Interactive scalable condensation of reverse engineered UML class diagrams for software comprehension

Osman, M.H.B.

Chapter 4

Assessing the Correctness and Completeness of UML CASE tools in Reverse Engineering

This chapter 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 and round-trip engineering. For these purposes, the quality and correctness of the reverse engineering capability of these tools are of key importance: Do the tools recover all important information from the source code? Are the reverse engineering results correct? What kind of information is presented in the result? Based on these questions, we compare eight UML CASE tools (six commercial tools and two open source tools). We evaluate i) the types of input that these tools can handle, ii) the types of diagrams that can be reconstructed, and iii) the quality of resulting diagrams.

4.1 Introduction

The Unified Modeling Language (UML) is the standard for graphically representing the design of object-oriented software systems. While UML diagrams are created in forward design, these diagrams are poorly maintained. Maintaining correspondence

This chapter is adapted from publications entitled **"An Assessment of Reverse Engineering Capabilities of UML Case Tools"**, In Proceedings of the 2nd Annual International Conference of Software Engineering & Applications (SEA 2011) and **"Correctness and Completeness of CASE tools in Reverse Engineering Source Code into UML Model"**, In GSTF Journal on Computing vol.2, num.1 (2012)

(between design and implementation) is particularly challenging because over time an implementation tends to evolve considerably from its initial design [118]. 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 are frequently present [72]. Lethbridge et al. [103] confirm the widely held belief that software engineers typically do not update the documentation as timely or completely as software process personnel and managers advocate. 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 [97].

Nowadays, a lot of commercial and open source CASE tools support reverse engineering. These tools provide the capability in reconstructing package and class diagrams based on source code, objects and/or executable files. These tools also provide an automated and semi-automated analysis of the software system regarding the software structure such as class, attribute and operation. Some of the CASE tools extend the UML reverse engineering capabilities by supporting sequence diagrams reconstruction (based on static analysis).

For this research, our motivation is to discover to what extent the CASE tools are able to reverse engineer UML diagrams out of source code. This information is useful because RE-CDs is one of the important inputs in this research. Particularly in UML class diagrams, we want to know what type of information provided in the RE-CD compared to the typical information that may exist in class diagrams (domain model/forward design). The study of this chapter also aim to gather information about CASE tool(s) that are suitable for the research in this thesis. In order to find the answers, we examined and compared the reverse engineering capabilities provided by the CASE tools. In total, eight CASE tools have been selected in this study, namely Visual Paradigm, Rational Software Architect, StarUML, Altova UModel, MyEclipse, Enterprise Architect, MagicDraw and ArgoUML. To understand how the tools analyze class diagram, we conduct three experiments.

The first experiment aims at discovering the capability of the evaluated CASE tools in performing the round-trip engineering [80] task. The second experiment evaluates the tools' capabilities of identifying class relationships (association, aggregation and composition) based on the code stated in [72]. The third experiment assesses the correctness and completeness of the tools in reverse engineering source codes into class diagrams.

The chapter is structured as follows. Section 4.2 presents the related work. Section 4.3 briefly describes the examined tools and properties used in this evaluation. Section 4.4 describes the sample cases and Section 4.5 explains the approach of this experiment. Section 4.6 presents our results and findings. Our evaluation is discussed in Section 4.7. This is followed by our conclusion and future work in Section 4.8.

4.2 Related Work

This section discusses work that related to this study. The following researchers have conducted several evaluations and comparisons of reverse engineering tools.

Kollmann et al. [94] presented a study that examined the reverse engineering capabilities of two CASE tools (Rational Rose, Borland Together), and compared the result with two academic prototypes (Fujaba, IDEA). Their study investigates the strengths, weaknesses and similarities of the tools' capability. In our research, we examine six commercial CASE tools and two open source 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.

Koskinen and Lehmonen [97] analysedten reverse engineering tools in term of four aspects: data structures, visualization mechanisms, information request specification mechanisms and navigation features. Their research focused on the information retrieval capabilities of the selected tools. However, not all of their selected tools were capable of reconstructing UML diagrams. In our study, we selected tools that support reconstruction of UML models.

Bellay and Gall [22] presented a study that compared reverse engineering tools for the C programming language. Four reverse engineering tools were selected in the study. Their study aimed to discover 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 were different in functionality and capability. In contrast, our evaluated tools are comparable because the functionalities of the evaluated tools are relatively similar.

Gahalaut and Khandnor [64] presented a study about reverse engineering Java code. The study aimed to compare bytecode reverse engineering tools (decompiler) with UML reverse engineering tools (Altova UModel and Enterprise Architect). The inputs for this comparison were Java source code and Java class files. They stated that the decompiler and the UML reverse engineering tools generated the same class structures. However, our extended study found that although the structure were about the same, the detail in class information and the relationship were different if we compare RE-CDs that are constructed based on the class file and Java source file.

