A Model for Rearchitecting Frameworks

Galal H. Galal-Edeen(1), Mohamed A. Shouman(2), Nissreen A. El-Saber(3)

(1) Department of Information Systems, Faculty of Computers and Information Systems, University of Cairo (Egypt) 
E-mail: galaledeenl@gmail.com
(2) Department of Decision Support Systems, Faculty of Computers and Informatics, 
Dean, University of Zagazig (Egypt) 
E-mail: drshouman@yahoo.com
(3) Department of Information Systems, Faculty of Computers and Informatics, 
University of Zagazig (Egypt) 
E-mail: nissr23@yahoo.com

ABSTRACT

Software rearchitecting is the process of obtaining a documented architecture for an existing system. There are many software rearchitecting frameworks which are based upon different concepts and context-related issues for a specific application or programming language, such as Rigi, Ciao, SPOOL, and Symphony, and Software Rearchitecting Action Framework (SRAF). Most of the frameworks focus on the reverse engineering process of source code. They neglect the role of stakeholders in enhancing and developing their systems. This paper presents a systematic analysis and comparative study for rearchitecting frameworks using generic architecture characteristics or elements. Based on the major requirements that should be available in the rearchitecting frameworks, the comparative study proceeds. An efficient model is proposed based on the trends that resulted from the comparative analysis. It considers the evaluation criteria of the compared frameworks. Conclusions and remarks are highlighted.

Keywords: Software Rearchitecting, Rearchitecting Framework, Frameworks Comparison, Rearchitecting Frameworks Evaluation

1- INTRODUCTION

Software rearchitecting has a major role in the maintenance of information systems. The act of rearchitecting, or obtaining the software system design, is generally understood to be luxury and many developers misunderstand its importance in providing suitable documentation for the undocumented software. Software rearchitecting is a critical process and requires special attention from the software architect. The majority of the rearchitected software is mission-critical, like hospitals, banks, financial and scientific applications.

Rearchitecting frameworks helped to improve the subject understanding by providing systematic approaches to rearchitecting process. However, many aspects of rearchitecting remain unclear. Vague aspects of rearchitecting include the following:

a. The scope of rearchitecting process – to what extent should the system rearchitected, should the source code be rearchitected
only or should the information system as a whole be included into the rearchitecting process.
b. The role of architect – actually, the architect’s role in the system development life cycle is often unclear [23]. Therefore, within the rearchitecting process, his or her role is not only limited to modifying the architecture resulting from the process, but also goes to validating the resulting architecture to make sure it is compatible with the original specifications.
c. The outcomes – what should be the rearchitecting outcomes? What kind of architecture should be delivered?
d. Verifying the resulting architecture – to what extent should and could the deliverables of rearchitecting be measured, verified or validated?
e. The level of the architecture that can be considered satisfying the rearchitecting needs – generally, detailed design specifications are not included into the architecture.

This paper investigates the concept of rearchitecting by examining five rearchitecting frameworks. These frameworks are Rigi [19], Ciao [4], SPOOL [15], Symphony [5], and Software Rearchitecting Action Framework (SRAF) [7]. The systematic analysis and comparison of rearchitecting frameworks using generic architecture characteristics derived from [23] are presented. In section 2 a brief introduction of software rearchitecting, terms that may conflict with it and its importance are presented. Rearchitecting frameworks which are under consideration in this research are introduced in section 3. In section 4, the evaluation criteria are presented and in section 5 the discussion and comparison are detailed. A new recommended model based on the comparison results is presented in section 6. Conclusion, findings and future work are presented in section 7.

2- SOFTWARE REARCHITECTING

Software rearchitecting is the process of obtaining a documented architecture for an existing system. Although such rearchitecting can make use of any available resource (such as documentation, stakeholder interviews, domain knowledge), the most reliable source of information is the system itself, either via its source code or via traces obtained from executing the system. The system users and all other stakeholders constitute vital sources of information. They provide insights into what’s wrong with the system. In this context, electronic technologies; such as e-commerce, e-learning ... etc., have special characteristics as far as rearchitecting is concerned. Their users are widespread all over the world and their requirements gathering can be assisted by electronic means rather than by ordinary interviews only.
Software rearchitecting is getting more important as software systems grow larger and more complex. Different terms are used to describe the reengineering process of software rearchitecting. Software reengineering includes radical and global changes in the software system under consideration while the software rearchitecting process mainly involves the architectural design extraction and manipulations. In this section the definitions and terms are presented.

