

An Assessment of Reverse Engineering Capabilities of UML CASE Tools

Hafeez Osman

Leiden Institute of Advanced Computer Science
Leiden University
Leiden, the Netherlands
hosman@liacs.nl

Michel R.V. Chaudron

Leiden Institute of Advanced Computer Science
Leiden University
Leiden, the Netherlands
chaudron@liacs.nl

Abstract— This paper focuses on Computer-aided Software Engineering (CASE) tools that offer functionality for reverse engineering into Unified Modeling Language (UML) models. Such tools can be used for design recovery or round-trip engineering. For these purposes, the quality and correctness of the reverse engineering capability of these tools is of key importance: Do the tools completely reconstruct the UML diagrams? Are the reverse engineering results correct? What kind of information is presented in the result? Based on these questions, we compare seven UML CASE tools (six commercial tools and one open source tool). We evaluate i) the types of inputs that these tools can handle, ii) the types of diagrams that can be reconstructed, iii) the quality of resulting diagrams.

Keywords—Reverse Engineering; UML; Round-trip Engineering, CASE tools

I. INTRODUCTION

The Unified Modeling Language (UML) has emerged as the de facto standard for graphically representing the design of object-oriented software systems [7]. While UML diagrams are created in forward design, these diagrams are poorly maintained. Maintaining correspondence (between design and implementation) is particularly challenging because over time an implementation tends to evolve considerably from its initial design [14]. Design models produced during the design phase are often forgotten during the implementation phase—under time pressure usually—and thus present major discrepancies with their actual implementation frequently [3]. Timothy C. Lethbridge et al [15] confirm the widely held belief that software engineers typically do not update documentation as timely or completely as software process personnel and managers advocate. At the same time, software engineers working in software maintenance express a need for better documentation. Tools support during maintenance, re-engineering or re-architecting activities has become important to decrease the time software personnel spend on manual source code analysis and help to focus attention on important program understanding issues [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 other form or at higher level of abstraction [12].

For this study, our motivation is to find out to what extent the CASE tools are able to reverse engineer UML diagrams out of source code. We want to know the strengths and weaknesses of the evaluated reverse engineering tools. In order to find the answers, we examined and compared the reverse engineering capabilities provided by the CASE tools. Seven tools have been selected in this paper, namely Visual Paradigm, Rational Software Architect, StarUML, Altova UModel, MyEclipse, Enterprise Architect and MagicDraw. To understand how the tools analyze class diagram, we have conducted three experiments. The first experiment tried to find out whether tools could be used to do round-trip engineering. In this experiment, we want to know whether the forward code generation capability is compatible with reverse engineering capability. In other words: whether the code that a tool generates from a UML model can be used by the reverse engineering capability to reconstruct to original UML model.

For the second experiment, we tested the tools in identifying class relationship (association, aggregation and composition) based on the code stated in [3]. This experiment is done to find out whether the tools are capable of identifying class relationships. In the third experiment, we tested the tools of reverse engineering class diagram.

The paper is structured as follows. Section II briefly describes the examined tools and properties used in this evaluation. Section III describes the case study and Section IV explains the approach on how we conducted the experiment. Section V presents our results and findings. Our evaluation will be discussed in Section VI. We suggest future work and present the differences of this paper and other related research in Section VII. This is followed by our conclusion in Section VIII.

II. EXAMINED TOOLS AND PROPERTIES

A. Examined Tools

We have chosen several well-known CASE tools based on their capability in performing forward and reverse engineering in Java, capability in exporting XML Metadata Interchange (XMI) files and supporting UML design. The evaluated CASE tools are listed in TABLE I.

The tool we use for extracting metrics from the output created by the CASE tools, is the SDMetrics tool. This tool can handle many different versions of XMI (.xml, .xmi). SDMetrics [11] version 2.11 (academic license) is used for this purpose.