Akehurst et al. [13] focused on providing solutions to the issues of mapping qualified associations and the UML 2.0 semantic variations of an association into Java 5. They presented a comparison of forward engineering functionality of some CASE tools. In contrast, our evaluation covers forward and reverse engineering of class diagrams based on the user's view. Their study was centered on how to generate code based on the design. Our study evaluates and compares the tools' capabilities to reverse engineer basic class information and relationships.

Boklund et al. [30] performed a comparative study of forward and reverse engi-

neering in UML tools; focused on three-tier layered web services application. They evaluated four modeling tools in the perspective of UML-Modeling, UML-based Code Generation and Reverse Engineering UML-diagram from code. However, not all tools that they have selected can be used in their evaluation. In contrast, our selected tools are comparable in term of the tools capability and functionality.

Kearney and Power [87] proposed a framework and automated tool for benchmarking UML CASE tools reverse engineering capabilities. The automated tools presented in this study tightly rely on the input from software metrics tools. Although we conduct our experiment semi-automated, we present more information rather than concentrate only on software metrics.

4.3 Examined Tools and Properties

This section describes the examined tools and properties that are involved in this experiment.

4.3.1 Examined Tools

The CASE tools were chosen based on the following criteria:

- Capable of performing forward and reverse engineering in Java.
- Capable of exporting UML Model to XML Metadata Interchange (XMI) format.

In total, eight well-known CASE tools are selected as listed in Table 4.1. For commercial CASE tools, we used fully functional evaluation and academic evaluation versions. We used SDMetrics [180] (version 2.11 – academic license) to extract UML model information from different versions and different type of XMI files.

4.3.2 Examined Properties

The examined properties are divided into two parts: Reverse Engineering Capability and Class Diagram Properties.

Reverse Engineering Capability

The reverse engineering tool's capabilities are evaluated from three perspectives: UML diagrams, supported programming languages and supported types of source.

- *UML Diagrams*
  The selected types of UML diagram are: package diagram, class diagram and sequence diagram. We analysed all selected diagrams by evaluating (1) the process of reconstructing (reverse engineer) the diagrams, and (2) the output in term of completeness and representation. Only static analysis is used for reconstructing those diagrams.

Table 4.1: *List of Evaluated CASE Tools*

No	CASE Tool	Information	Vendor	License Type
1	Visual Paradigm 8.1	http://www.visual-paradigm.com	Visual Paradigm	Evaluation
2	MagicDraw 17.0	http://www.magicdraw.com	No Magic	Evaluation (academic)
3	Altova Umodel 2011	http://www.altova.com	Altova	Evaluation
4	Enterprise Architect 8.0	http://www.sparxsystems.com.au	Sparx System	Evaluation
5	Rational Software Architect 8.0.1	http://www-142.ibm.com/software/products/my/en/swarchitect-websphere	IBM	Evaluation
6	MyEclipse 8.6	http://www.myeclipseide.com	Genuitec	Evaluation (academic)
7	StarUML 5	http://staruml.sourceforge.net	StarUML	Open Source
8	ArgoUML	http://argouml.tigris.org	Tigris.org	Open Source

- *Supported Programming Languages*
  We study the capability of the CASE tools to reverse engineer source code from several common programming languages: PHP5, C++, Java, C#, Delphi, Python and Visual Basic (V.B). However, we perform reverse engineering using a sample case developed in Java. Other programming languages are evaluated based on the documentation/manual provided by the CASE tools' provider.
- *Additional Types of Input Formats*
  The supported input-types for reverse engineering UML diagrams (in addition to source code; e.g. binaries).

Class Diagram Properties

We evaluate the class diagram properties based on the following elements:

- *Attributes and Methods*
  - Number of attributes: The tools' ability to reconstruct all attributes including the type of attribute (public, private, protected) defined in the source code.
  - Number of operations: The tools' ability to reconstruct all methods (of all: public, private, protected, constructor) defined in the source code.
- *Relationship*
  - Number and types of relationship: The ability of the tools to reconstruct all relationships between classes.

– Association relationship: The capability of the tool in detecting association and binary association [68] relationship (i.e. aggregation[1] and composition[2]).

4.4 Sample Cases

This section describes the sample of cases that are used in this evaluation.

4.4.1 Movie Catalog System (MovieCat)

This sample case is derived from [177]. We altered the relationships in this class diagram to make sure all types of relationship were presented. This sample case is selected because of a little amount of classes and the class diagram can be altered to suit our purpose in this research. We use this sample case to evaluate the class diagram properties.

4.4.2 Automatic Teller Machine (ATM) Simulation System

This sample case is selected because it has a fully functional simulation system, a forward design and a complete implementation source code. It is developed by the Department of Mathematics and Computer Science, Gordon College [28]. The complete software documents consist of 22 designed classes. Various types of relationships were used in the UML diagram such as association, composition, dependency and inheritance. Some of the elements (especially class relationship) in this sample case have been altered to suit our requirement for the experiment. This sample case was used to evaluate the reverse engineering capability.

