

A COMPARATIVE STUDY OF FORWARD AND REVERSE ENGINEERING IN UML TOOLS

Andreas Boklund¹, Stefan Mankefors-Christiernin, Christer Johansson, Håkan Lindell
*Department of Technology, Mathematics and Computer Science
University of Trollhättan-Uddevalla
P.O. Box 957, SE-461 29 Trollhättan
Sweden*

ABSTRACT

In the process of developing computer based applications the use of descriptive models to depict the architecture and design of the application is practical and brings many advantages. The most commonly used standard today is the Unified Modeling Language. There are many tools that implemented support for this standard for modeling, and they vary in functionality. A comparative study, including a selection of these tools, will show advantages and disadvantages for each tool. The purpose of this study is to test a selection of these modeling tools for a typical three-tier layered web service application. The tools tested in this study are Microsoft Visual Studio 2005 .NET Beta, PowerDesigner and Visual Paradigm. Rational Rose would have been included if it would have supported C#. In this study tests are performed in regard to modeling a class diagram, forward engineering to source code from the class diagram and reverse engineering from source code to class diagram. Despite the shortage of diagram types MS Visual Studio proved to be the most reliable tool for this particular type of applications.

KEYWORDS

Software tools and techniques, modeling languages, object oriented analysis and design, UML

1. INTRODUCTION

Software in the modern sense has been developed continuously for more than half a century. Already at an early stage it was realized that it is important to be able to properly analyze, design and structure the implementation and/or maintenance of software. Towards this end e.g. flow charts (Sommerville, 2001) and JSP-diagrams (King, Pardoe, 1992) were introduced relatively early on. The software of today, however, is exponentially much more complex, and is possibly only surpassed by the bio-chemical complexity in modern genetics. In order to address the need for proper modeling tools that had emerged during the 80-ties and early 90-ties Rumbaugh and Booch (Booch, Rumbaugh, Jacobson, 1999), two leading methodologists by the time, found it wise to merge their processes into one, which shortly after was merged together with the process of another methodologist by the name of Jacobson (Booch, Rumbaugh, Jacobson, 1999)(Fowler, Scott, 2000)(OMG, 2003). In 1996 their joint effort had led to a notation called the Unified Modeling Language (UML). In 1997 OMG assumed responsibility for the notation along with parts of processes of leading software vendors and practitioners such as Rational Software, Microsoft and IBM. Today UML is a de facto standard for modeling of software.

Already from the first version it was clear that the modeling capabilities of UML does not only provide a way to analyze, describe and model the software, however, but actually could be utilized in order to build it as well: given robots to follow a detailed enough blue print, a house can be “built” by the architect him/herself. Since software is a virtual product just as the UML-model, an automatic tool could translate a detailed model directly into code; in almost the same way as a compiler translates high level programming languages into machine code that can be interpreted by computer hardware (Whalen, Heimdahl, 1999). This

¹Authors are listed in alphabetical order

code generating possibility of UML is a fact that has resulted in numerous code-generating tools (Baxter, Mehlich, 1997)(Holt, 2001). Whether such code actually meets the formal requirements or not is a combination of the quality of the UML language itself (Ekberg, Christiernin, Boklund, 2004), the capabilities of the tool at hand and the nature of the requirements, since certain requirements might be well outside the scope of UML such as e.g. “optimized for execution speed on platform X”. Also in such scenarios, UML-generated code might be used as a “first skeleton” or “draft”, though. The cost saved by limiting the need for manual implementation has consequently made code generating UML-tools widely popular (Code Generation Network, 2005). It is also the only obvious mean able to keep up the pace of the exponential growth of produced software: only machines can keep up with machines.

Against this backdrop, the quality and potential drawbacks of current UML-based code generators (UML-CG:s) is of vital importance. Even if UML itself does not allow for certain types of code structures (Ekberg, Christiernin, Boklund, 2004), any CG should adhere to the current UML standard. Otherwise does the modeling and code-generation risk becoming incoherent over time. Similarly should any combined CG/analytic tool manage to forward/reverse engineer from both UML to code and back from code to UML without loosing any essential functionality or content.

