IBIMA Publishing
Communications of the IBIMA
http://www.ibimapublishing.com/journals/CIBIMA/cibima.html
Vol. 2011 (2011), Article ID 875759, 10 pages
DOI: 10.5171/2011.875759

An Evaluation on Software Redocumentation Approaches and Tools in Software Maintenance

N. Sugumaran[1] and S. Ibrahim[2]

[1]Universiti Tunku Abdul Rahman (UTAR), Malaysia

[2]University Technology Malaysia, Malaysia

Abstract

This paper describes an evaluation on software documentation generated using redocumentation approaches and tools. The evaluation is based on the selected Document Quality Attributes (DQA). Firstly, the paper presents an overview of the software redocumentation and the main components involved in the process for better understanding of the redocumentation. Consequently, several approaches and tools are highlighted in the context of aiding understanding to support the software evolution. Finally, the evaluation identifies some aspects of DQA that might benefit from refinement to better reflect the redocumentation approaches and tools capabilities that support the software maintenance.

Keywords: Software Redocumentation, Reverse Engineering, Software Maintenance, Program Understanding

---

Introduction

Software redocumentation is one of the approaches for aiding in program understanding to support the maintenance and evolution. According to (Elliot and James 1990), "Redocumentation is a creation or revision of a semantically equivalent representation within the same relative abstraction level". In other words, redocumenting code is a transformation from code (and other documents and stakeholder knowledge) into new or updated documentation about code. It becomes an aid for the recovery and recording of software comprehension. Since software comprehension is the most expensive part of software maintenance, redocumentation is the key to software maintainability. (K.Lano 1994) has specified three main goals for redocumentation process. Firstly, to create alternative views of the system to enhance understanding, for example (Benedusi et al. 1989) explained the generation of a hierarchical data flow or control flow diagram from the source code. Secondly, to improve the current documentation. Ideally, such documentation should have been produced during the development of the system ad updated as the system changed. This, unortunately, is not usually the case. Thirdly, to generate documentation for a newly modified program. This aimed at facilitating future maintenance work on the system preventive maintenance. Generating quality documentation through redocumentation process is important for program comprehension and software evolutions.

Reverse engineering and redocumentation output is thought to be the same. However, reverse engineering extract the design information which includes data flow diagram, control flow graphs (CFG), metrics and etc. The redocumentation tools

 Contact author: N. Sugumaran e–mail: sugumaran2@live.utm.my

emphasize on reformatting tools. Otherwise known as "pretty printers", reformatters make source code indentation, bolding, capitalization, etc. consistent thus making the source code more readable.

In this paper, we report the results gained from the empirical study on redocumentation approaches. Next, the approaches and tools are evaluated based on the Documentation Quality Attributes (DQA) for software documentation. The purpose of this evaluation is to assess the quality of the documentation produced via reverse engineering. However, in our context the evaluation is used specifically to evaluate the quality of document produced from redocumentation process.

The remainder of this paper is organized as follows. Section 2 describes the process of software redocumentation. It is followed by section 3 which offers the approaches and tools for redocumentation process. Section 4, provides evaluation of these current approaches and tools based on a set of criteria and finally section 5 concludes this paper.

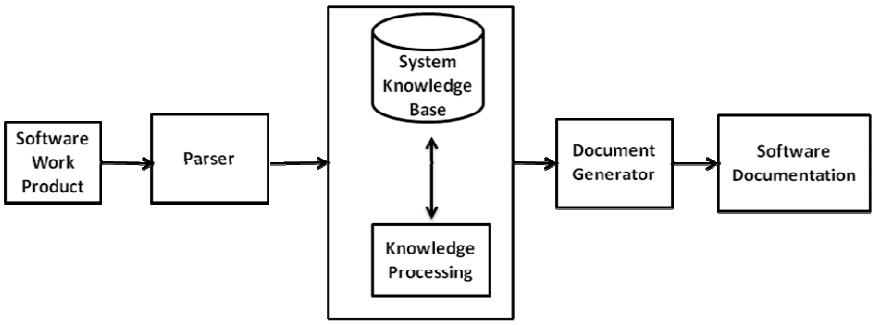

Fig1. Redocumentation Process

Software Redocumentation Process

Mainly, the redocumentation process is viewed as a knowledge rescue process as shown in Fig 1. One of the ways to implement the redocumentation process is using the reverse engineering. The redocumentation process consists of 5 main components namely; software work product, parser, system knowledge base, document generator and software documentation. The following describes these main components:

- **Source Materials (SM):**

SM can be source code, configuration files, build scripts or auxiliary artifacts. Auxiliary artifact can be a data gathering, manuals, job control and graphic user interface which help to understand the source code. The author (Shihong et al. 2005) explained the approach for redocumentation process and emphasized on the importance of the software work product to produce documentation for different type of information.