2-1 TERMS CONFLICTING WITH SOFTWARE REARCHITECTING

Terms that may conflict with the term “software rearchitecting” include the terms of business process reengineering, software reengineering, reverse engineering, restructuring, and architecture recovery [7]. Reverse engineering is another term that may conflict with the rearchitecting process. Reverse Engineering can extract design information from source code, but the abstraction level, the completeness of the documentation, the degree to which tools and human analyst work together, and the directionality of the process are highly variable [7]. Software reengineering aims to deliver radical changes in the software system that may affect the system functions and to create versions of existing programs that exhibit higher quality and better maintainability. Architecture recovery can be seen as a discipline within the reverse engineering domain that is aimed at recovering the software architecture of a system [21].

While Wood et al (2003) agreed that architecture is an essential ingredient in the engineering of non-trivial systems, there was also general agreement that an “architectural storm” is brewing with many overlapping architectural buzzwords [25]. While it is easy to find references to “information architecture,” “enterprise architecture,” “system architecture,” “system-of-systems architecture,” “software architecture,” “communications architecture,” “hardware architecture,” “security architecture,” “data architecture,” and many other “architectures,” it is harder to find crisp definitions of any of them, or descriptions of how they should be used in the engineering discipline.

2-2 THE IMPORTANCE OF REARCHITECTING

Architecture represents high-level system components and the relationships among them governing their design, and evolution potential over time. Believing in the fact that Architectures are lost or eroded over a system’s lifetime due to their intangibility, this results in the need for techniques to recover or extract architectures from legacy systems. A legacy system is a computer system which continues to be used as a result of the difficulties existing in either of the replacement or the redesign processes [20]. Many people use this term to refer to “antiquated” systems [20]. They are potentially problematic [2] for several reasons. Legacy systems often run on obsolete hardware, and sometimes spare parts for such computers become increasingly difficult to obtain. Upgrading the hardware often results in an upgrade to the operating system also, with obsolescence consequences for the application software. These systems are often hard to maintain, improve, and expand. This is due to a general lack of understanding of the system. The system designers may
have left the organization, leaving no one who knows how it works. Such a lack of understanding can be exacerbated by inadequate documentation or manuals getting lost over the years. Integration with new systems may also be difficult as new software may use completely different technologies.

Many programmers start to write code before they make the design. After they finish the software they forget about it till they are asked to maintain or enhance part of it. In such case, it is impossible to trace the bugs or the modifications. A reliable system that extracts the architectural design of the source code to facilitate the modification or enhancement process through architectural design retrieval is needed. Hence, the importance of the concept of rearchitecting processes started to emerge.

3- SOFTWARE REARCHITECTING FRAMEWORKS

The term framework is used in many contexts in software development [1]. A framework, in general, is a structure composed of parts that together support a structure. Examples of programming frameworks for constructing object-oriented graphical user interfaces (GUIs) are the Microsoft Foundation Classes (MFC) for building C++ GUIs and Swing for building Java GUIs. Object-oriented application frameworks are class libraries, but rather than writing your own main routine and calling library functions, the framework manages the event dispatching and you supply classes that the framework calls [1]. The application framework is a partially complete application, a skeleton that needs to be extended or fleshed-out. In this section the famous rearchitecting frameworks are introduced.

Researchers have designed several standalone frameworks to help developers understand large software systems, including Ciao [4], Dali [13], ManSART [26], PBS [12], Rigi [19], SPOOL [15], and TkSee [16]. These frameworks are necessary because, for many systems, there is no high-level documentation that accurately describes the current system implementation [3]. Even where documentation does exist, these tools are helpful because they can help developers assess how closely a system’s implementation matches its documented structure. In order to develop this research, some of the rearchitecting frameworks were investigated to come up with the suitable set of frameworks to be compared and hence to develop general specifications and characteristics that should be available in any future rearchitecting frameworks. The selection is based on the output presentation; which means whether the system delivers a visualized design of the software system or not. Table 1 summarizes the selection process.
Table 1 Framework Selection Criteria

<table>
<thead>
<tr>
<th>Framework</th>
<th>Visualized Output</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rigi (1993)</td>
<td>Yes</td>
<td>Rigi uses arcs and nodes to represent the architecture. Rigi has been developed to commercial software [24].</td>
</tr>
<tr>
<td>Ciao (1995)</td>
<td>Yes</td>
<td>Ciao is a graphical navigator for software and documentation repositories [4].</td>
</tr>
<tr>
<td>ManSART (1997)</td>
<td>No</td>
<td>ManSART manipulates recovered software architecture [26].</td>
</tr>
<tr>
<td>TkSee (1997)</td>
<td>No</td>
<td>TkSee is a source code exploration tool [16].</td>
</tr>
<tr>
<td>Dali (1998)</td>
<td>Yes</td>
<td>Dali uses Tarski relational Algebra to manipulate software architecture [13].</td>
</tr>
<tr>
<td>SPOOL (1999)</td>
<td>Yes</td>
<td>SPOOL provides a GUI with a lot of software presentation diagrams [15].</td>
</tr>
<tr>
<td>Symphony (2004)</td>
<td>Yes</td>
<td>Symphony uses a full UML GUI for the output [5].</td>
</tr>
<tr>
<td>SRAF (2007)</td>
<td>Yes</td>
<td>Software Rearchitecting Action Framework (SRAF) delivers a flowchart for the input open source code [7].</td>
</tr>
</tbody>
</table>