In this paper we present an explorative study of four commonly used industrial UML-tools. The code used is a typical textbook based example of a web-based client-server solution. From the software UML-diagrams were generated by each of the four tools. The tools were then used to generate code (forward engineering) from the generated diagrams and then the code was reverse engineered back to UML. Each step was analyzed separately and the conclusions are presented in section 4. Somewhat surprisingly we found that Microsoft Visual Studio .NET 2005 beta, despite its limited support of UML diagrams, gives the best result for the application at hand.

2. EVALUATION

All evaluations were performed on a typical model web-application, implemented in C# under ASP .NET as a three-layer structure with a client-/server side and database (Faison, 2002). The application contains straightforward functionality such as billboards, ranking lists, handling of log in, editing of member lists etc. (Microsoft, 2003). With the exception of the main method being left out in order to see which tools that automatically added such a method during forward engineering, the detailed functionality of the application should be of no significant importance since all standard components of any regular web service should be easily modeled by UML. (Especially since web services is one of the standard examples for UML-modeling). Instead we are focusing on method I/O, handling of attributes, returned types and generated code because of the importance of the practical usability of any UML-tool. Differently put; the probability of a major failure in any acknowledged tool to properly model a basic web-service is considerably much lower than that of the same tool failing to generate the correct individual data-type or code.

2.1 Modeling, Reverse and Forward Engineering

The comparison of the different tools will be made based on the modeling of the chosen web application. A full UML-model will be created as far as possible with each and every one of the individual tools. The UML-diagrams are created by click-drag-and-drop, complemented with necessary information about methods and attributes based on the specification in Table 1. Potential lack of support in any tool for any functionality is also logged and especially noted.

In the second step, C#-code is automatically generated based on the UML-model created in respectively tool. The resulting code is evaluated based on which, if any, functionality is missing, how empty methods are handled and what error messages are generated. As noted above, the main() method is intentionally left out in the class diagram in order to examine if the tools automatically generate such a method or not.

Finally, to test the reverse engineering capabilities of each tool, we wrote a complete software skeleton in C# based on the generated full UML-diagrams for subsequent reverse engineering by each tool, generating “tool specific” UML-models of the code. Again we compared functionality against the “true” model and logged all error messages. Turning to the details in the model specification in Table 1 above, it should be noted that the number of classes has not been in focus, but the classes are instead mainly “holders” for the

methods, data and return types being investigated. The class “server” contains the return data type “DataSet” for instance, which requires a using directive normally not part of auto-generated code. The intention is to investigate if each tool automatically adds the necessary using directive or not. The same class also contains the attribute “ArrayList”, which in the same way is not a part of the most common using directives.

Table 1. The different classes, their attributes, methods, the in data and return types.

Class Server					
Method	Input data	Return Type	Method	Input data	Return Type
Login	String user, String pass	Int	btnApply	Object sender, EventArgs e	Void
selectSQL	String SQL	DataSet	btnShowRank	Object sender, EventArgs e	Void
updateSQL	String SQL	Int	btnShowCal	Object sender, EventArgs e	Void
Attributes			btnLogIn	Object sender, EventArgs e	Void
Data type	Name		Class About		
ArrayList	Classes		Class Help		
Class Client			Class Gui		
Method	Input data	Return Type			
Login	String user, String pass	Int			
Apply	String ID, String club	Int			
showRanking	String Class	DataSet			
showCalender		DataSet			

Standard data types such as String, Int and even object are part of all classes, however. These represent a selection of types that can be used without any additional using directives and are all possible to control in all the investigated tools during forward engineering (code generation).