- **Parser:**

Parser is used to extract necessary information from SM and store them into the repository or system knowledge base. The importance of the parser is to return the relevant information, using specific technique. There are parsers which only focus on specific language such as tools created by (Simon 2002) and also focus on various types of programming language such as Universal Report specified by (Tadonki 2004).

- **System Knowledge Based:**

According to (Shirabad 2003), knowledge based is a collection of simple fact and general rules representing some universe of discourse. The purpose of this component is to store extracted information from the SM in

order to describe the context of information. This component becomes the heart of the system to allow the tools accessing the required information. In other words, all parts of the model are able to access and organizes the information from the knowledge based. There are researches which only focus on the knowledge based to make sure that the knowledge retrieved supports the best for program understanding. Knowledge based works closely with the knowledge processing to recover and revealed the various relationships implicit in the SM (Inc 2009). The processed knowledge is represented in different types of forms such as; data modeling and procedure or function. As an example, (Holger M. Kienle 2008) mentioned that Rigi uses the RSF file as a repository and presents the knowledge presentation in procedural.

- **Document Generator:**

Knowledge produced from the knowledge based is used by the Document Generator to search and select the knowledge that is not presented explicitly. Generally, Document Generator is used as a management tool to identify needed component to generate new software work product.

- **Software Documentation**:

Finally, the processed knowledge is presented in various form of documenting to the user (developer, maintainer, software engineer or end user) such as; directed graph, annotation, visualization, metrics or in documentation. The software components are extracted including modules, procedures, classes, subclasses, interface, control flow, composition and enslavement among the component. The software documentation can be categorized into Textual and Graphics. Textual documentation ranges from inline style which is written informally, to personalized views which are dynamically generated from a document database. HTML or XML are considered as a more flexible form of textual documentation which allows automating the indexing and creating hyperlink between document partitions or sections. The least mature type of graphical documentation is a static image, which may use non-standard representations of software artifacts and relationships. The most advanced graphical documents are editable by the user and are better enabling them to create customized representations of the subject system. Software visualization technique is used to present graphical documentation which helps the maintainer to understand.

Classification of Redocumentation Approaches

This section will present recent state-of-art approaches and tools that have produced solution for redocumentation process. Most of the approaches and tools are developed for reverse engineering which are generally compared to the redocumentation process. However, we have identified some significant approaches and tools which can contribute to further development of the quality documentation from the redocumentation process. The following redocumentation process is classified based on approaches and tools.

Approaches

The following are the approaches used in redocumentation process to create various types of documentation.

- **XML Based Approach:**

XML based approach is one of the common redocumentation approaches used to generate the documentation. It contains structured information that extracts the content and the meaning of the documentation. XML is reassembled from HTML to make it more useful for program documentation. (Jochen et al. 2001) stated that by using XML the technical writer or software engineer can create their own format, such as <CONSTRANT>, <TASK>, <FILE>, <VARIABLE> and<FUNCTION>.This feature helps to identify the implicit semantic of the document. 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 to make sure that the data can be exchanged between different software systems. (Jochen et al. 2001) used the XML as

a knowledge base to redocument the program and integrate every level in redocumentation process to produce high quality documents(Jochen et al. 2001). In the first level, SM captured the data from source code and blended it with other resources (manual, programmer, and software documents) to have more data sources. Following that, commercial or specific parser is used to extract the structure in the extraction process. In level 2, captured structure or data from various SM are merged into one repository to facilitate knowledge processing. One of the important activities in the level 2 is to uncover important information hidden in the gathered data (Jochen et al. 2001). Finally, in level 3, generated documentation can be viewed in both textual and graphical representation. The output produced can be used back in the next iteration of data gathering phase to refine the information contained in the repository.