4.5 Approach

This section explains our approach to evaluating the tools. The evaluation is divided into two parts which are: Round-trip Capability and Reconstruction of UML Diagram types.

4.5.1 Round-trip Capability

We performed the round-trip capability experiment to assess the completeness of the CASE tools in recovering all information specified in the forward design (illustrated

[1] Aggregation (or shared aggregation) is a part-of relationship [63]. It is used when part of an instance (or class) is independent which means, if the related instance is deleted, the other instance may still exist.

[2] Composition (or composite aggregation) is a strong form of aggregation that requires a part instance be included in at most one composite at a time. If a composite is deleted, all of its parts are normally deleted with it [68].

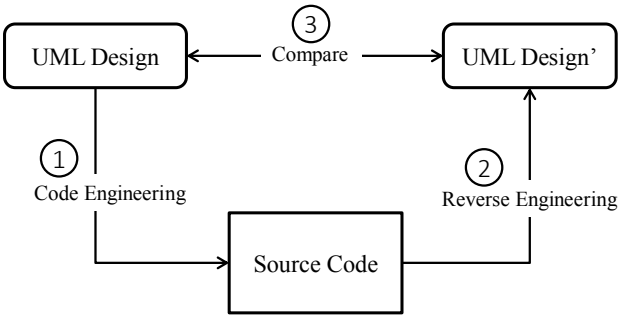


Figure 4.1: *Round-trip Engineering Experiment*

in Figure 4.1). We expected to get an overview of an automation of software lifecycle phases (i.e. software design->code generation->reverse engineering) using CASE tools. This experiment begins with creating the forward design class diagram (UML Design) that consisted of: (i) Attributes and Methods (private, protected, public) and, (ii) Relationships (association, aggregation, composition, inheritance). We generate the source code based on the UML Design through "Code Engineering" process, and produce the UML Design' by reverse engineer this source code. Then, we compare the UML Design and UML Design'.

4.5.2 Reconstruction of UML Diagram Types (package/class/sequence)

To assess the capability and the quality of the reverse engineering of UML diagrams, we conduct the following experiments.

UML Diagrams Reconstruction Capability

We evaluate the supported types of (reverse engineered) diagrams by i) reconstructing the diagram using the CASE tools and/or ii) finding the information on the tools' documentation/manuals. The tools' capabilities of reconstructing UML diagrams are analysed using a three-level scale which are described as follows:

- "+" : The tool is able to reverse engineer the specified diagram.
- "o" : The tool is able to reverse engineer the specified diagram, but present minimal information. For instance, the CASE tool is capable of presenting classes, attributes and operations but not relationships. Another example is the tool needed user intervention to generate the sequence diagram.
- "-" : The tool is unable to reverse engineer the specified diagram.

The information about the supported programming language and supported types of reverse engineering sources are gathered from the tool's documentation or/and manuals.

Detection of Aggregation, Association and Composition Relationship

The evaluation of reconstructing various types of class relationship is performed by using the source code defined in [72]. We created a source code that represents each evaluated relationship (i.e. aggregation, association and composition). Then, we reverse engineer this source code to evaluate the ability of the CASE tools in detecting multiple types of relationship.

Correctness and Completeness (CnC) of Reconstructed UML Diagram

This experiment aims at evaluating the completeness and correctness of the RE-CD constructed by the CASE tools. For the expected result (concrete result), we manually extract all class diagram information from the sample case design document and implementation code. For instance, we calculate (manually) the number of attribute and operation in every class in the class diagram. Then, the class diagram information gathered from RE-CDs (from each tool) are compared with the expected result. The evaluation is divided into two parts:

- *CnC of Class Information*: We tested all possible options to reconstruct the best RE-CD for each tool. The diagrams then exported to XMI files. We extract the metrics from the XMI files and compare the class diagram information with our expected result.
- *CnC of Reconstruction of Class Relationship*: All possible reverse engineering options are evaluated to achieve the best view of class relationship. Then, we manually compare the RE-CD relationships with our expected result.

4.6 Result and Findings

In this section, we present our findings which include: Reverse Engineering Capability and Class Diagram Properties.

4.6.1 Reverse Engineering Capability

In this subsection, we present the capability of the CASE tools in reconstructing the UML Diagrams, the supported programming languages and the supported types of source.

UML Diagrams

The results in Table 4.2 show that most of the tools were capable of reconstructing package diagrams. Visual Paradigm, MagicDraw and Altova UModel are good at reconstruction package diagram because these tools can perform this task automatically. An example of a reverse engineered package diagram is shown in Figure 4.2.