2.2 Tools

The palette of tools chosen aims in no way to be complete, but to represent the most common and well-known tools. The selection might be debated quite harshly, but the idea behind the following selection is the (potential of) true usage in the software engineering community rather than technical idealism – there has always existed cutting edge solutions in labs, certain parts of the open source community or within highly specialized companies. Consequently we have chosen to include four rather well established candidates:

- Microsoft Visual Studio 2005 .NET beta. Visual Studio is a graphical, highly common graphical IDE with support for C, C++, C# and VB. (Microsoft, 2005)
- Rational Rose (Enterprise Edition 2003.06.12.180. 000), a specialized modeling tool for system architectures based on UML, included for manual UML-modeling and as “standard” only. Unfortunately Rose does not support neither forward nor reverse engineering support for C#. (IBM, 2005)
- PowerDesigner v11.0, a tool for modeling of data systems with the aid of UML. (Sybase, 2005)
- Visual Paradigm v2.1. VP SDE is made for UML-modeling and integration with Visual Studio, but lacked support for the 2005 version of Visual Studio. It was therefore tested together with the installation of MS Visual Studio 2003 .NET. (VisualParadigm, 2005a)

We believe that these tools make up a good selection of industrial and widely available alternatives for UML-modeling and code generation tools of today (2005).

3. RESULTS

The results section is divided into three subsections, each of the subsections focus on one of the three tasks outlined in Section 2; UML-modeling, UML-based code generation and Reverse engineering: UML-diagram from code support. All tests were performed according to the outline in sec. 2 without any unexpected problems or failures.

3.1 UML-Modeling

Class diagrams were created in all tools according to Table 1, through straightforward click-drag-and-drop functions. Methods, parameters and attribute were added using right-click functionality when it was available, or from appropriate menus. Since Visual Studio mainly is an IDE, does Visual Studio lack support for other UML diagrams than traditional class diagrams. There are possibilities to create other sorts of diagrams also in Visual Studio, but that lies outside the scope of this article. All other tools (see sec. 2.2) are designed to primarily be used for modeling, which made it possible to model all aspects of the test application according to UML 1.3 (Booch, Rumbaugh, Jacobson, 2000).

Visual Paradigm is the only tool that is integrated into an IDE (Visual Studio), which enables full scale modeling and programming within the same environment. Code generation is done in real-time in Visual Paradigm and Visual Studio, while Rational Rose and PowerDesigner create source code files that have to be processed in other environments. Rational Rose lacks, as mentioned in Section 2.2, support for forward engineering to C#, though.