- **Model Oriented Redocumentation:**

(Feng and Hongji 2007) have proposed the Model Oriented Redocumentation approach to produce models by using Model Driven Engineering (MDE) technique from existing systems and documentations which are generated based on the models. The main objective of the MDE is to raise the abstraction level in program specification and increase the automation in program development. The MDE concept is suited the redocumentation process, specifically, to produce higher level of abstraction in the final documentation. Basically, the MDE concept is merged with the Model Driven Architecture (MDA) in general and fastened with the Technological Spaces(TS) (Ivan Kurtev 2002). The first step is to transform the legacy system into formal models. These formal models are written using a formal language and transform into TSs. Are Generated TSs are stored in repository and produced documentation in a uniform way. To support the framework, the tool called Maintainer Integration Platform(MIP) is developed and supported by Wide Spectrum Language (WSL) to present high and low level abstraction(Feng and Hongji 2007).

- **Incremental Redocumentation:**

One of the common issues in maintaining the system is to record the changes requested by customer or user to occur in the source code. Often, the program comprehension is not stored and integrated in a single location which is done by the programming team. The team, by having code ownership will overwork some of the programmers and leave others unutilized. (Vaclav 1997, Rajlich 2000) have used the Incremental Redocumentation approach to rebuild the documentation incrementally after the changes are done by the programmer. As a first step, the change request will be collected, which is normally received from customers. Next, the collected change request will be analyzed and assigned to the programmer to implement the change request. The programmer will do the changes accordingly and confirm the correctness of the system. Finally, by using the PAS tool the program comprehension achieved during the change request implementation is recorded. According to (Rajlich 2000), the PAS also helps to store the information either top-down or bottom-up, complete or partial and also whether confirmed or tentative. The advantage of PAS is using the hypertext in the style of World Wide Web in which there is no any limit for the number of partitions or their contents. The main partition is domain partition which is important to understand the application domain.

- **Island Grammar Approach:**

Grammar definition language SDF is used as a parser to define the island grammar(Verhoeven 2000). (Mark G. J. van den Brand 1996) mentioned that SDF will return parse tree in Java object which is encoded in *aterm* format. The result can be written in a repository which can be joined, queried and used in the process of document generation. The filtering data process starts during the analyses phase. The output can be abstracted in different layers depending on the documentation requirement. (Arie van and Tobias 1999) used Cobol system to generate hierarchy associated with documentation requirements. On the other

hand, (Moonen 2001) explained, in details, the supporting tool for island grammar approach called Mangrove.

- **DocLike Modularized Graph(DMG):**

Based on the research study by (Shahida Sulaiman 2003), DocLike Modularized Graph(DMG) provides the template to present the artifacts extracted as a software design documentation. The user is able to update basic description related to each specific design using Description Panel. The Content Panel focuses on the representation of the modules and the associated graph is displayed accordingly in Graph Panel. Description Panel is used to describe the associated section manually. The DocLike Viewer Prototype Tool system can be expressed by using the redocumentation framework. As an input, they used C source code and the existing parser provided by Rigi. In the later section the author is going to discuss this tool in details. DocLike Viewer uses the existing storage provided by Rigi and filters the data by selecting only the required information to be visualized in DocLike Viewer.

Tools

The following are the tools developed to redocument the source code to generate documentation.

- **Rigi:**

Rigi uses a reverse engineering approach to extract the artifacts from the source code, organizes them into medium level abstractions, and shows the output graphically(Müller 1996). According to (Kenny et al. 1995), there are three types of methodology used in Rigi, which are rigireverse, rigiserver and rigiedit. Rigireverse is a parser that supports C and COBOL language and also a parser for Latex used to analyze the documentation. The main function of Rigiserver is to store the information extracted from the source code in the repository. The rigiedit is an interactive, window-oriented graph editor to manipulate the program representations. In Rigi, the first phase involves parsing the source code and storing the extracted artifacts in the repository in Rigi Standard Format (RSF) file. There are two types of Rigi Standard Format (RSF) files. First, is an unstructured Rigi Standard Format (RSF) file which may contain duplicate tuples. The other is structured Rigi Standard Format (RSF) files used for displaying graphical architecture. Files contain information about the nodes (e.g., functions, variables, data structures, etc.) and arcs (e.g., function to function calls, or function to the variable calls). There is a tool called sortrsf which converts an unstructured file into a structured file. Once a C programming input file is being parsed by the Rigi parser, a RSF file is produced and it is sent to Rigi editor for further processing. Rigi editor is a graph editor which uses windows based interface. It is developed using a TCL scripting language. Rigi editor is used for architecture display, traverse, and modify the graphical model. (Margaret-Anne et al. 1997) stated that the second phase involves cluster functions into subsystems according to the rules and principles of software modularity to generate multiple views called Simple Hierarchical Multi-Perspective view, layered hierarchies for higher level abstractions. Finally, the Rigi Editor assists the maintainers in understanding the structure of large, integrated, evolving software systems.

