International Journal of Computer Applications Technology and Research
Volume 3– Issue 12, 785 - 790, 2014, ISSN:- 2319–8656

Reverse Engineering for Documenting Software Architectures, a Literature Review

Hind Alamin Mohamed
College of Computer Science and Information Technology,
Sudan University of Science and Technology,
SUST University, SUDAN

Hany H Ammar
Lane Computer Science and Electrical Engineering Department,
College of Engineering and Mineral Resources,
West Virginia University, USA

Abstract: Recently, much research in software engineering focused on reverse engineering of software systems which has become one of the major engineering trends for software evolution. The objective of this survey paper is to provide a literature review on the existing reverse engineering methodologies and approaches for documenting the architecture of software systems. The survey process was based on selecting the most common approaches that form the current state of the art in documenting software architectures. We discuss the limitations of these approaches and highlight the main directions for future research and describe specific open issues for research.

Keywords: Reveres Engineering; Software Architecture; Documenting Software Architectures; Architectural Design Decisions.

1. INTRODUCTION

Reverse engineering has become one of the major engineering trends for software evolution. Reverse engineering is defines as the process of analyzing an existing system to determine its current components and the relationship between them. This process extracts and creates the design information and new forms of system representations at a higher level of abstraction [1, 2]. Garg et al. categorized engineering into forward engineering and reverse engineering. Both of these types are essential in the software development life cycle. The forward engineering refers to the traditional process for developing software which includes: gathering requirements, designing and coding process till reach the testing phase to ensure that the developed software satisfied the required needs [1]. While reverse engineering defined as the way of analyzing an existing system to identify its current components and the dependencies between these components to recover the design information, and it creates other forms of system representations [1, 2].

Legacy systems are old existing systems which are important for business process. Companies rely on these legacy systems and keep them in operations [2]. Therefore, reverse engineering is used to support the software engineers in the process of analyzing and recapturing the design information of complex and legacy systems during the maintenance phase [2, 3].

In addition, the main objectives of reverse engineering are focused on generating alternative views of system's architecture, recover the design information, re-documentation, detect limitations, represent the system at higher abstractions and facilitate reuse [1, 2, 4].

The main purpose of this survey paper is to achieve the following objectives: provide a literature review on the existing reverse engineering methodologies for documenting the architecture of software systems, and highlights the open issues and the directions for future research.

The rest of the paper is organized as follows: *Section 2*; presents a literature review of the common existing researches on reverse engineering from different perspectives. *Section 3*; highlights the new research areas as open issues for future works. Finally, concludes with summarizing the main contribution and the future research.

2. LITERATURE REVIEW

Program understanding plays a vital role in most of software engineering tasks. In fact; the developers use the software documentation to understand the structure and behavior of existing systems [4, 5]. However, the main problem that developers face is that the design document or others software artifacts were out-of-date to reflect the system's changes. As a result, more effort and time needed for understanding the software rather that modifying it [4, 5]. The following sections will introduce the most common reverse engineering approaches that focused in documenting the architecture of software from different perspectives.

2.1 Reverse Engineering for Understanding Software Artifacts

Kumar explained that developers should understand the source code based on the static information and dynamic information [5]. The static information explained the structural characteristic of the system. While dynamic information explained the dynamic characteristics or behaviors of the system. Hence, these details help the developers on understanding the source code in order to maintain or evaluate the system. However, Kumar clarified that few reverse engineering tools supported both of dynamic and static information [5]. Therefore, he presented alternative methodology to extract the static and dynamic information from existing source code. This methodology focused on using one of the reverse engineering tools; namely, *Enterprise Architect (EA)* to extract the static and dynamic views.

Additionally, all of the extracted information was represented in form of Unified Modeling Language (UML) models. The main purpose was to get the complementary views of software in the form of state diagrams and communication diagrams. The stages of this methodology are summarized as it shown in Figure 1.

