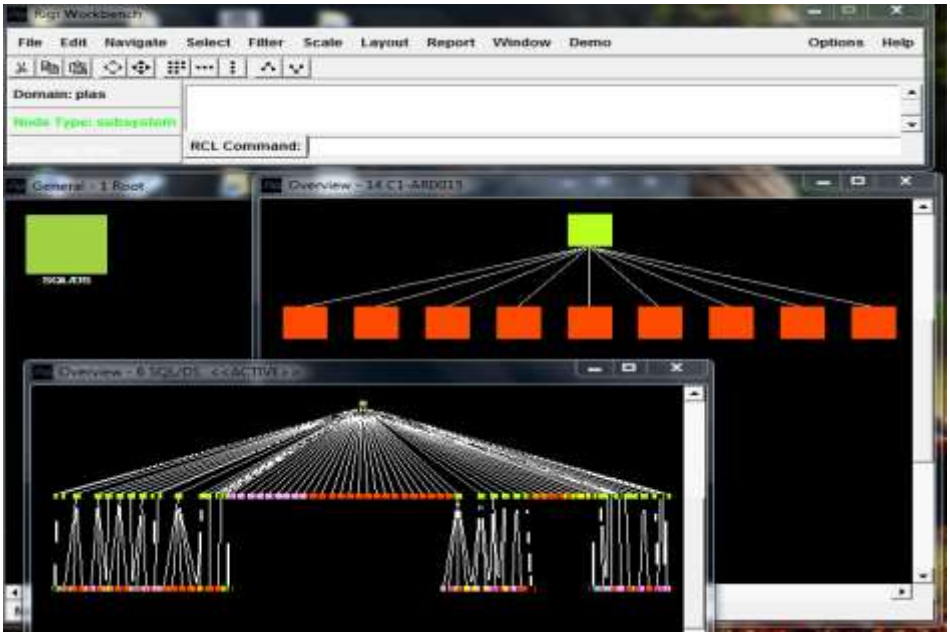

Figure. 5 Fact representation

- **Documentation Generation.** To generate the documentation, it was not possible to automate all the work so some parts required human expert interpretation in order to complete work.

UML notation was chosen to generate a standard graphical documentation. Rigi has been used for static reverse engineering. The extracted static information is viewed as directed graphs. The static dependency graph contains approximately the same information as a class diagram. The Table (1) enumerates the main UML class diagram 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. 1 Class diagram construct vs Rigi graph constructs

Constructs		
A UML class diagram	**A Rigi static dependency graph**	**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

In Rigi, classes and interfaces have their own node types, methods and variables inside a class in UML class diagram as shown in figure (6).

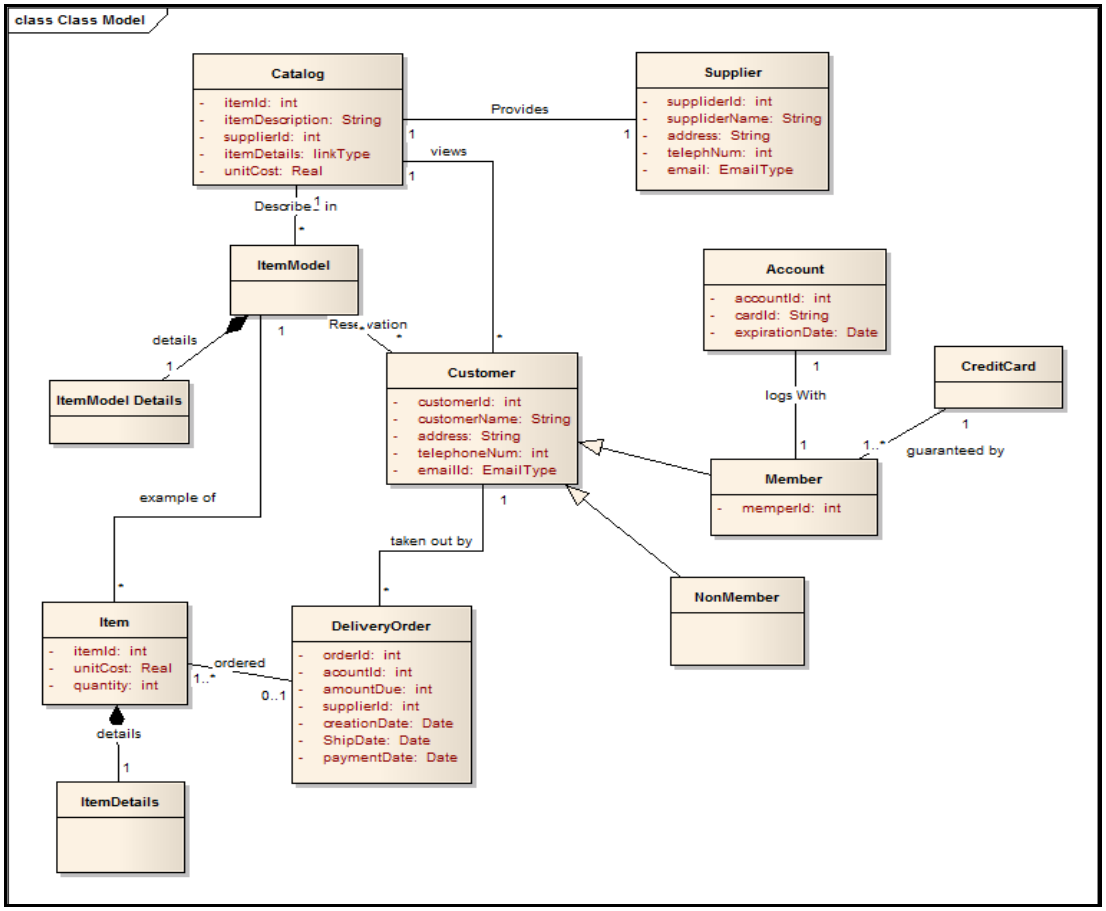

Figure. 6 Shopping management system class diagrams

3.3 Document quality model

To classify quality- related attributes of the documentation generated by the tool a Document Quality Model (DQM) was constructed. The model consists of two main attributes: Usability and Benefit. The attributes of the model are shown in Figure (7).