- **Scribble:**

Scribble focuses on generating library documentation, user guides and tutorials for PLT scheme(Matthew et al. 2009). It combines all of these threads producing a scribble language or tool that spans and integrates document categories. Scribble was built using PLT Scheme technology, which is built based on academic and also on practical tradition. It is suitable to use for tasks related to application development, including GUIs and web services and supports the creation of new programming languages through a rich expressive syntax system. The features in PLT schemes help to develop Scribble system more easily and Scribble is just an extension of the PLT schema. So, the main input and the parser in the documentation process is the PLT Scheme itself. Central PLanet package repository is used to store the libraries. The final output is produced in

HTML form which consists of libraries with the guides and tutorials. In fundamentals, the basic concept is to construct representations of documents using scheme functions and macros.

- **Haddock Tool for Haskell Documentation:**

Haddock Tool is a tool for generating documentation from the source code automatically. Haddock, primarily, focuses on generating the library documentation from the Haskell source code(Simon 2002).

- **Universal Report**

(Tadonki 2004) has presented the Universal Report, a tool used to analyze the source code and documents the software system. The main objective of this tool is to analyse and generate the structured and well formatted document of various types of languages such as C++. Visual Basic, Ada, Cobol, Fortran, Java, Assembler,Perl, PHP, Python and many others. It uses pattern matching algorithm and compilation techniques to extract the information from the source code and generate the documentation in HTML, Latex and plain text files(Tadonki 2004). The HTML output has a lot of features including searching the script for text over the entire documentation, an online commenting and annotating system, a dynamic flowchart, routine call graph, screenshots from form files, detailed analysis and dynamic composition of each routine. In addition, the Universal Report can also read the database files and generate the detailed report of the structure and elements such as table, fields and reports. However, the features in Universal Report tool emphasize redocumentation of source code in the implementation level only. It doesn't focus on higher abstraction level such as design or specification level.

Comparative Evaluation

DQA are the simplified attributes from the assessment of reverse engineering techniques and tools (Shihong and Scott 2003, Scott 1998). In this paper, the criteria have been restructured to evaluate the document produced from redocumentation process. The evaluation criteria consist of number of criteria namely; efficiency, format (textual and graphical) and granularity.

- **Efficiency**: Efficiency refers to the level of direct support the documentation provides to the software engineer engaged in a program understanding task.

- **Format:** Format refers to the type of document produced either in textual or graphic. Textual is the documentation from inline prose in an informal manner to personalized view. Graphic format presents the software artifacts and relationships in graphical form.

- **Granularity:** Refers tothe level of abstraction which describes the documentation.

Table 1: Summary of Criteria and Level in DQA

Quality Level	Format		Granularity	Efficiency
	Text	Graphic		
Level 1	• Explained in low level functionality. • No standard format and style. • Requires developer experience.	• Static graph as a hardcopy-like image can be in format such as GIF or PDF and informal. • Read only graphic.	• Documentation at level of source code. • Comment on algorithm and source code.	• Generated manually in textual form.
Level 2	• Standard documentation and includes also the developer's own format • Using a standard template documentation	• Standard representation in the graphical form. • Using template such as UML.	• One level above design patterns. • Helps developer to understand high level rational. .	• Semi-automatic using reverse engineering. • Static and reflects the system changes at the time of generation.
Level 3	• Hyperlinked add indirection to the text. • Can be text, graphic or multimedia commentary.	• Animated graphical documentation in visual manner. • Users have little interaction.	• High level design software architecture. • Able to make changes based on system architecture.	• Dynamic and semi-automatically reflect the changes as long as the developer direct the tools.
Level 4	• Contextual Documentation using tools support. • Enhances the information on the context.	• Interactive and permits the user to navigate from one node to next level node. • Can chase down the artifacts and relationships. • Better response to user feedback.	• Captures the system requirements from the point of view of the user. • Multiple level of abstraction.	• Automated and static but no need for developer involvement.
Level 5	• Personalized document for the reader • Multiple view of the system.	• Editable graphical documentation. • Able to add new nodes and can be saved in the repository (if available).	• Product line documentation • Captures the commonalities and variability in the product. • Defines the domain knowledge.	• Fully automatic and dynamic. • Produces the documentation on demand.

