A Reverse Engineering Process to Support Software Design Document Generator

Izzul Hidayat Naisan and Suhaimi Ibrahim
Center for Advanced Software Engineering (CASE)
Universiti Teknologi Malaysia
Kuala Lumpur, Malaysia
izzul@ic.utm.my, suhaimiibrahim@utm.my

ABSTRACT
Reverse engineering provides a better understanding of an existing system by maintenance team, especially when they are facing a large and evolving legacy system. 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. Therefore, generating the documentation directly from the source code makes the result consistent with the code at all times. This study focuses on the reverse engineering process to extract related information from source code of software. Three steps performed in this research. First, extract information from database to XML. Next step is processing data from XML into database. Finally, Word-like document is arranged from the database. The output of this process will be a software design document (SDD) which is an IEEE standard and should be significant for developer team in analyzing their old systems.

Keywords
Reverse engineering, software design document, re-documentation

1. INTRODUCTION
It is well known that software systems have played a great role to many companies nowadays. The business process has become so much involved with computer systems that an interruption or break down of software system can cause loss of huge amount [1].

Many software engineering activities done by the companies entail dealing with existing systems. Software maintenance, testing, quality assurance, reuse, and integration are only a few examples of software processes that involve existing systems. A key aspect related to all these processes is the identification of the components of a system and the comprehension of the relationships existing among them [2].

One way to identify the components of system is by analyzing its documentation. The documentation is needed to understand a system at a certain level of abstraction, in a limited amount of time. It is needed, for instance, when a system is migrated or re-engineered. It can be used to map functional modification requests as expressed by end users onto technical modification requests, and to estimate the cost of such modifications. Finally, documentation will help in the process of outsourcing maintenance or when engineers that are new to the system need to learn about the system [3].

However, in many cases, documentation is a missing item in the maintenance legacy software systems (e.g., configuration plan, quality plans, etc. [4]. 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 [5]. To lessen this problem, organizations try to re-document their software systems, but this is a costly operation that would benefit from a clear indication of the software documents to focus on.

The key point solution to the above problems is the software re-documentation. It reconstructs the software documentation, which either no longer exists or which has became obsolete. Incremental re-documentation rebuilds documentation in small incremental steps. Each step is taken at the end of the software change mini process and re-documents the part of source that is related to the change, and therefore was recently comprehended by the developer team [6].

This paper presents a reverse engineering process implemented by a tool to generate an IEEE standard of software documentation. It begins with the importance of document generation directly from source code. Next, some backgrounds on software maintenance, reverse engineering, software documentation, and software documentation are provided, followed by methodology, results and discussion. Finally, it describes conclusions and acknowledgment.

2. LITERATURE REVIEW
2.1 Software Maintenance
IEEE [7] defines software maintenance as “modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment”. It is reflected in this definition that software maintenance is a post-delivery activity: it begins when a system is delivered to the customer and encompasses all activities that keep the operational of the system and meet the customer’s requirements.
Another definition of software maintenance is provided by Pressman [8]. It is defined by describing four activities that are undertaken after a program is released for use as follows.

i. The first activity occurs because it is unreasonable to assume that software testing will uncover all latent errors in a large software system. During the use of any large program, errors will occur and be reported to the developer.

ii. The second activity occurs because of the rapid change that is encountered in every aspect of computing. Examples for these changes are: new operating system, new release of old ones, peripheral equipment or other system elements are frequently upgraded or modified.

iii. The third activity occurs when a software package is successful. As the software is used, recommendations for new capabilities, modifications to existing functions, and general enhancements are received from user.

iv. The fourth activity occurs when software is changed to improve future maintainability or reliability, or to provide a better basis for future enhancements.

2.2 Reverse Engineering

Although software reverse engineering originated in software maintenance, it is applicable to many problem areas. In the context of software engineering, as defined by Chikofsky and Cross [9], 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”. Another definition is provided by IEEE [7], “reverse engineering is the process of extracting software system information (including documentation) from source code”.

Chu et al. [10] introduced four phases of reverse engineering processes:

i. Context Parsing Phase. This phase analyzes the source code and extracts syntactic and semantic program information using techniques like token parsing and syntax analysis. The code then transforms into an intermediate form e.g. Abstract Syntax Tree (AST) which can be input for next phase.

ii. Component Analyzing Phase. Using the abstract representation of the source code created by the parser in the previous phase (i.e. the AST), the extraction model can be applied to find the components of interest which are to be extracted. In this phase, the components that matched the specifications of the extraction model are copied from the sources and stored partially in a component repository [11].

iii. Design Recovering Phase. Currently, with techniques that combine both structural and knowledge representation, analyzers can infer some high-level information successfully from the results of the previous phase.

iv. Design Recovering Phase Currently, with techniques that combine both structural and knowledge representation, analyzers can infer some high-level information successfully from the results of the previous phase.