TABLE I. LIST OF EVALUATED CASE TOOLS

No	CASE Tools	Information	License type
1	Visual Paradigm 8.1	http://www.visual-paradigm.com/	Evaluation
2	MagicDraw 17.0	http://www.magicdraw.com/	Evaluation (academic)
3	Altova Umodel 2011	http://www.altova.com/	Evaluation
4	Enterprise Architect 8.0	http://www.sparxsystems.com.au	Evaluation
5	Rational Software Architect 8.0.1	http://www-142.ibm.com/software/products/my/en/swarchitect-websphere	Evaluation
6	MyEclipse 8.6	http://www.myeclipseide.com/	Evaluation (academic)
7	StarUML 5	http://staruml.sourceforge.net/	Open Source

B. Examined Properties

1) *Reverse Engineering Capability*: The reverse engineering tools capabilities are evaluated from the following perspectives :

a) *UML Diagrams*: Three UML diagram types are selected for our evaluation. First, we evaluate the package diagram. The package diagram is used to group the classes together into high-level unit [1]. Second, we study the class diagram. Class diagrams describe the type of objects in the system and the various kinds of static relationships that exist among them [1]. Third, we evaluate the sequence diagram. Sequence diagrams describe an interaction between objects and actors of the system by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines [2]. Reverse-engineered sequence diagrams can be created through static or dynamic analysis [5]. Only static analysis is used to generate the sequence diagram in this experiment. We analyze all three diagrams by evaluating the process of generating the diagrams and the output in term of completeness and representation.

b) *Supported Programming Language(s)*: Several common programming languages are selected to study the capability of the tools in reverse engineer source code. The selected programming languages are PHP5, C++, Java, C#, Delphi, Python and Visual Basic (V.B.).

c) *Additional Types of Input formats*: The supported input-types for reverse engineering UML diagrams (in addition to source code; e.g. binaries).

2) *Class Diagram Properties*: There are two types of basic information about a class that are important for this evaluation. The basic information is the following:

a) Class Attributes and Methods

- Number of attributes: We evaluate the tools' ability to reconstruct all attributes including the type of attribute (public, private, protected) defined in the source code.
- Number of operations: We evaluate the tools' ability to reconstruct all methods (of all: public, private, protected, constructor) defined in the source code. In addition, we assess whether the tools can distinguish public from private or protected methods.
- Getters and Setters: We evaluate the tools' ability to identify the difference between getters and setters and other operations.

b) Class Relationship

- Number and type of Relationship: We evaluate whether the tools reconstruct all relations between classes.

III. CASE STUDY

This section describes the case study used for this paper. The case study is a fully functional simulation system that has class design and implementation source code. The class design was made using forward design. The case study is an ATM simulation example developed by the Department of Mathematics and Computer Science, Gordon College [4]. This simple simulation system is used to show the overall process of UML usage in analysis, design and implementation phase. The complete software documents based on UML were provided that consist of 22 design classes.

IV. APPROACH

This section explains the approaches that were used to evaluate the tools. The evaluation is divided into two parts that are: Round Trip Capability and Reconstruction of UML Diagram types.

A. Round Trip Capability

To assess how well the tools can be used for round-trip engineering, we conducted round trip experiments. The experiment is done to compare the difference between the forward design and the reverse engineering design. The experiment begins by creating a sample UML Design (class diagram) that consists of basic information such as attributes and methods (private, protected, public) and class relationship information (such as association, composition and aggregation). Then, we performed forward engineering to produce the source code. Based on forward-engineered-source code, we performed reverse engineering to get the UML Design'. Both UML Design and UML Design' were then compared. This experiment is illustrated in Figure 1.

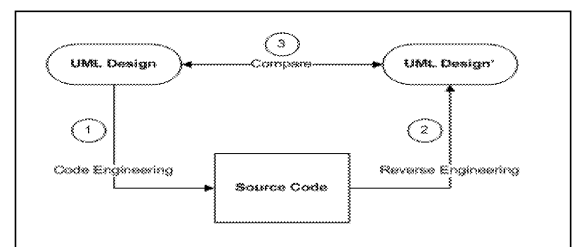


Figure 1. Roundtrip Engineering Experiment

B. Reconstruction of UML Diagram types (Package/class/sequence)

To assess the quality of the reverse engineering of UML diagrams, we conduct the following experiments:

1) *UML Diagram Reconstruction capability* : Information about the tools' support for diagram types is gathered from the tools' manual. The results of each tested diagrams are evaluated based on a four-level scale. The scale explanations are the following:

- “++” - this scale is set if the tools are able to completely reverse the specified diagrams
- “+” - this scale is set if the tools are able to reverse engineer the specified diagram but there is/are minor information that cannot be analyzed such as aggregation and composition relationship for a class diagram and dependency relationship for a package diagram.
- “o” - this scale is set if the tools are able to reverse engineer the specified diagram but present minimal or basic information about the diagram, for an example, the tool is capable in presenting the class name with attribute and method only. No relationship is presented. Another example is the tools need user intervention to generate the sequence diagram.
- “-” - this scale is set if the tool are unable to reverse the specific diagram.