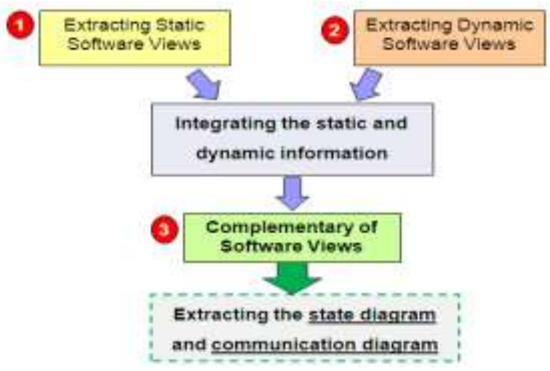
1 Extracting Static Software Views
2 Extracting Dynamic Software Views
Integrating the static and dynamic information
3 Complementary of Software Views
Extracting the state diagram and communication diagram

Figure 1. Reverse Engineering thorough Complementary Software Views [5]

This proposed methodology was very useful for supporting developers to understand the software artifacts of existing software systems. However, the methodology needs to support additional stakeholder beside the developers in order to identify the stakeholders' concerns and their decisions about the whole system.

2.2 Model Driven Reverse Engineering

Model driven reverse engineering (MDRE) was proposed as described in [6] to improve the traditional reverse engineering activities and legacy technologies. It is used to describe the representation of derived models from legacy systems to understand their contents. However, most of MDRE solutions focused on addressing several types of legacy system scenarios, but these solutions are not complete and they do not cover the full range of legacy systems. The work also introduced several reverse engineering processes such as: the technical/functional migration, processes of MDRE [6].

Recently, Hugo et al. presented a generic and extensible MDRE framework called "MoDisco". This framework is applicable to different refactoring and re-documentation techniques [6]. The architecture of MoDisco is represented in three layers, each layer is comprised of one or more components (see Figure 2). The components of each layers provided high adaptability because they are based on the nature of legacy system technologies and the scenario based on reverse engineering.

However, the MoDisco framework was limited to traditional technologies such as: JAVA, JEE (including JSP) and XML. This framework needs to be extended to support additional technologies and to add more advanced components to improve the system comprehension, and expose the key architecture design decisions.

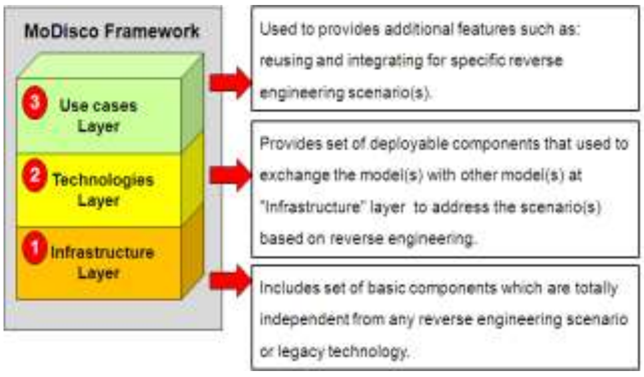
MoDisco Framework
3 Use cases Layer
2 Technologies Layer
1 Infrastructure Layer
Used to provides additional features such as: reusing and integrating for specific reverse engineering scenario(s).
Provides set of deployable components that used to exchange the model(s) with other model(s) at "Infrastructure" layer to address the scenario(s) based on reverse engineering.
Includes set of basic components which are totally independent from any reverse engineering scenario or legacy technology.

Figure 2. MoDisco Framework's Architecture [6, p9]

2.3 Documenting of Architectural Design Decisions (ADDs)

Historically, Shaw and Garlan introduced the concepts of software architecture and defined the system in terms of computational components and interactions between these components as indicated in [7]. Moreover, Perry and Wolf defined software architecture in terms of elements, their properties, and the relationships among these elements. They suggested that the software architecture description is the consequence of early design decisions [7].
Software architecture is defined by the recommended practice (ANSI/IEEE Std 1471-2000) as: the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution. Software architecture development is based on a set of architectural design decisions (ADDs). This is considered as one of the important factors in achieving the functional and non-functional requirements of the system [8].
Che explained that the process of capturing and representing ADDs is very useful for organizing the architecture knowledge and reducing the possibility of missing this knowledge [8]. Furthermore, the previous research focused on developing tools and approaches for capturing, representing and sharing of the ADDs.

However, Che clarified that most of the previous research proposed different methods for documenting ADDs, and these methods rarely support architecture evaluation and knowledge evaluation in practice [8]. Accordingly, Che et al. presented an alternative approach for documenting and evaluating ADDs. This approach proposed solutions described in the following subsections [8, 9]:

2.3.1 Collecting of Architectural Design Decisions

The first solution focused on creating a general architectural framework for documenting ADDs called the Triple View Model (TVM). The framework includes three different views for describing the notation of ADDs as shown in Figure 3. It also covers the features of the architecture development process [8, 9].