Table 4.2: *Supported UML Diagrams for Reverse Engineering*

No	Tools	UML Diagram		
		Package	**Class**	**Sequence**
1	Visual Paradigm 8.1	+	+	o
2	MagicDraw 17.0	+	+	o
3	Altova Umodel 2011	+	+	-
4	Enterprise Architect 8.0	o	+	-
5	Rational Software Architect 8.0.1	o	+	-
6	MyEclipse 8.6	o	+	o
7	StarUML 5	o	+	o
8	ArgoUML	o	o	-

In terms of the class diagram, all the evaluated tools are good at automatically reconstructing RE-CD (from source code) except ArgoUML. The reason is that the ArgoUML was unable to reconstruct the class relationship other than inheritance. All CASE tools give an option to generate the class diagram separately using the "drag and drop" function.

The reconstruction of sequence diagrams requires a lot of manual intervention because the users need to provide the information on the methods and classes that should be in the sequence diagram. Based on our experiment, only four tools have the capability of reverse engineering sequence diagrams. An example of a reverse engineered sequence diagram is illustrated in Figure 4.3.

The Supported Programming Languages

The supported programming languages results are presented in Table 4.3. It shows that the Enterprise Architect is able to reverse engineer all the programming languages listed in this evaluation. We also found that all evaluated tools were able to reverse engineer source code in Java.

The Types of Source

Overall, all evaluated CASE tools can reverse engineer class diagrams out of source code files (e.g. .java, .cpp and .cs). The CASE tools also offer an option to specify the source directory, and then automatically determine the source code file from the directory. Visual Paradigm, Altova and Enterprise Architect are capable of decompiling and reconstructing class diagram based on Java bytecode (.class), dynamic link library (.dll), execution file (.exe) and Java archive (.jar). Then, the tools generate class information that enable the users to construct a class diagram. The supported types of source are presented in Table 4.4.