The tool manuals are also used to collect (the) information about the supported programming language and supported type of reverse engineering sources.

2) *Detection of Aggregation, Association and Composition relationship*: This experiment aims to test the reverse engineering capability on various types of class relationship based on code defines in [3]. We create different version of source code based on relationship types defined in [3] and use the tools reverse engineering functionality to generate the class diagram. Then, the results were observed.

3) *Correctness and Completeness (CnC) of Reconstructed UML Diagram*: This experiment aims to test the completeness and the correctness of the result of reconstructed UML diagram based on the CASE tools reverse engineering capability. We begin this task by capturing the expected result derived by the provided case study design document and implementation source code. The expected result for all basic information and relationship is gathered by manual and by using software metrics tool. This evaluation is divided into two sections as described below:

a) *CnC of Basic Class Information*: A new separated project is created for each tool. All possible options in the reverse engineering function were tested to get the best result. The best reverse engineered class diagram from each tool is exported to XMI or XML file format. Then, the software metrics and evaluation results were recorded.

b) *CnC of Reconstruction of Class Relationship*: A new project is separately created from the tasks mentioned above

and all possible options in reverse engineering function were tested to have the best view of class relationship. Then, manually, the relationships constructed by each tools were evaluated and compared with the expected result. The evaluations were then recorded.

V. RESULT AND FINDINGS

In this section, we present the result of our evaluation and findings. The results are divided into two subsections' which are Reverse Engineering Capability and Class Diagram Properties. Complete evaluation results and test diagrams are shown in [16].

A. Reverse Engineering Capability

The assessment was done by using a four-level scale to evaluate the tools in reverse engineering task. The four-level scale are “++” excellent, “+” good, “o” minimal and “-” not capable.

TABLE II. SUPPORTED UML DIAGRAM FOR REVERSE ENGINEERING

No	Tools	UML Diagram		
		Package	Class	Sequence
1	Visual Paradigm for UML Enterprise Edition	+	+	o
2	UModel Altova 2011	+	+	o
3	My Eclipse 8.6	o	+	-
4	Star UML 5	o	+	-
5	Magic Draw 17.0	o	+	-
6	Rational Software Architect v8.0.1	o	+	o
7	Enterprise Architect 8	o	+	o

Most of the tools are capable of generating Package Diagrams. Generally, all the evaluated tools are good at automatically generating class diagrams based on the source code. They also give the option to the user to separately generate the class diagram using the “drag and drop” function.

There are only four tools in our evaluation that have the capability of reverse engineering sequence diagrams. To generate the sequence diagram, the user is required to choose a method in a class. The support of different tools for reverse engineering different UML Diagram is given in TABLE II.

The supported programming languages results are presented in TABLE III. It shows that the Enterprise Architect is able to reverse engineer all the programming language listed in this evaluation. We also found that all evaluated tools are able to reverse engineer source codes in Java.

Overall, the evaluated tools are able to use source code files such as .java, .cpp and .cs. They also offers an option to the user to specify the source directory where the source code is located and automatically determine the source code file from the directory.

Visual Paradigm, Altova and Enterprise Architect are capable of decompiling a java bytecode (.class), dynamic link library (.dll), execution file (.exe) and java archive (.jar). The

tools then generate class information that enables the users to construct a class diagram.

TABLE III. SUPPORTED PROGRAMMING LANGUAGE

No	Tools	Programming Language						
		PHP 5	C++	Java	Delphi	Python	V.B	C#
1	Visual Paradigm	Y	Y	Y	N	Y	N	N
2	UModel Altova	N	N	Y	N	N	Y	Y
3	My Eclipse	N	N	Y	N	N	N	N
4	StarUML	N	Y	Y	N	N	N	Y
5	MagicDraw	N	Y	Y	N	N	N	Y
6	Rational Software Architect	N	Y	Y	N	N	Y	Y
7	Enterprise Architect	Y	Y	Y	Y	Y	Y	Y

B. Class Diagram Properties

This subsection presents the assessment for the class diagram properties. The result is divided into three subsections which are Roundtrip Findings, Class Relationship Test, and Class Diagram Correctness and Completeness.

1) *Round Trip Findings*: We found that the CASE tools can easily extract all the listed class attributes and operations. However, the results vary for class relationships. All the tools show the same result that association and inheritance were correctly presented but aggregation and composition are visualized as association. Rational Software Architect shows the association, aggregation and composition as dependency. Most of the tools declare the association, aggregation and composition in the forward engineering code as link declaration as stated in [6]. This is the reason why the tools were not able to differentiate the class relationship.