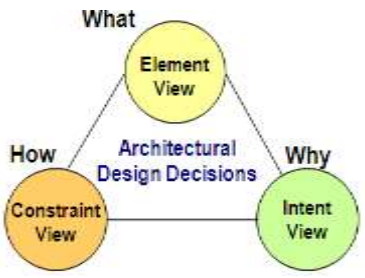

Figure 3. Triple View Model Framework [8, p1374]

As it shown in Figure 3; the *Element View* describes the elements that should be defined to develop the architecture; such as: Computation elements, Data elements, and Connector elements. The *Constraint View* explains how the elements interact with each other by defining what the system should do and not to do, the constraint(s) on each element of the element view. Additionally, define the constraints on the interaction and configuration among the elements.

Finally, the *Intent View* includes the rationale decision that made after analyzing all the available decisions, Moreover, the selection of styles and patterns for the architecture and the design of the system.

2.3.2 Scenario-Based Documentation and Evaluation Method

The second solution called SceMethod is based on the TVM framework. The main purpose is to apply the TVM framework by specifying its views through the end-user scenarios; then manage the documentation and the evaluation needs for ADDs [8, 10].

2.3.3 UML Metamodel

The third solution is focused on developing the UML Metamodel for the TVM framework. The main purpose was to make each view of TVM specified by classes and a set of attributes for describing ADD information. Accordingly, this solution provided the following features [8]: a) establish traceable evaluation of ADDs, b) apply the evaluation related to the specified attributes, c) support multiple ways on documenting during the architecture process and allow explicit evaluation knowledge of ADDs.

Furthermore, TVM and SceMethod solution was validated in using a case study to ensure the applicability and the effectiveness. Supporting the ADD documentation and evaluation in geographically separated software development (GSD) is currently work in progress.

2.4 Comparison of Existing Architectural Design Decisions Models

Researchers made a great of effort to present related tools and models for capturing, managing, and sharing the ADDs. These proposed models were based on the concept of architectural knowledge to promote the interaction between the stakeholders and improve the architecture of the system [8, 11].

Accordingly in [11], Shahin et al. presented a comparison study that is based on surveying and comparing the existing architectural design decisions models. Their comparison included nine ADD models and used six criteria based on desired features [11, 12]. The main reason was to investigate the ADD models to decide if there are similarities and differences in capturing the ADDs. Moreover, the study aimed at finding the desired features that were missed according to the architecture needs [11]. The authors in [11] classified the ADD elements into two categories: *major elements* and *minor elements*. The *major elements* refer to the consensus on capturing and documenting ADDs based on the constraints, rationale, and alternative decisions. While the *minor elements* refer to the elements that used without consensus on capturing and documenting the ADDs, such as: stakeholders, problem, group, status, dependency, artifacts, and phase/iteration.

The main observations of this comparison study are highlighted as follow: 1) all of the selected ADD models included the major elements and used different terms to express similar concepts of the architecture design. 2) Most ADD models used different minor elements for capturing and documenting ADDs. 3) All the selected ADD models deal with the architecture design as a decision making process. 4) While not all of them are supported by tools, some were based on only textual templates for capturing and documenting ADDs. 5) The most important observation was that most of existing ADD tools do not provide support for ADD personalization which refers to the ability of stakeholders to communicate with the stored knowledge of ADD [11, 12] based on their own profile.

We summarize the approaches and methodologies described in this section in Table 1. The main observation is that existing methods are focused on the developer's concerns and viewpoints as the main stakeholder. Recent approaches such as: Triple View Model (TVM) [8], scenario-based method (SceMethod) [9], and managing ADDs [10] suggested the need for alternative solutions for supporting ADDs personalization for different stakeholders.

3. OPEN ISSUES

We describe in this section the open issues that require further research based on the research work described in the previous section. These issues are listed as follows:

- There is a significant need to develop alternative approaches of reverse engineering for documenting the architectures that should simplify and classify all of the available information based on identifying the stakeholders' concerns and their decisions about the system.
- Improve the system's comprehension by establishing more advanced approaches for understanding the software artifacts. These approaches should help in documenting the architecture at different levels of abstractions and granularities based on the stakeholders concerns.
- Finally, it's important to support multiple methods and guidelines on how to use the general ADDs framework in the architecting process. These methods should be base on the architecture needs, context and challenges in order to evaluate the ADDs in the architecture development and evolution processes.