2.3 Software Documentation

US Department of Defense [12] defines software documentation as technical data or information, including computer listing and printouts, which documents the requirements, design, or details of computer software, explains the capabilities and limitations of the software, or provides operating instructions for using or supporting computer software during the software’s operational life. From the DoD definition it is briefly described that software documentation provides the capabilities and limitations of the software or provide instruction for using the software.

2.3.1 Software Re-documentation

Rajlich [13] proposed a definition of software redocumentation and its difference with software reengineering as below.

“Software 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”.

From these definitions it is clearly stated that redocumentation is to recreate a documentation of existing system from the available resource. Actually the documentation has ever made in the past. Due to some conditions, the documentation is required to be reconstructed. It is also stated that redocumentation has very close relationship with software reengineering and reverse engineering.

2.3.2 Some Documentation Generators

Some redocumentation tools were examined and analyzed in this research as described below.

Universal Report is a code analysis and documentation software. Its goal is to analyze and generate a structured and well formatted documentation of a given program [15]. Since it is not an open source program, only the trial version can be downloaded at http://www.omegacomputer.com. Some highlights of the tool [16] are: runs with most of common programming languages (C, Fortran, Matlab, Basic, Java, Pascal, etc…) and object oriented codes; generates .txt, .rtf, .text, and .html reports with high standard format; provides a number of useful features for global text processing including searching, replacing, word/lines statistics, and spell checking; generates and includes graphics (routines calling graph) in the report; does not require any predefined comment tags or special markups in the source codes and handles GUI application documentation with automatic screenshots of forms for C++ Builder, Delphi, and Visual Basic cases.
**Doxygen** is an open source documentation system for C++, C, Java, Objective-C, Phyton, IDL, Fortran, VHDL, PHP, C#, and some extent D. It can generate output in HTML, LaTeX, RTF, PostScript, hyperlinked PDF, CHM, and Unix man pages [17]. One more benefit is this tool also support for XML output which can be intermediate files for another purpose.

**Javadoc** [18] is a tool for generating API documentation in HTML format from doc comments in source code. It can be obtained only as part of the Java 2 SDK, which is free to download. Kramer [19] explained a same definition as below.

“Javadoc is the standard API documentation generation for the Java programming language. It generates web pages from documentation comments in the source code.”

For many Java product libraries, Javadoc represents the most widely used form of documentation due to its simplicity and accurateness [20].

**DOC++** [21] is a documentation system for C, C++, IDL and Java generating both, TeX, output for high quality hardcopies and HTML output for sophisticated online browsing of documentation. The documentation is extracted directly from the source code. Some features of this tool are hierarchically structured documentation, automatic class graph generation (as Java applets for HTML), cross references, and high end formatting support including typesetting of equations.

**Haddock** [22] is a tool for automatically generating documentation from annotated Haskell source code. Currently the only fully supported the output format is HTML, although there is a partial implementation of a DocBook (SGML) back-end [23].

**Sandcastle** [24] is a documentation generator under Microsoft Public License that automatically produces MSDN style reference documentation out of reflection information of .NET assemblies and XML documentation comments found in the source code of these assemblies. Sandcastle can produce three types of format: HTML files, CHM (Help 1.0) format, and HxS (Help 2.0) format [25].

### 2.4 Software Design

Software design is a process through which requirements are translated into a representation of software. From a project management point of view, Pressman [8] explained two steps of software design: preliminary and detail design. Preliminary design is concerned with the transformation of requirements into data and software architecture. Detail design focuses on refinements to the architectural representation that lead to detailed data structure and algorithmic representations for software. Some document standards in software design, SDD and NASA Software Documentation Standards, are explained below.

**Software Design Document (SDD)** defines phases of a software development life cycle [26]. It was initially initiated by Department of Defense standard (DOD-STD-2167A) but currently it has been upgraded to IEEE 12207 for global standard development. Some UML diagrams are included in SDD. DoD [27] stated the purpose of this standard is to establish requirements to be applied during the acquisition, development, or support of software systems.

The SDD contains the following parts:

- **The Data Design** describes structures that reside within the software. Relationships and attributes between data objects dictate the choice of data structures.
- **The Architecture Design** uses information flow characteristics, and maps them into the program structure.
- **The Interface Design** describes internal and external programs interfaces as well as the design of human interface.
- **The Procedural Design** describes structured programming concepts using graphical, tabular, and textual notations.