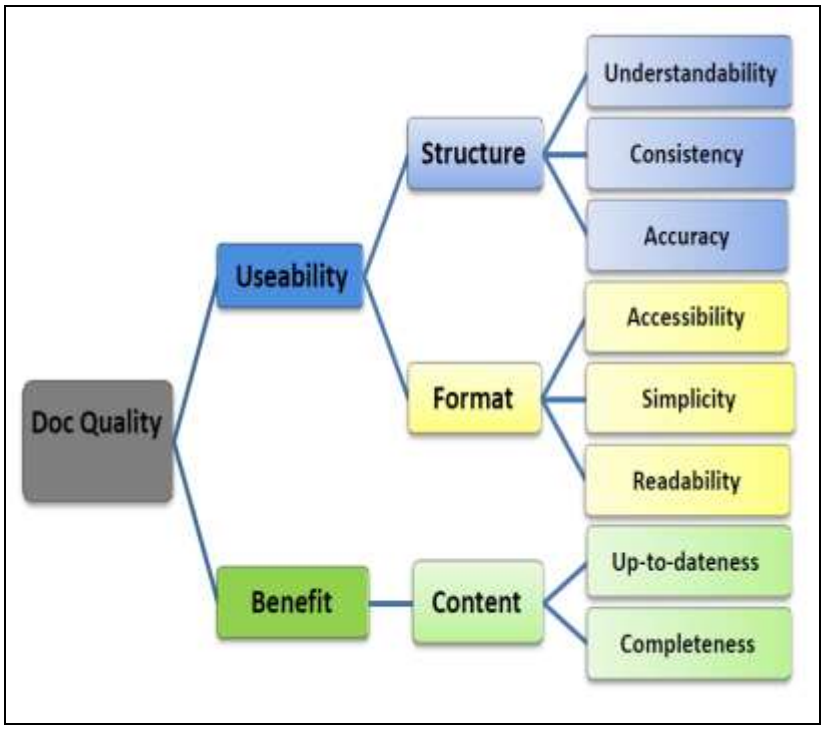

Figure. 7 Document quality model

4. RESULT AND DISCUSSION

The quality of the documents generated was measured using the DQM. Table (2) below shows the results of the evaluation. According to these results the generated document was understandable, consistent, accurate, accessible, simple, and readable. The results also showed the document was up-to-date and complete.

Table. 2 Documents evolution result

Quality Attributes		Document Evaluate
Usability	Understand ability	√
	Consistency	√
	Accuracy	√
	Accessibility	√
	Simplicity	√
	Readability	√
Benefit	Up-to-dateness	√
	Completeness	√

The Usability was measured by set of attributes. These are: understandability, consistency, accuracy, accessibility, simplicity and readability. UML notation chosen was accepted as standard for visualizing, understanding and documented software systems and this proved that UML document format have all usability aspect of the model. The Model measured Benefit criteria by content of documents, it should be up-to-date and complete. The document generated of the system is up-to-date since it depends on the latest version of the software system. It is complete and accurate since it is derived from the actual all source code.

5. 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 resulted document consistent with the code at all times. Therefore, reverse engineering is very effective to understand large software systems then re-documented.

REFERENCES

1. Chikofsky E., Cross J., "Reverse Engineering and Jan. Design Recovery Taxonomy", *IEEE Software*, vol.7 (1). 1990, pp. 13-17
2. Garousi G., " A Hybrid Methodology for Analyzing Software Documentation Quality and Usage ", UNIVERSITY OF CALGARY. September 2012.
3. Institute of Electrical and Electronics Engineers. *Standard for Software Maintenance*. New York, IEEE Std. 1219-1998.
4. Jemutai N.1, 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
5. Naisan I. and Ibrahim S., "Reverse Engineering Process to Support Software Design Document Generator ". 2010
6. Nallusamy S., Ibrahim S., "A Software Redocumentation Process Using Ontology Based Approach in Software Maintenance "*International Journal of Information and lectronicsEngineering,2011*.
7. SUFI ABDI B., "Framework for Measuring Perceived Quality in Technical Documentation", University of Gothenburg Chalmers University of Technology, February 2013.
8. Sugumaran N., Ibrahim S., "*An Evaluation on Software Redocumentation Approaches and Tools in Software Maintenance*". Vol. 2011 (2011), Article ID 875759 http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
9. Sugumaran N., Ibrahim S. 2," A Review of Re-documentation Approaches", University Technology Malaysia, 2009.
10. Tilley, S. 2008. Three Challenges in Program "Re-documentation for Distributed Systems." In Proceedings of IEEE Conference 2008.

AUTHORS PROFILE