This research will focus on the comparison among the frameworks that satisfy the presentation condition for the output of the rearchitecting process. Hence, the comparison will proceed with Rigi, Ciao, Symphony, SPOOL and SRAF. Although Rigi was built on arc and node representation, it was fully automated to a working commercial software in 1998 by University of Victoria [24]. These frameworks have in common condition that the visualization of the architecture is by means of UML notations, arc and node notations, or special notations. Dali produces a visualized output but it uses relational algebra to produce it, which is not of a common feature and hence it will not be considered in the current comparative study.

3.1 INTRODUCING REARCHITECTING FRAMEWORKS

In spite of its importance nowadays, the optimal solution for the lack of documentation is still proving elusive. Comparing to the number of frameworks found in software architecture, there are fewer frameworks to deal with software rearchitecting. Several standalone frameworks have been designed to help developers understand large software systems. Chen (1995) introduced Ciao, a graph-based navigator that helps programmers query and browse structural connections embedded in different software and document repositories. Keller and Schauer (1999) introduced SPOOL, which is a prototype environment for reverse engineering with the aim of the recovering of C++ software structure.
Kazman and Carriere (1999) presented a structure framework for rearchitecting. Program understanding frameworks can help to answer questions about a software system at varying levels of abstraction. At the code-level of abstraction, frameworks can provide detailed answers to questions about the expressions and their values, and the functions whether in their actual programming language or in another languages.

Rearchitecting can be considered reverse engineering as a crucial part of the rearchitecting process. According to Lethbridge and Anquetil (1997), a reverse engineering approach should consist of extraction, abstraction, and presentation. The Extraction phase extracts information from source code, documentation, and documented system history. The abstraction phase abstracts the extracted information based on the objectives of the reverse engineering activity. Abstraction should distill the possibly very large amount of extracted information into a manageable amount. The presentation phase transforms abstracted data into a representation that is conducive to the user. In this aspect, two phases are added; preparation phase and validation phase. The preparation phase is responsible for making the source code ready for analysis. In the validation phase the resulting architecture should be revised for consistency with the original function of the system; the functional and nonfunctional requirements should also be mapped to the new system architectural design.

Objectives for rearchitecting code drive what is extracted, how it is abstracted, and how it is presented. For example, if the objective is to rearchitect with the associated goal to reengineer (let’s say into an object oriented product), architecture extraction is likely to be based on identifying and abstracting implicit objects, abstract data types, and their instances [22].

Other ways to look at system rearchitecting include using state machine information [9], or release history [8]. CAESAR [8] used the release history for a system. It tried to capture logical dependencies instead of syntactic dependencies by analyzing common change patterns for components. This allows identification of dependencies that would not have been discovered through source code analysis. It requires data from many releases. This method could be seen as a combination of identification of problematic components and architectural recovery to identify architectural problems [22].

These frameworks and tools are necessary due to the absence, for many systems; of high-level documentation that accurately describes the current system implementation.

Harris et al. (1995) outlined a framework for architecture reconstruction using a combined bottom-up and top-down approach. The framework consists of three parts: architecture representation, source code recognition engine and supporting library of recognition queries, and a “bird’s-eye” program overview capability. Another important class of frameworks attempts to find high-level abstractions which are used in software systems. The architecture-level frameworks aid in reconstructing system’s software architecture based on facts extracted from the system artifacts. Typically, these architecture level frameworks are not designed
to answer questions at the level of detail provided by the code-level frameworks [7].

Previous research in architecture reconstruction focused on recovering a single architectural view or few pre-selected views [5]. The application of these techniques usually involves three steps: extract raw data from the source, apply the appropriate abstraction technique, and present or visualize the information obtained. These steps are specific to the views to be reconstructed.

Lung (1998) introduced clustering technique which can be applied to software during various lifecycle phases. Lung (1997) demonstrated that clustering can be used to effectively support both software architecture portioning at the early phase in the forward engineering process and software architecture recovery of legacy systems in the reverse engineering process.