Table 2: Comparing Redocumentation Approaches

Approaches	Benchmarks			
	Format		Granularity	Efficiency
	Text	Graphic		
XML Based Approach	L3	L2	L2	L2
Model Oriented Approach	L2	L3	L3	L2
Incremental Approach	L2	-	L2	L3
Island Grammar Approach	L3	L1	L2	L2
DocLike Modularized Graph Approach	L3	L3	L2	L2

Table 3: Comparing Redocumentation Tools

Tools	Benchmarks			
	Format		Granularity	Efficiency
	Text	Graphic		
Rigi	L2	L4	L2	L2
Haddock Tool	L3	-	L2	L4
Scribble Tool	L3	L1	L2	L4
Universal Report	L3	L1	L1	L4

Each of the criteria measured based on 5 levels. Level 1 indicates the lowest level and level 5 is the highest level of the document quality. The summary of each criteria and the maturity level are shown in Table 1. Based on the evaluation criteria above, the approaches and tools have been evaluated. The evaluation results are shown in Table 2 for approaches and Table 3 for tools. Table 2 and Table 3 exemplify the quality level achieved by the approaches and tools to generate the documentation. The result shows that the strength of existing approaches and tools emphasize on the different levels of the redocumentation process. The example of model oriented approach shows the highest maturity level for the granularity criteria (L3). The Rigi tools criteria shows the highest level for the graphics (L4) and Scribble tool shows the highest for efficiency (L4). However, XML approaches shows moderate level and balance for each except for efficiency. The analysis in Table 1 and Table 2 shows the quality of the documents produced depending on the research emphasis on the specific component in the redocumentation process. Like Rigi, the emphasis is on visualization (output), Island grammar approach for extracting syntatic structure of the code (parser) and Model Oriented for software evolution (knowledge based). Generally, XML based approach is very common approach that is used nowadays. The advantage of this approach, as compared to other approaches, is that it can create different types of view according to the user needs. However, the abstraction is in medium level and it will be difficult to show semantically the related knowledge of the same domain. Model Oriented approach is useful for system evolutionary because it allows showing he exact characteristics as the original system or the domain level. The maintainer has a better view and understanding to handle maintenance task. Island grammar approach is used at the parser level and it contributes to speeding up the extraction process and concentrates on the data analysis for the documentation. Based on the tools the granularity is still

considered low and medium level. Most of the tools are developed only until the level of the view the software architecture but not on the requirement level of the system.

The text format documentation produced by all the approaches and tools still are in the medium level in which most of the documents created are in the hyperlink such as XML based approach. On graphic, Rigi is one of the tools which helps to view the graphic form of the software component and allows navigating to certain extends. However, other approaches and tools emphasize only on viewing the graphics but do not allow navigating using graphic. In term of efficiency, the haddock and scribble tools are able to automate the redocumentation process compared to other approaches and tools. However the automation process is easy because it involves the low level abstraction.

Conclusion

This paper aimed to provide general overview and compare the progress in software redocumentation. The problem with most of these approaches and tools for redocumentation is that the granularity level is low and it limits the understanding on the domain knowledge. The maintainer needs a better understanding of the semantic relationships among the component from real world domain point of view, especially, if the maintainer is the new member in the program domain. The model oriented approach tries to solve this problem, however, the efficiency level is low and not able to search the information as needed. The main issues that is needed to be addressed here is that, the software documentation produced from redocumentation process needs to emphasize on the importance of explicit documenting domain knowledge to improve the program comprehension in software maintenance and to be presented in standard documentation.

Acknowledgments

This research is supported by Ministry of Science and Technology and Innovation (MOSTI), Malaysia and Universiti Teknologi Malaysia (UTM).

References

Benedusi, P., Cimitile, A. & De Carlini, U. (1989). “A Reverse Engineering Methodology to Reconstruct Hierarchical Data Flow Diagrams for Software Maintenance,” Software Maintenance, 1989., Proceedings., Conference on, ISBN:0-8186-1965-1, 06 August 2002, Miami,FL,USA 180-189.

Chikofsky, E. J. & Cross II, J. H. (1990). “Reverse Engineering and Design Recovery: A Taxonomy,” IEEE Software, 7(1), 13-17.

Chen, F. & Yang, H. (2007). “Model Oriented Evolutionary Redocumentation,” 31st Annual International Computer Software and Applications Conference, 2007. COMPSAC 2007, ISBN:0730-3157, Beijing, 543-548.

Flatt, M., Brazilay, E. & Findler, R. B. (2009). “Scribble: Closing the Book on Ad Hoc Documentation Tools,” Proceedings of the 14th ACM SIGPLAN International Conference on Functional Programming, ISBN:978-1-60558-332-7, New York, NY, USA, 109-120.