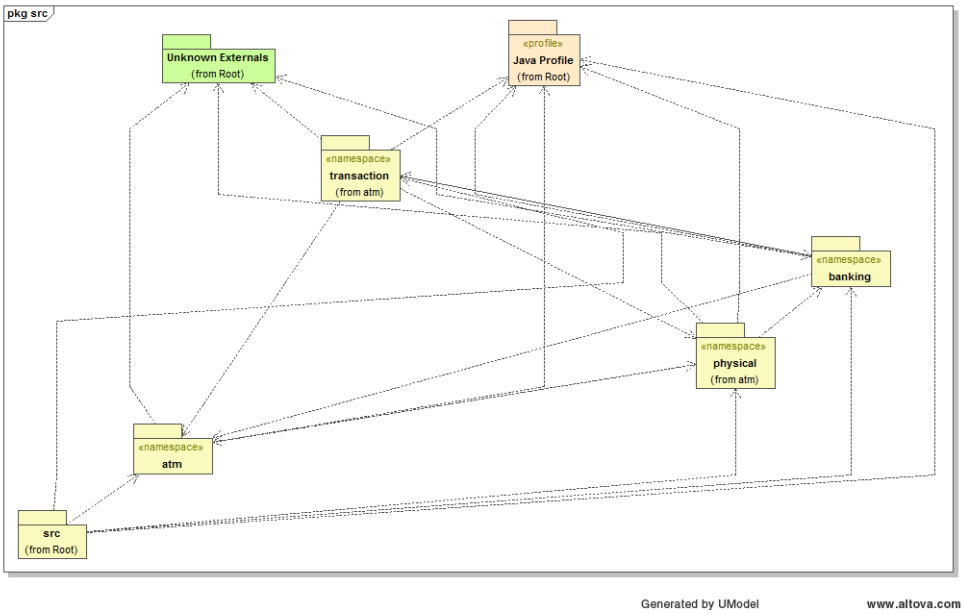


Figure 4.2: *Altova Reverse Engineered Package Diagram*

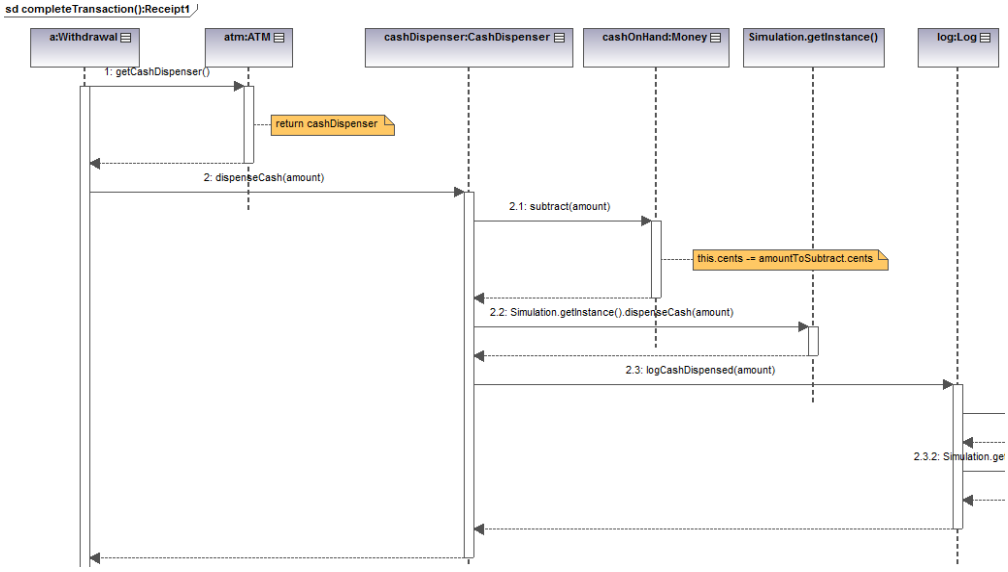


Figure 4.3: *Reverse Engineered Sequence Diagram using Altova*

Table 4.3: *Supported Programming Language for Reverse Engineering*

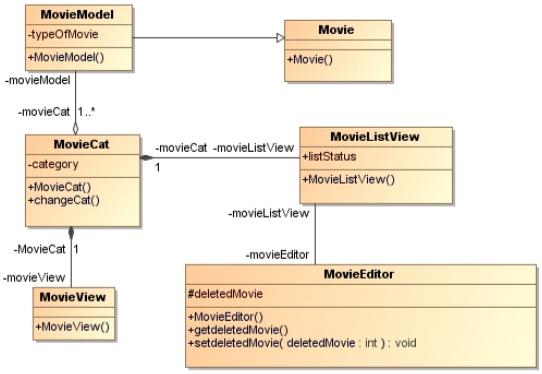
No	Tools	PHP 5	C++	Java	Delphi	Phyton	V.B	C#	Total 'Y'
1	Visual Paradigm	Y	Y	Y	N	Y	N	N	4
2	Altova UModel	N	N	Y	N	N	Y	Y	3
3	MyEclipse	N	N	Y	N	N	N	N	1
4	StarUML	N	Y	Y	N	N	N	Y	3
5	MagicDraw	N	Y	Y	N	N	N	Y	3
6	Rational Software Architect	N	Y	Y	N	N	Y	Y	4
7	Enterprise Architect	Y	Y	Y	Y	Y	Y	Y	7
8	ArgoUML	N	Y	Y	N	N	N	Y	3

4.6.2 Class Diagram Properties

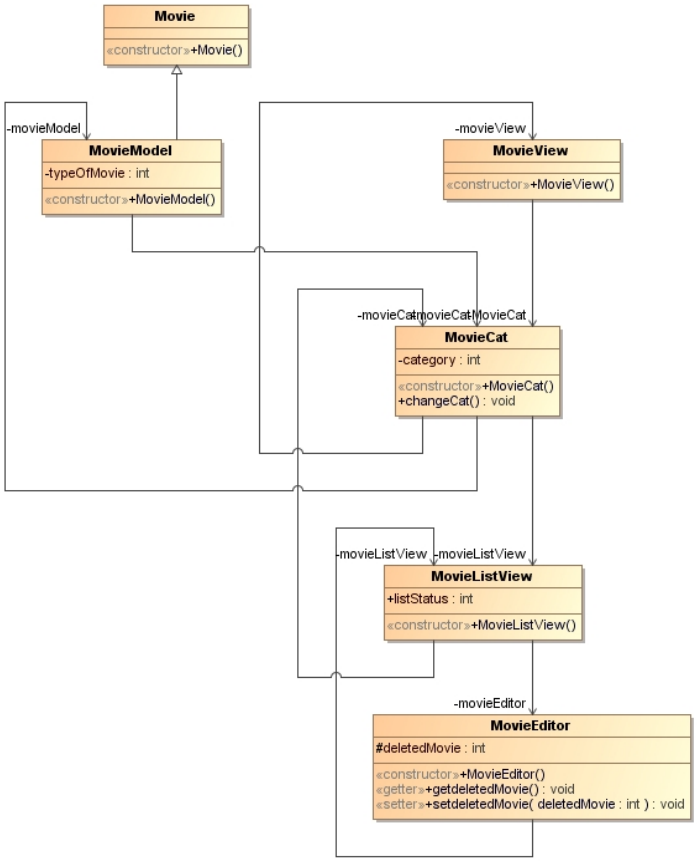
This subsection presents the assessment of the class diagram properties. The results are divided into: Round-trip Findings, Class Relationship Test, and Class Diagram Correctness and Completeness.