Kazman (1999) proposed an iterative reconstruction process where the historical design decisions are discovered by empirically formulating/validating architectural hypotheses. They point out the importance of modeling not only system information but also a description of the underlying semantics [14]. Their approach is currently extended to include the reorganization of recovered assets into software product lines.

A Focus approach was described in [6]. It contrasts a logical architecture with a physical one. By applying refinement to the logical and abstraction to the physical architecture, the two are brought together incrementally. Deursen et al. (2004) described Symphony, a process framework that has an explicit step for the discovery of the view that should be reconstructed in order to solve the problem at hand. All the previous works address a determined goal, concrete techniques, and certain fixed sets of views to be reconstructed.

Based on the survey above, some remarks on software rearchitecting approaches from the perspective of their visualization ideas, tools, goals, and stakeholders are:

1. Information presentation is different in each approach (UML, Views, special notation, …)
2. Architectural data refinement step is not included in most of the existed approaches.
3. The majority of these approaches focus on either stakeholder’s requirement analysis or source code reverse engineering, but not both.
4. All approaches surveyed have a determined goal, concrete techniques, and certain fixed sets of views to be reconstructed.

Some remarks on software rearchitecting frameworks concerning their platforms, tools, and stakeholders perspectives can be made:

1. Frameworks and tools are vital to the systems due to the absence, for many systems, of high-level documentation that accurately describes the current system implementation.
2. Existing rearchitecting frameworks focus on the reverse engineering process for the source code.

3. They neglect the role of stakeholders in enhancing and developing their systems.

4. Recent frameworks are oriented to certain languages, supporting their applications rearchitecting.

4- EVALUATION CRITERIA

In order to rearchitect a system, we need a number of vital aspects. First of all, this system may not be working as a standalone system; there can be other systems that are related to it. Functionally-dependent systems are more difficult than independent standalone system. The architecting of complex system involves consideration of multiple dimensions such as business requirements, costs, current architecture and future architecture etc. [10]. As Tang et al (2004) argued, the dimensions, or inputs, to the architecture process are themselves interrelated and cannot be considered in isolation within the overall process, the same idea fits the rearchitecting process.

The research determined the following criteria for the evaluation of the rearchitecting frameworks under consideration. The evaluation criteria are categorized into four main categories:

- Basic requirements: that allows the system to work, such as input data types and source code type and its availability to be processed via the framework.
- Human involvement and automation level: that measures the level to which the framework design allows for human manual processes or makes it automated process.
- Processing: this category evaluated the framework to verify if it has the reverse engineering process as part of it. Another processing issue is the refinement and abstraction of the resulted output.
- Presentation: in this category the type of presentation of the resulted architectural design. Visualizing the software is a crucial step in order to make it readable and understood.

In the following section the comparison takes place and all results are presented. After that, a proposed model for rearchitecting frameworks is presented.

5- COMPARISON RESULTS

This study uses basic requirements, human involvement and automation level, processing and outcomes presentation as fundamental elements to analyze rearchitecting frameworks. It shows that all rearchitecting frameworks support the purpose of software rearchitecting and selected by the previously mentioned
criteria related to the output visualization. Table 2 exhibits the criteria of the presented comparative study of the considered rearchitecting framework. In Table 2, “Yes” represents that the framework satisfies the condition or the feature is considered in its components and “No” represents that the framework doesn’t support it. “N/A” means the framework lacks this feature. Automation level and architect involvement are determined by “Partial” for the frameworks that have specific rules for each one of them; so that the rearchitecting activities can be accomplished by the architect and others by automated functions. “Major” and “Minor” are used for the full or small involvement into the rearchitecting activities; so that it is either done by the automated functions and procedures or by the human architect.

Table 2 Comparison of Rearchitecting Frameworks

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Basic Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Different Prog. Lang.</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Requires Input Source Code Specifications</td>
<td>C/C++</td>
<td>C/C++</td>
<td>C/C++</td>
<td>C++/ Java/ Smalltalk</td>
<td>Java</td>
</tr>
<tr>
<td>Human Involvement and Automation Level</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Automation Level</td>
<td>Minor</td>
<td>uses Aero</td>
<td>Minor</td>
<td>Major</td>
<td>Partial</td>
</tr>
<tr>
<td>Architect Involvement</td>
<td>Major</td>
<td>Partial</td>
<td>Major</td>
<td>Major</td>
<td>Partial</td>
</tr>
<tr>
<td>Processing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refinement Step</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>Reverse Engineering</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Presentation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Availability of Reasonable Output Information Representation</td>
<td>Nodes and arcs and graphical representation.</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Presentation Type (UML, Views, Special Notations, …)</td>
<td>Arcs &amp; Nodes</td>
<td>Entity Relationship Diagrams (ERD)</td>
<td>Special Notation</td>
<td>GUI</td>
<td>UML (Class Diagram )</td>
</tr>
</tbody>
</table>