2) *Class Relationship Test*: Based on the source code that was presented in [3], we found that all the evaluated tools are unable to detect the required class relationship. Visual Paradigm, Altova UModel, StarUML, MyEclipse, MagicDraw and Enterprise Architect give the same result that all association relationships were unable to be generated while the aggregation and composition relationship was presented as association relationship. On the other hand, Rational Software Architect shows different result by generating all the class relationship as dependency relationship.

3) *Class Diagram Correctness and Completeness*: Class Diagram Correctness and Completeness evaluation is divided into two parts; Class Attributes and Methods and Class Relationship.

a) *Class Attributes and Methods*: This evaluation present the capability of the CASE tools in identifying and differentiate class attributes and methods or operations.

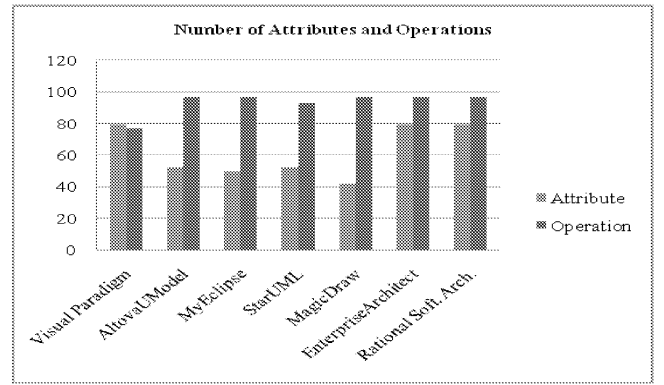


Figure 2. Number of Attributes and Operations

Number of Attribute (NA): We expected the tools to extract 79 attributes (NA) from the case study. Visual Paradigm, Enterprise Architect and Rational Software Architect successfully extracted all the attributes as shown in Figure 2. Other tools like Altova UModel, MyEclipse, StarUML and MagicDraw show some weakness in this operation where they were unable to extract all the expected attributes.

Number of Operations (NO): The number of operations (NO) expected to be extracted by the tools is 91. However, most of the tools found more. The additional operations come from the operations that are derived from the superclass which is also redeclared inside the inherited classes or subclasses. This shows that the reverse engineering tools did not check for the usage of superclass operations. StarUML did not completely extract all operations because it is unable to extract 4 constructors of 4 classes. Visual Paradigm was only capable to extract 77 operations. In addition, we also assess the Number of Public Operations (NPO). The expected NPO of the case study is 83. Overall, all the evaluated tools are capable of identifying the NPO except Visual Paradigm and StarUML. Visual Paradigm identified 63 NPO and StarUML identified 79 NPO. This is a consequence of the weakness of these tool in extracting operations.

Number of Setters (NS): All the tools are capable to identify all expected NS.

Number of Getters (NG): All 32 expected getters were successfully identified by six of the evaluated tools. Only Visual Paradigm did not identify all the getters.

b) Class Relationship

From the Round Trip Experiment, and Class Relationship test, we found that the evaluated tools can only identify the association and inheritance (generalization) relationship. In the code, the proposed guidelines that enable recognizing different relation-types describe in [3] were not used. Hence, we further evaluate the relationship of the class by evaluating the tool capability in extracting association and generalization relationship. We have extracted all the link declarations in our case study and use it as the expected result.

TABLE IV. RELATIONSHIP CORRECTNESS

No	Tool	Total Relationship Count	Association		Inheritance	
			Association Relationship count	Association Correctness (%)	Inheritance Relationship count	Inheritance Correctness (%)
1	Rational Software Architect	30	26	67.57	4	100
2	Visual Paradigm	31	27	54.05	4	100
3	Star UML	31	27	54.05	4	100
4	Enterprise Architect	31	27	54.05	4	100
5	MagicDraw	31	27	54.05	4	100
6	Altova Umodel	31	27	54.05	4	100
7	MyEclipse	20	16	27.03	4	100

From the case study, there are 37 association relationships and 4 inheritance or generalization relationships that make it 41 in total. Of these relationships, 3 are bidirectional. The result of this observation is shown in TABLE IV. By completing this observation, we found that only Rational Software Architect is capable of reconstructing bidirectional relationship. Other tools reconstruct bidirectional relations by means of two separate links in opposite directions. The RSA tool is also able to show single relationship for source code that declared two separated link relationships to the same class. Other tools show this kind of relationships as two separated associations. With those advantages, Rational Software Architect presented the highest percentage of relationship correctness. Visual Paradigm, Star UML, Enterprise Architect, MagicDraw and Altova UModel show the same percentage of correctness where the result in each extracted relationships are almost the same. However, MyEclipse shows some weakness in extracting the association relationship.