Round-trip Findings

The CASE tools successfully round-trip the information for class attributes and operations. However, the round-trip result for class relationships is different amongst the CASE tools. In general, all CASE tools correctly round-trip association and inheritance relationship. The aggregation and composition relationships were presented as association. Only Rational Software Architect presented aggregation and composition as a dependency. These aggregation and composition relationships were failed to be detected during the round-trip experiment due to the code-engineering process (transform design to source code) that defined those relationships as a link declaration [177] in the source code. Therefore, the tools cannot differentiate the types of relationship (example is in Figure 4.4). The discontinuity that may exist between object-oriented modeling language and programming language may be the reason behind this result. This discontinuity arises from the ambiguous concept in modeling languages and lack of corresponding concepts in programming languages [72]. These relationships represent different knowledge in software design. Unable to recognize these relationships correctly may hinder the traceability between source code and design, hence, obstructing software analysis.



(a) Forward Engineering Class Diagram



(b) Reverse Engineered Class Diagram

Figure 4.4: *Round-trip Test Result*

Table 4.4: *Additional Types of Input Format*

No	Tools	Source Code	Class/Object /Dynamic Link Library	Executable	Other
1	Visual Paradigm	.java, .cpp, .h, .php	.dll, .class, .inc	.exe, .jar	Source Directory, .zip
2	AltovaUModel	.java	.dll, Global Cache (GAC), MSVS .Net, .class	.exe, .jar	Source Directory
3	MyEclipse	.java	-	-	Source Directory
4	StarUML	.java, .cpp, .h, .cs	-	-	Source Directory
5	MagicDraw	.java, .cpp, .h, .cc, .cs	-	-	Source Directory
6	Rational Software Architect	.java, .cpp, .h, .cc	-	-	Source Directory
7	Enterprise Architect	.java, .h, .cs, .hpp, .pas, .php, .php4, .inc, .py, .vb, .cls, .frm, .ctl	.class, .dll	.exe, .jar	Source Directory
8	ArgoUML	.java, .cpp, .cs	.class	.jar	Source Directory

Class Relationship Test

The results of this experiment are illustrated in Table 4.5. It shows that all CASE tools are unable to reconstruct the specified relationships based on the source code (defined in [72]). Visual Paradigm, Altova UModel, StarUML, MyEclipse, MagicDraw and Enterprise Architect unable to generate the association relationships, while the aggregation and composition relationships were presented as association relationships. An example of aggregation test results is shown in Figure 4.5.

Class Diagram Correctness and Completeness

Class Diagram Correctness and Completeness (CnC) evaluation is divided into two parts: Attributes and Methods, and Relationship.

Table 4.5: *Class Relationship Test Result*

No	Tools	Association	Aggregation	Composition
1.	Visual Paradigm	No relationship presented	Present as association	Present as association
2.	Altova UModel	No relationship presented	Present as association	Present as association
3.	MyEclipse	No relationship presented	Present as association	Present as association
4.	StarUML	No relationship presented	Present as association	Present as association
5.	MagicDraw	Present as dependency	Present as association and dependency	Present as association and dependency
6.	Rational Software Architect	Present as dependency	Present as dependency	Present as dependency
7.	Enterprise Architect	No relationship presented	Present as association	Present as association
8.	ArgoUML	No relationship presented	No relationship presented	No relationship presented

1. *CnC of Attributes and Methods* evaluation presents the capability of the CASE tools in identifying class attributes and methods (or operations). The results are presented in Figure 4.6 and the explanations are the following:

    - **Number of Attribute(NA)**: We expected the tools to extract 79 attributes (NA) from the sample case. Visual Paradigm, Enterprise Architect and Rational Software Architect successfully extracted all the attributes. However, Rational Software Architect can only show the attributes by using the "Drag and Drop" function instead of using the "Transform" function. The class diagram generated from the "Transform" function did not show any attribute even though it exist in the tools "Project Explorer" pane. Furthermore, the generated XMI file also does not include any attribute. Other tools like Altova UModel, MyEclipse, StarUML and MagicDraw were unable to extract all the expected attributes.
    - **Number of Operations(NO)**: We expected the tools to extract 91 operations. However, most of the tools found more than expected. The reason is that the additional operations come from the "superclass" operations. StarUML did not completely extract all operations because it was unable to extract 4 constructors of 4 classes. On the other hand, Visual Paradigm extracted 77 operations.