Since the SDD is now fully supported by IEEE [28] it should be widely global recognized. Therefore, this standard will be used as result in this research.

**NASA Software Documentation Standard** is designed to support the documentation of all software developed for NASA [29]. Its goal is to provide a framework and model for recording the essential information needed throughout the development life cycle and maintenance of a software system.

This standard consists of four major sections: The Management Plan contains all planning and business aspects of a software project, including engineering and assurance planning; The Product Specification contains all technical engineering information, including engineering and assurance planning; The Assurance and Test Procedures contains all technical assurance information, including Test, Quality Assurance (QA), Verification and Validation (V&V); and The Management, Engineering, and Assurance Reports is the library and/or listing of all project reports.

## 3. METHODOLOGY

The proposed process will be separated into three main steps, as explained below.

- Firstly, it will extract all related information from the source code. This step will produce intermediate files result.
- Next is processing the intermediate files from the previous step. The result of this step will be saved in temporary database.
- The last step is documentation generation. Information from database will be arranged into SDD prepared format.

The relationship of source code, documentation template, and guideline is shown in Figure 1. This implementation is still under development.

![Figure 1. Relationship of code, template, and guideline](image-url)
4. RESULTS AND DISCUSSION

This research focuses on reverse engineering process to generate documentation from source code. To understand more about the generator, six tools which have almost similar features were evaluated and analyzed during this research. The analysis and comparison are described below.

Universal Report is not an open source software, which make users unable to learn the process or modify to their own purpose. There is no Word-like generated document from the output. It only run on Windows platform.

Opposite to Universal Report, Doxygen is an open source tool. However, the generated documentation does not follow any standard from DoD, NASA, or IEEE.

Javadoc is the official tool from Sun Microsystem to create Java documentation. The drawbacks are: it is only for Java, and the output is just for HTML. No standard documentation is supported.

The license of DOC++ is still unknown. It can be run on Windows or Linux. However, the documentation output is in HTML or LaTeX. No support for Word-like document. Further, the result is not a standard documentation.

Haddock is used for Haskell only. This language is rarely used in software development. The output is just in HTML and CHM.

Since Sandcastle is created by Microsoft, it supports Microsoft-based language such as C#, Visual Basic, and/.NET framework. There is no global standard for documentation, only HTML and CHM.

Table 1. Comparison of Documentation Generators

<table>
<thead>
<tr>
<th>Tools</th>
<th>Diagrams</th>
<th>Highlighting and Linking of Docs</th>
<th>Parameter Types Extracted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal Report*</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Doxygen</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Javadoc</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>DOC++</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Haddock</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Sandcastle</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>sddGen**</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>

*) trial version **) developed prototype version

From Table 1 above, it is shown that only Doxygen which is already generate diagrams in their output. DOC++ and Haddock have no support in generating diagram, while Sandcastle will support using additionally component.

Some generators have highlighting and linking on their documentation. User can browse the information on Universal Report, Doxygen, and Javadoc documents, while DOC++ and Haddock has one page of document only. Sandcastle has related the linking on its CHM output.

Universal Report and Haddock have parameter types extracted on its documentation. Others have no support on this aspect.

5. CONCLUSIONS

Legacy software systems require a different approach to software documentation than has traditionally been used. Re-documentation through reverse engineering is very important to understand large, evolving software systems. Reconstructing and effectively generating the design documentation of existing software systems is even more difficult than initial design [3]. 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.

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.

Therefore, reverse engineering is very significant to support software maintenance in order to maintain existing system. However, it is impossible to do everything automatically. Some parts still require human interaction to complete all information in the generated documentation [3].

This paper discusses a reverse engineering process implemented by a tool to generate an IEEE standard of software documentation. Currently, the prototype is already capable to produce basic content of the documentation.

5.1 Future Works

At the time of writing, we are still working at the parsing related information from the source code to intermediate files such as XML or XMI files. There are two alternatives in this part. First,
algorithms from some open source parser can be implemented into our prototype. Second, other tool is used to perform this part.

On the generation side, we are finalizing the documentation chapters. There are eight chapters prepared and the template can be modified on the database.

6. ACKNOWLEDGMENTS
The study is funded by eScience Fund under the vot no. 79277. The authors would like to thank the Universiti Teknologi Malaysia and Government of Malaysia for a researcher position in software engineering.

7. REFERENCES