Table 1. Examples of some Methodologies and Approaches for Documenting Software Architecture

#	Author (year)	Problem Statement	Proposed Solution(s)	Results and Findings	Limitation(s)
1	**Kumar (2013)**	Reverse engineering for understanding the software artifacts	- Alternative methodology to extract the static and dynamic information from the source code. - The main purpose is to get complementary views of software systems.	- This methodology support *developers* to achieve the reverse engineering goals in order to understand the artifacts of software systems.	This methodology needs to support additional stakeholder beside the developers in order to identify the stakeholders' concerns and their decisions about the whole system.
2	**Hugo et al (2014)**	Understanding the contents of the legacy systems using model driven reverse engineering (MDRE)	- Generic and extensible MDRE framework called "MoDisco". - This framework is applicable to different types of legacy systems.	- MoDisco provided high adaptability because it is based on the nature of legacy system technologies and the scenario(s) based on reverse engineering.	MoDisco should extend to support additional technologies and include more advanced components to improve the system comprehension.
3	**Che et al (2011)**	Collecting architectural design decisions (ADDs)	- Triple View Model (TVM) an architecture framework for documenting ADDs.	- TVM framework includes three different views for describing the notation of ADDs. - TVM covers the main features of the architecture process.	TVM framework should extend to manage the evaluation and documentation of ADDs by specifying its views through the stakeholders' scenarios.
4	**Che et al (2012)**	Managing the documentation and evolution of the architectural design decisions	- Scenario based method (*SceMethod)* for documenting and evaluating ADDs. - This solution is based on TVM. The main purpose is to apply TVM for specifying its views through end-user scenario(s).	- Manage the documentation and the evaluation needs for ADDs through stakeholders' scenario(s).	There is a need to support multiple ways on managing and documenting the ADDs during the architecture process.

#	Author (year)	Problem Statement	Proposed Solution(s)	Results and Findings	Limitation(s)
5	**Che (2013)**	Documenting and evolving the architectural design decisions	- Developed UML Metamodel for the TVM framework. The main purpose was to make each view of TVM specified by classes and a set of attributes for describing ADDs information.	- Apply the evaluation related to the specified attributes and establish traceable evaluation of ADDs, - Allow explicit evaluation knowledge of ADDs. - Support multiple ways for documenting ADDs during the architecture process.	This solution is focused on the developers view point and their work is currently in progress to support the ADD documentation and evaluation in geographically separated software development (GSD).
6	**Shahin et al (2009)**	A survey of architectural design decision models and tools	- The purpose of this survey was to investigate ADD models to decide if there are any similar concepts or differences on capturing ADD. - The survey classified ADD concept into two categories: *Major elements* which refer to the consensus on capturing and documenting ADD based on the constraint, rationale and alternative of decision. While the *Minor elements* refers to the elements that used without consensus on capturing and documenting ADD. - Moreover, to clarify the desired features that are missed according to the architecture needs	- All of selected ADD models include the *major elements*. - Most of ADD models are based on using different *minor elements* for capturing and documenting the ADD. - All of selected ADD models deal with the architecture design as the decision making process. - Not all models were supported by tools. Hence, some of these ADD based on *text template* for capturing and documenting ADDs. - However, most of existing ADD tools do not support the ability of stakeholders to communicate with the stored knowledge of ADD.	There is a need to focus on stakeholder to communicate with the stored knowledge of ADDs. This could be achieved by applying the scenario based documentation and evaluation methods through stakeholders' scenario(s) to manage the documentation and the evaluation needs for ADDs.

4. CONCLUSIONS

This paper presented a survey on the current state of the art in documenting the architectures of existing software systems using reverse engineering techniques. We compared existing methods based on their findings and limitations. The main observation is that existing methods are focused on the developer's concerns and viewpoints as the main stakeholder.

We outlined several open issues for further research to develop alternative approaches of reverse engineering for documenting the architectures for development and evolution. These issues show the need to simplify and classify available information based on identifying the stakeholders' concerns and viewpoints about the system, improve comprehension by documenting the architecture at different levels of abstractions and granularities based on the stakeholders concerns, and support multiple methods and guidelines on how to use the ADDs framework based on the architecture needs, context and challenges in order to evaluate these ADDs during the architecture development and evolution processes.

5. ACKNOWLEDGMENTS

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

6. REFERENCES

[1] Mamta Gar and Manoj Kumar Jindal. 2009. Reverse Engineering – Roadmap to Effective software Design. In Proceedings of 2th International Journal of Recent Trends in Engineering. Information Paper, vol.1, (May 2009).