VI. DISCUSSION

Strength: Most of the tools are excellent in presenting the class attributes and methods. The tools are capable of extracting source code, visualizing the class diagram and enabling the user to manipulate the generated diagram. Some of the tools such as Altova UModel and Visual Paradigm are able to automatically generate the class diagram. Most of the tools need user intervention to drag and drop the classes in the project explorer-canvas to recreate a class diagram. This drag and drop function can be useful to the user to select the reverse engineered classes that they want to visualize in the class diagram. Of course, user intervention requires additional effort.

Weakness: All CASE tools are unable to correctly identify all the class relationship. Most of the tools identify aggregation and composition relationships as association relationships. Rational Software Architect shows the result differently by presenting dependency relationships for all class relationships

that were tested. For further investigation, we tested all evaluated tools by generating the source code based on design and then we reverse engineered the generated source code to produce the design. This test shows that we are unable to generate the same design that we created. We observed the generated source code and it shows the tools did not differentiate code generations between those types of relationship. This may be the reason why the tools are unable to produce the class relationship correctly. The tools' weakness(es) in generating code (forward engineering) and reverse engineer source code for class relationship have mentioned by Ralf Kollmann et al. [7] in 2002 and Akehurst et al. [13] in 2007. Although this paper is more recent, the tools are still unable to generate correct class relationships and present the relationship in reverse engineering functionality. However, two tools (MagicDraw, Rational Software Architect) give additional information by presenting dependency relationships as an addition to class relationship (association, aggregation and composition). These tools present a lot of dependency relationships (some of which are redundant) that make the resulting generated class diagram appear disorganized and sometimes confusing.

Today, CASE tools support the reverse engineering capability by not only using source code as input but also support object or class files and executable files such as .jar and .exe. Some tools such as Altova UModel, Rational Software Architect and Visual Paradigm offer more functionality where they are able to present sequence diagrams based on the reverse engineering result. Although they are not able to automatically generate the sequence diagram, it at least may help the software engineer or software maintainer to understand the class interactions. Overall, from the user point of view, the functionality to do reverse and forward engineering are easy to access by the user and the tools give good instruction and information to the user to use the functionality and analyze the results.

VII. RELATED WORK AND FUTURE WORK

Several evaluations and comparisons on reverse engineering tools have been made [7, 8, 9 and 10]. Ralf Kollmann et al. [7] presented a study which examined the reverse engineering capabilities of two CASE tools (Rational Rose, Borland Together) and compare the result with two academic prototype (Fujaba, IDEA). It shows the comparison between industrial Case tools and academic tools by presenting the strengths, weaknesses and similarities of the tools' capability. In our study, we examined seven industrial case tools that we believe are commonly used in the Industry. We extend the examination by observing the capabilities of the tools in reverse engineering the source code into package diagram and sequence diagram.

Jussi Koskinen and Tero Lehmonen [10] analyzed ten reverse engineering tools. The paper analyzed the capabilities in term of four aspects: data structures, visualization mechanisms, information request specification mechanisms and navigation features. Their paper focused on the information retrieval capabilities of the selected tools. Their selected tools do not offer the same functionality as our tools because not all tools are capable of reconstructing UML class

diagrams. In our paper, we have selected tools that support reconstruction of UML models and support Java source code.

Bellay and Gall [9] presented a study to compare reverse engineering tools for the C programming language. Four reverse engineering tools were selected in the study. The study aimed to show the differences of the strength and weakness of the selected reverse engineering tools based on their usability, extensibility and applicability for embedded software systems. The tools selected in their study are different in functionality and capability. In our study, our evaluated tools are comparable because the functionality of the evaluated tools is relatively similar.

Gahalaut and Khandnor [8] presented a study about reverse engineering java code. The study aimed to compare byte code reverse engineering tools (decompiler) with UML reverse engineering tools (Altova UModel and Enterprise Architect). The input for this comparison is java source and java class files. They stated that the decompiler and the UML reverse engineering tools generate the same class structures. However, in our study, although the structure is about the same, the detail in class information and relationship is different if we compare reverse engineered class diagram based on the class file and java source file.