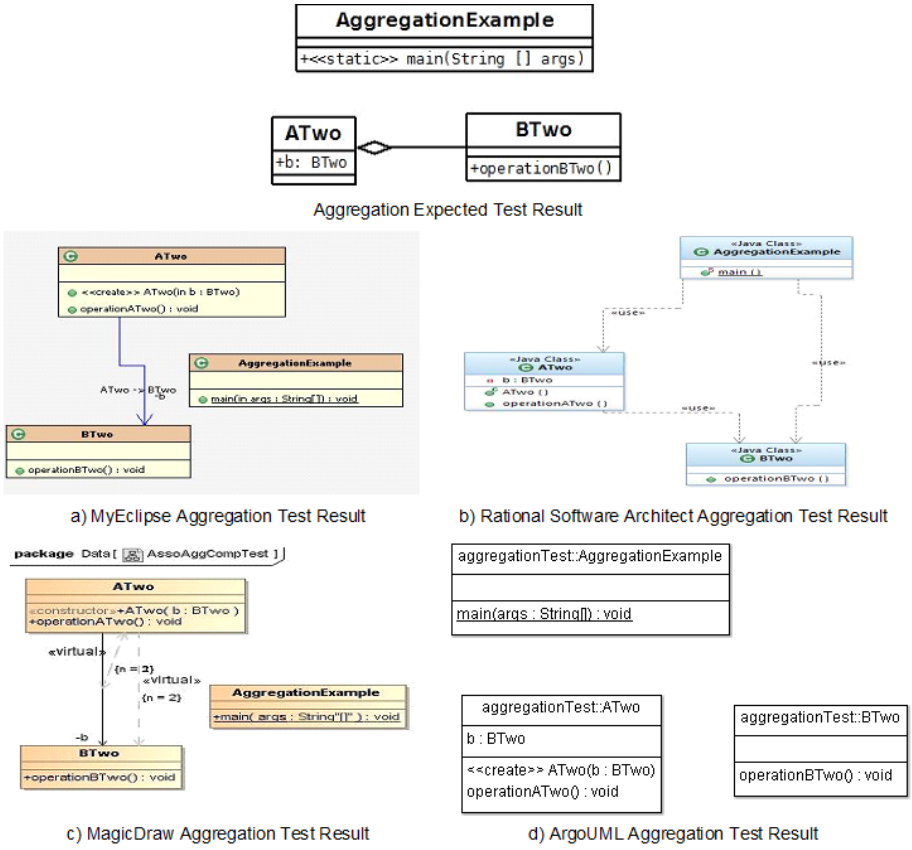


Figure 4.5: *Example of Diagram on Aggregation Test*

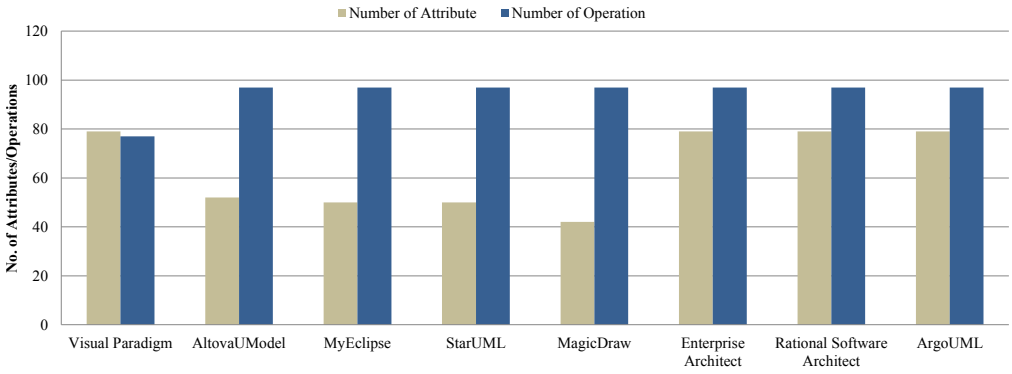


Figure 4.6: *Number of Attributes and Methods*

Table 4.6: *Relationship Correctness*

No	Tools	No. of Relationship	Association		Inheritance	
			Total	Correctness (%)	Total	Correctness (%)
1	Visual Paradigm	31	27	54.05	4	100
2	StarUML	31	27	54.05	4	100
3	Enterprise Architect	31	27	54.05	4	100
4	Rational Software Architect	30	26	67.57	4	100
5	MagicDraw	31	27	54.05	4	100
6	MyEclipse	20	16	27.03	4	100
7	Altova UModel	31	27	54.05	4	100
8	ArgoUML	4	0	0	4	100

2. *CnC of Class Relationship* evaluates the tools' capability of extracting association and generalization (inheritance) relationships. Other class relationships are not included in this evaluation as the Round-trip results (see Section 4.6.2) shows that the evaluated tools can only identify these relationships. For this purpose, we extracted all link declarations from our sample case (ATM simulation system) and used it as the expected result. In total, there are 41 relationships identified that consist of 37 association relationships and 4 generalization relationships. Of these relationships, three of them were bidirectional. The overall results are presented in Table 4.6. We found that only Rational Software Architect was capable of reconstructing bidirectional relationship. Other tools (except ArgoUML) reconstruct bidirectional relations by means of two separate links in opposite directions (example in Figure 4.7). Rational Software Architect also capable of presenting two separated relationships that directed to the same class as a single relationship. Other tools identified this kind of relationships as two separated associations.

4.7 Discussion

This section discusses the experiment results.