[2] Rosenberg, Linda H. and Lawrence E. Hyatt, Software re-engineering. Software Assurance Technology Center, 1996. http://www.scribd.com/doc/168304435/ Software-Re-Engineering1, visited on 26 April 2014.

[3] M. Harman, W. B. Langdon and W. Weimer.2013. Genetic Programming for Reverse Engineering, In R. Oliveto and R. Robbes, editors, In Proceedings of 20th Working Conference on Reverse Engineering (WCRE'13). Koblenz, Germany (14-17 October 2013), IEEE, 2013.

[4] M. Harman, Yue Jia, W. B. Langdon, Justyna Petke, Iman H. Moghadam, Shin Yoo and Fan Wu. 2014. Genetic Improvement for Adaptive Software Engineering. In Proceedings of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14). Hyderabad, India (2-3 June 2014), ACM, 2014.

[5] Niranjan Kumar. 2013. An Approach for Reverse Engineering thorough Complementary Software Views. In Proceedings of International Conference on Emerging Research in Computing, Information, Communication and Applications (ERCICA'13), 2013, 229-234.

[6] Hugo Brunelière, Jordi Cabot, Grégoire Dupé and Frédéric Madiot. 2014. MoDisco: A Model Driven Reverse Engineering Framework. Information and Software Technology 56, no. 8, 2014, 1012-1032.

[7] May Nicholas. 2005. A survey of software architecture viewpoint models. In Proceedings of 6th Australasian Workshop on Software and System Architectures, 2005, 13-24.

[8] Meiru Che. 2013. An Approach to Documenting and Evolving Architectural Design Decisions. In Proceedings of International Conference on Software Engineering (ICSE'13), San Francisco, CA, USA, IEEE, 2013. 1373-1376.

[9] Meiru Che and Dewayne E. Perry. 2011. Scenario-based architectural design decisions documentation and evolution. In Proceedings of Engineering of Computer Based Systems (ECBS'11), Las Vegas, NV, (27-29 April 2011), IEEE, 2011, 216-225.

[10] Meiru Che and Dewayne E. Perry. 2012. Managing architectural design decisions documentation and evolution. In Proceedings of 6th International Journal of Computers, 2012, 137-148.

[11] M. Shahin, P. Liang and M.R. Khayyambashi. 2009. Architectural design decision: Existing models and tools. In Proceedings of Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference, IEEE, 2009, 293-296.

[12] M. Shahin, P. Liang, and M.R. Khayyambashi. 2009. A Survey of Architectural Design Decision Models and Tools. Technical Report SBU-RUG-2009-SL-01. http://www.cs.rug.nl/search/uploads/Publications/shahin2009sad.pdf, visited on 8 July 2014.

7. AUTHORS BIOGRAPHIES

Hind Alamin Mohamed BSIT and MSCS, is a lecturer in Software Engineering department, College of Computer Science and Information Technology at Sudan University of Science and Technology (SUST). She has participated in the Scientific Forum for Engineering and Computer Students (December 2005) in SUDAN, and had the first prize of the Innovation and Scientific Excellence for the best graduated project on computer science in 2005. She has been teaching in the areas of Software Engineering and Computer Science since 2006. In 2010 she was the head of Software Engineering Department till December 2012. She is currently a PhD candidate in Software Engineering since 2013.

Hany H. Ammar BSEE, BSPhysics, MSEE, and PhD EE, is a Professor of Computer Engineering in the Lane Computer Science and Electrical Engineering department at West Virginia University. He has published over 170 articles in prestigious international journals and conference proceedings. He is currently the Editor in Chief of the Communications of the Arab Computer Society On-Line Magazine. He is serving and has served as the Lead Principal Investigator in the projects funded by the Qatar National Research Fund under the National Priorities Research Program. In 2010 he was awarded a Fulbright Specialist Scholar Award in Information Technology funded by the US State Department - Bureau of Education and Cultural Affairs. He has been the Principal Investigator on a number of research projects on Software Risk Assessment and Software Architecture Metrics funded by NASA and NSF, and projects on Automated Identification Systems funded by NIJ and NSF. He has been teaching in the areas of Software Engineering and Computer Architecture since 1987. In 2004, he co-authored a book entitled Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems, Addison-Wesley. In 2006, he co-authored a book entitled Software Engineering: Technical, Organizational and Economic Aspects, an Arabic Textbook