For future work, we propose this evaluation to be extended to larger systems to evaluate the scalability and performance of the tools. Also, future research in reverse engineering should try to come up with abstraction mechanisms for leaving out details and emphasize important information from reverse engineered source code.

VIII. CONCLUSION

This paper has provided an assessment of the reverse engineering capability of seven CASE tools (6 commercial and one open source). We have assessed the tools by evaluating the reverse engineering features that are provided. Basically, all CASE tools are capable of performing reverse engineering from source code to class diagrams and package diagrams. Some of the tools can also reverse engineer sequence diagram, but need a little help from the user to do this. The tools also support various types of input formats other than source code, such as class or object file and executable file. Even though these input formats offer additional options to the user, the resulting diagrams differ from the results from using source code as input.

Generally, there are not many differences between the capabilities of tools in reverse engineering into UML. Almost all the evaluated tools have relatively the same strengths and weaknesses: CASE tools do not completely show all class information, and CASE tools are also not capable of correctly and completely presenting the class relationships – especially aggregation and composition.

With the state of the practice of current tools, details that are omitted from relations in class diagrams can lead to

misinterpretations. The CASE tools providers could improve their reverse engineering capabilities by better identifying the association, aggregation and composition relationship. For user that consider using reverse engineering using CASE tools as a means of discovering their design should be aware of the weaknesses of these tools: Even though the tools result present a lot of UML diagrams, not all the result are correct or complete.

ACKNOWLEDGMENT

We would like to thank all the CASE tools and the software metric tool providers for giving us the evaluation and academic license for this assessment purpose.

REFERENCES

- [1] Fowler, M.: UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd Edition Addison-Wesley, New York,(1997)
- [2] <http://www.uml-diagrams.org/>
- [3] Yann-Gaël Guéhéneuc and Hervé Albin-Amiot. Recovering Binary Class Relationships: Putting Icing on the UML Cake. In Doug C. Schmidt, editor, *Proceedings of the 19th Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, pages 301–314, October 2004.
- [4] <http://www.math-cs.gordon.edu/courses/cs211/ATMExample/>
- [5] C. Bennett, D. Myers, D. Ouellet, M.-A. Storey, M. Salois, D. German, and P. Charland. A survey and evaluation of tool features for understanding reverse engineered sequence diagrams, *J.Softw. Maint. Evol.: Res. Pract.*, vol. 20, no. 4, pp. 291–315, 2008.
- [6] Bruce E.Wampler. *The Essence of Object-Oriented Programming with java and UML*, 1st Edition, Addison-Wesley, 2002.
- [7] R. Kollmann, P. Selonen, E. Stroulia, T.Systa and A. Zundorf. A Study on the Current State of the Art in Tool-Supported UML-Based Static Reverse Engineering. *Proc. IEEE Working Conference on Reverse Engineering*, Richmond VA, pp. 22-32, 2002.
- [8] Asit Kumar Gahalut, Padmavati Khandnor. Reverse Engineering : An Essence for Software Re-engineering and Program Analysis. *International Journal of Engineering Science and Technology*, Vol. 2, No. 06, 2010.
- [9] B. Bellay and H. Gall. A Comparison of Four Reverse Engineering Tools. 4th Working Conference on Reverse Engineering, pages 2-11, The Netherlands, 1997.
- [10] Jussi Koskinen, Tero Lehmonen. Analysis of Ten Reverse Engineering Tools. *International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE)*, 2008.
- [11] <http://www.sdmetrics.com/>
- [12] Elliot J. Chikofsky and James H. Cross II. Reverse Engineering and Design Recovery: A Taxonomy. *IEEE Software* 1990.
- [13] D. Akehurst, G. Howells, K. McDonald-Maier. Implementing Associations : UML 2.0 to Java 5. *Software and Systems Modeling*, vol. 6, no. 1: 3-35, March 2007.
- [14] Ariadi Nugroho, M. R. V. Chaudron. A Survey of the Practice of Design - Code Correspondence amongst Professional Software Engineers, *Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, p.467-469, September 20-21, 2007
- [15] Timothy C. Lethbridge , Janice Singer , Andrew Forward. How Software Engineers Use Documentation: The State of the Practice, *IEEE Software*, vol. 20 no. 6, p.35-39, November 2003.
- [16] <http://www.liacs.nl/~heijstek/hafeez/Hafeez2011.pdf>