- **Strength** : Most of the tools are excellent in recovering the class attributes and methods. From the result, the tools were capable of extracting source code, visu-

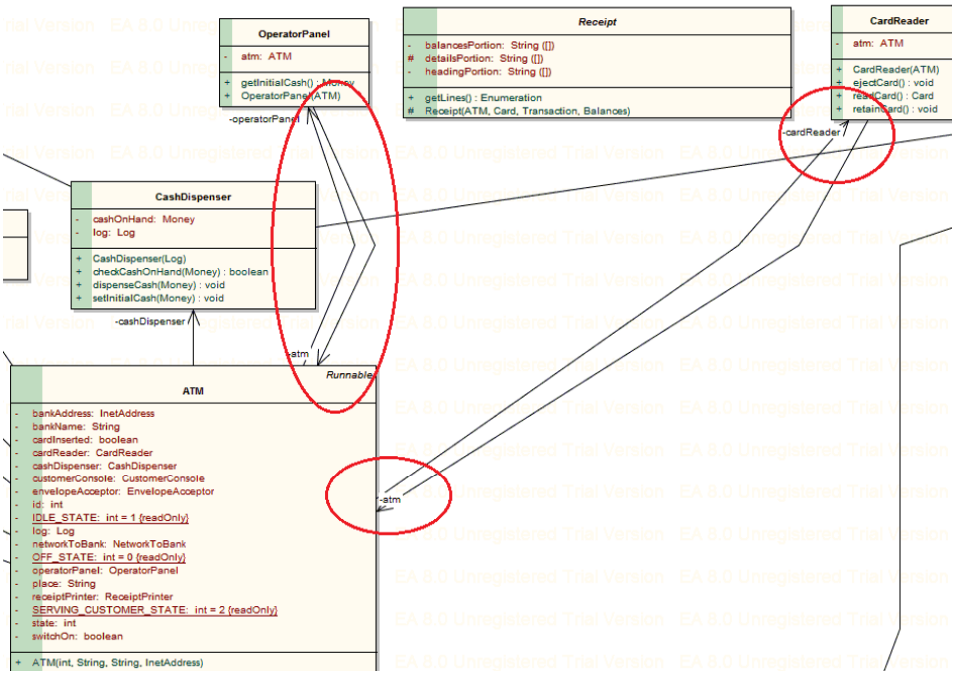


Figure 4.7: *Bidirectional Relationship with Two Separate Links*

alizing the class diagram and enabling manipulation of the generated diagrams. Some of the tools such as Altova UModel and Visual Paradigm are able to generate the class diagram automatically; 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 desired to visualize in the class diagram. Of course, user intervention requires additional effort.

- **Weakness**: All CASE tools are unable to identify all the class relationships correctly. Most of the tools identify aggregation and composition relationships as association relationships. Rational Software Architect has presented 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 the design and then we reverse engineered the generated source code to produce the design (Round-trip). As the result, this test indicated that we were unable to generate the same design that we created. We observed the generated source code and it showed that the tools did not differentiate code generation between those types of relationship. This may be the reason of the tools are unable to recover the class relationship correctly. Incorrect code generation would lead to inaccurate synchronization between source code

and software design document. The design document becomes out-of-sync that decays the knowledge of the document.
The tools' weaknesses in generating code (forward engineering) and reverse engineer source code for class relationship have mentioned by Ralf Kollmann et al. (2002) [94] and Akehurst et al. (2007) [13]. Although this study is more recent, the tools are still unable to generate correct class relationships. However, two tools (MagicDraw, Rational Software Architect) give additional information by presenting dependency relationships as an addition to class relationship (association, aggregation and composition). The aggregation and composition relationship are essential to show how the software works. This relationship information may give some hints for the software engineer or the software maintainer which classes are important based on the software design before they browse the source code. The class relationship knowledge (especially which class to initialize after another) has to be discovered before the software engineer or software maintainer touch the source code.

Today, CASE tools support the reverse engineering capability not only by 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 the 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.

The experiments that were conducted in this chapter rely on manual evaluation of the test result and from the support of software metrics tool. As we know that some of the inputs are based on XMI files, we did not consider a faulty XMI generation by UML CASE tools. We also did not consider if the software metrics tool used was unable to extract some of the metrics from the XMI files.

4.8 Conclusion and Future Work

This chapter has provided an assessment of the reverse engineering capability of eight CASE tools (six commercial and two open source). We have assessed the tools by evaluating the reverse engineering features that are provided. In summary, 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 diagrams, but need a little help from the user. The tools also support various types of input formats other than source code, such as class or object files and executable files. 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 the CASE 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.

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 RE-CD.

From this research, after the correctness and completeness of the RE-CD is identified, the studies on condensation of RE-CD in Chapter 7, 8 and 9 will consider the following information:

1. Aggregation and composition as association relationship.
2. Two directional relationships (with different direction) should present as bidirectional relationship.
3. Two directional relationships (with the same direction) should present as a single relationship.
4. Only several CASE tools are suitable to reverse engineer source code to the class diagrams.