Hartmann, J., Huang, S. & Tilley, S. (2001). “Documenting Software Systems with Views II: an Integrated Approach Based on XML,” Proceedings of the 19th annual international conference on Computer documentation, ISBN:1-58113-295-6, New York, NY, USA, 237 - 246.

Huang, S. & Tilley, S. (2003). “Towards a Documentation Maturity Model,” 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05),ISBN:1-58113-696-X, New York, NY, USA, 93 - 99.

Huang, S., Tilley, S., Van Hilst, M. & Distante, D. (2005). “Adoption-Centric Software Maintenance Process Improvement via Information Integration,” Software Technology and Engineering Practice, 2005. 13th IEEE International Workshop on, ISBN:0-7695-2639-X, IEEE 25-34.

Inc, B. S. (2009). “Reverse Engineering,” [Online], Business Software Inc., [March 02,2010], http://bus-software.com/re.htm

Kienle, H. M. & Muller, H. A. (2008). “The Rigi Reverse Engineering Environment,” Proceedings of the International Workshop on Advanced Software Development Tools and Techniques, Paphos, Cyprus, July 8, 2008.

Kurtev, I., Bezivin, J. & Aksit, M. (2002). “Technological Spaces: an Initial Appraisal,” CoopIS, DOA'2002 Federated Conferences, Industrial track,Irvine

Lano, K., Haughton, H. (1994). “Reverse Engineering and Software Maintenance : A Practical Approach," Mc-Graw Hill, London.

Marlow, S. (2002). “Haddock, a Haskell Documentation Tool,” Proceedings of the 2002 ACM SIGPLAN workshop on Haskell, ISBN:1-58113-605-6, New York, NY, USA, 78 - 89.

Moonen, L. (2001). “Generating Robust Parsers Using Island Grammars,” Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01, ISBN:0-7695-1303-4, Washington, DC, USA, 13-22.

Müller, H. A. (1996). “Rigi User's Manual,” [Online], University of Victoria, [July 18,2007], http://www.rigi.csc.uvic.ca/downloads/rigi/doc/user.html

Rajlich, V. (1997). “Incremental Redocumentation with Hypertext,” Proceedings of the 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97), ISBN: 0-8186-7892-5, IEEE Computer Society Press, 68 - 72.

Rajlich, V. (2000). “Incremental Redocumentation Using the Web,” Software, IEEE, 17(5), 102-106.

Shirabad, J. S. (2003). “Supporting Software Maintenance by Mining Software Update Records,” 17th IEEE International Conference on Software Maintenance (ICSM'01), ISBN: 0-7695-1189-9, Washington, DC, USA, 22.

Storey, M-A. D., Wong, K. & Muller, H. A. (1997). “Rigi: a Visualization Environment for Reverse Engineering,” Proceedings of the 19th international conference on Software engineering, ISBN:0-89791-914-9, New York, NY, USA, 606 - 607.

Sulaiman, S., Idris, N. B., Sahibuddin, S. & Sulaiman, S. (2003). “Re-documenting, Visualizing and Understanding Software System Using DocLike Viewer,” Proceedings of the 10th Asia-Pacific Software Engineering Conference (APSEC’03),ISBN:0-7695-2011-1, Washington, DC, USA, 154 - 163.

Tadonki, C. (2004). “Universal Report: a Generic Reverse Engineering Tool,” 12th IEEE International Workshop on Program Comprehension (IWPC'04),ISBN: 0-7695-2149-5, 266-267.

Tilley, S. (1998). “A Reverse-Engineering Environment Framework,” [Online], Carnegie Mellon Software Engineering Institute, [25/11/2010], www.sei.cmu.edu/reports

Van Den Brand, M., Klint, P. & Verhoef, C. (1996). "Core Technologies for System Renovation," [Online], Springer-Verlag, [May 12,2010], http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.3744

Van Deursen, A. & Kuipers, T. (1999). “Building Documentation Generators,” Proceedings of the IEEE International Conference on Software Maintenance, ISBN:0-7695-0016-1, 06 August 2002, Oxford, 40 - 49.

Verhoeven, E-J (2000). “Cobol Island Grammer in SDF,” [Online], University of Armsterdem, [15/2/2010], http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.12.8491

Wong, K., Tilley, S. R., Muller, H. A. & Storey, M-A. D. (1995). “Structural Redocumentation: A Case Study,” 12(1), 46-54.