Based on the results in the Table 2, all frameworks use the reverse engineering step as a crucial phase in the rearchitecting process. All of the compared frameworks are language-specific; four out of five are specific for C/C++ languages. However, SRAF supports Java source code rearchitecting in open source software. Different visualized presentation methods were used by the frameworks. Rigi used the nodes and arcs notations, Ciao used the database entity relationship diagrams (ERD). SRAF used the unified modeling language (UML) class diagram and flowchart which is considered privilege advantage.
Refining the resulted architecture is not verified in most of the compared frameworks. Refinement is only considered as a formal step in the SRAF which is considered as privilege advantage allover the considered frameworks. Architect involvement varied from architecture to another. It was major in Rigi and SPOOL, whereas partial in Ciao and SRAF.

6- PROPOSED FRAMEWORK MODEL

In order to get the ultimate benefit from a framework designed for the rearchitecting process, and depending on the above comparison and discussions, the main concept in building the presented model is to unify standard guideline characteristics in rearchitecting frameworks. These standard characteristics should be existed in the framework to verify its validity in the rearchitecting activities. These characteristics will be explained in the next section.

6.1 SPECIFIC REARCHITECTURE FRAMEWORKS REQUIREMENTS

Some basic characteristics should be existed in the rearchitecting framework to be successfully applied. First, it should accept the manipulation of different kind of source codes; hence it can accept any programming language as an input, which is not applicable in the most of the available frameworks. Second, the type and nature of the source code which inserted into the framework should be standard in some way. There are no clear coding conventions in open source code, for example. This forms a big obstacle facing all the rearchitecting process. Third, the involvement of the architect into the rearchitecting process is considered as basic characteristics. Rearchitecting requires software architecture experts [5]. These experts are supposed to study the system and the active involvement of stakeholder representatives, such as testers, developers, management, the business owning the system, and system users.

Furthermore, the automation level limits to which the rearchitecting process is subject to change. It depends on the software parts that are involved in the rearchitecting, recovery or maintenance, processes. In addition to that, the nature of the new user’s requirements will affect the types of changes that should be applied to the software system.

6.2 PROPOSED REARCHITECTING MODEL

Based on the above discussion, the rearchitecting optimal framework should satisfy most, or all, the mentioned requirements in section 6.1. In this section a model for the rearchitecting framework is presented. Fig.1 illustrates the main phases that should be considered when building a software rearchitecting framework.

First, the required inputs are prepared to be ready for processing. In this phase the source code files are extracted. To reach the optimal configuration in which it will accept any kind of programming languages considering the fact that many programmers are tend to use open source code more and more these days, standards verification step were added to the framework model. The software
source is tested if it conveys the standards that the framework accepts as input or not. Second phase is architecture extraction. This phase includes reverse engineering and architecture refinement, which are the most important in the framework due to its criticality. If the resulted architecture is not accurate then the whole rearchitecting process could be failed. Modifications and enhancements are made in the third phase, where the architect can make use of the available documentation and stakeholder’s new requirements. Fourth phase is concerned with the recoding of the system and finally to test the resulted system in the real life environment.

In this paper a comparison among software rearchitecting frameworks is presented. These frameworks are Rigi, Ciao, Symphony, SPOOL and SRAF. They have the visualized presentation step whether via an UML diagram, arc and node notation or other special notations. The research doesn’t include some function-specific frameworks, such as, architecture recovery frameworks and architecture maintenance frameworks.
Figure 1 Proposed Model for Software Rearchitecting Framework
7- CONCLUSIONS

The paper presents some recommendations and standard characteristics that have to be considered when designing any future rearchitecting frameworks. Software rearchitecting frameworks should ensure the acceptance of various kinds of source code. Furthermore, the involvement of the architect into the rearchitecting process is considered as basic characteristics and outlined in the framework. A comparative study is introduced to different kinds of rearchitecting frameworks. The recommendations rendered for the comparative study form the basis for the design of new software rearchitecting model. The model for the recommended software rearchitecting framework model can be used by researchers and architects to build their own rearchitecting frameworks with maximum benefit. Future work in this field can proceed by building a new rearchitecting framework and avoids the disadvantages mentioned in the surveyed rearchitecting frameworks.

8- REFERENCES