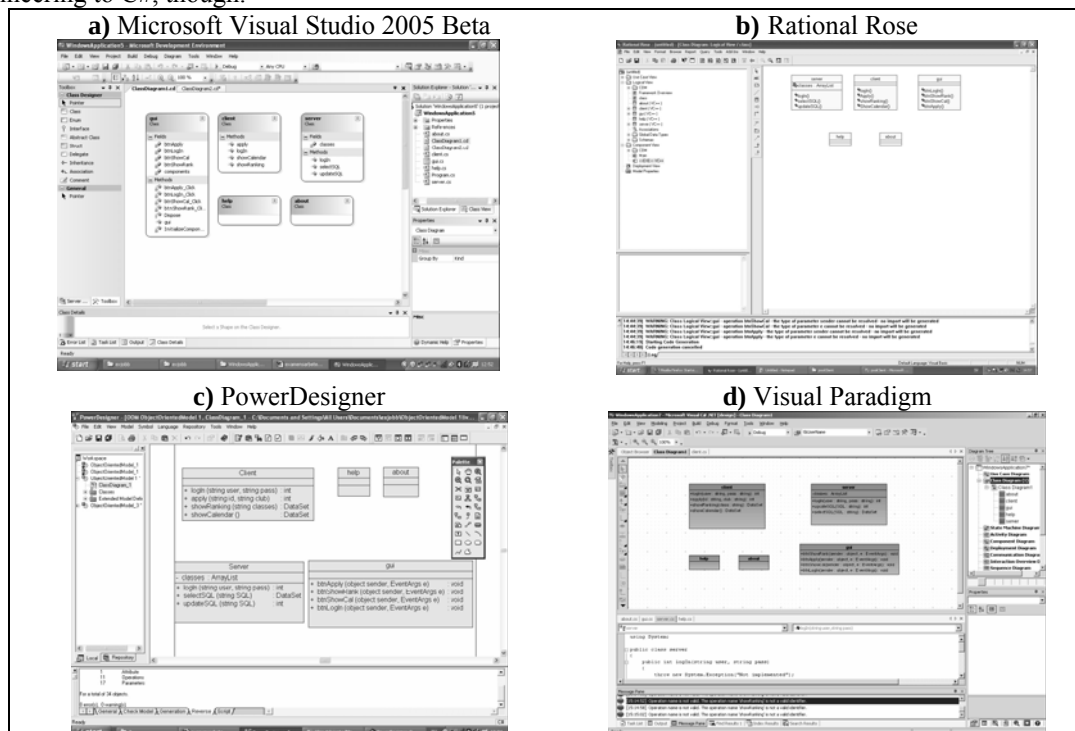


Figure 1. UML-Modelled class diagrams

3.2 UML-based Code Generation

All three tools (Visual Studio, PowerDesigner and Visual Paradigm) that supports C# generation, also correctly supported the standard data types in the generated code. The data types "DataSet" and "ArrayList" were treated as especially defined data types, however, i.e. none of the three tools added the using directives that were needed (see rows 1-4 in Table 2). Furthermore was MS VS the only tool that automatically generated a main method, which added the class "Program" that contained this method. No error messages were generated in any of the tools (see row 6 in Table 2). Furthermore, empty method bodies were treated in essentially two different ways. Visual Studio and Visual Paradigm added: "throw new System.NotImplementedException()", VirtualParadigm SDE added: "throw new System.Exception("Not implemented")" while PowerDesigner only added the comment: "//TODO: Implement"

On the other hand was PowerDesigner the only tool that generated Return statements in the empty methods. In methods with return data type Int a Return 0 was generated. While methods that used the

especially defined return data type “DataSet” got the return expression Return null (see row 8 in Table 2). In Tables 2 & 3 in those cases when forward (reverse) engineering has not been possible at all for certain functionality, we have recorded a N/A (not available) value. When the tool could not handle the data types in question we chose to record the incident as “Esp. def.”.

Table 2. The results from forward engineering

Test/Tool	MS Visual Studio	PowerDesigner	Visual Paradigm
Standard data types	Supported	Supported	Supported
ArrayList	Esp. def.	Esp. def.	Esp. def.
DataSet	Esp. def.	Esp. def.	Esp. def.
Main()	Generated	Not Generated	Not Generated
Empty Methods	Not Implemented Exception	//TODO: Implement	Not Implemented Exception
Return statements	Not Generated	Generated	Not Generated
Using Directive	System System.Collections.Generic	System	System
Error Message	No	No	No

3.3 Reverse Engineering: UML-diagram from Code

Turning to the generation of UML-diagrams from C#-code, we noticed that Visual Paradigm could not perform the UML-generation due to an internal error, preventing the generated UML-diagrams from being saved correctly to file. This particular problem is addressed in the bulletin board on VP:s web site where several users refer to the same problem. VP SDE have not been able to present a solution to the problem which leads to the conclusion that VP SDE at the moment not (fully) supports reverse engineering (VisualParadigm, 2005b). This, together with Rational Roses incapability of handling C# code, unfortunately made it impossible to compare VP SDE and Rational Rose with MS VS and PowerDesigner.

As for the two remaining tools, we notice that MS VS presents a correct class diagram after reverse engineering. Since the class “gui” has “modifier partial” PowerDesigner was not able to handle the generation of that class in the class diagram. Although it should be mentioned that an error message were generated stating that PowerDesigner could not handle the modifier partial (see row 5, Table 3).

Table 3. Results of reverse engineering and comparison to the “ideal” UML-diagram.

Test/Tool	MS Visual Studio	PowerDesigner	Visual Paradigm
Standard data types	Supported	Supported	N/A
ArrayList	Esp. def.	Esp. def.	N/A
DataSet	Esp. def.	Esp. def.	N/A
Return statements	Not Generated	Generated	N/A
Using Directive	System System.Collections.Generic	System	N/A
Error Message	No	Yes	N/A

The same problem occurred with the class “settings” in the package properties. Nevertheless, the class “resources” in the package properties which lacks modifier was included in the UML-diagram. Apart from the above mentioned problems all classes, methods and attributes were modeled correctly. The only error messages were those stating that PowerDesigner could not handle the modifier partial (see row 5, Table 3).

4. CONCLUSIONS AND DISCUSSION

The modeling tools that have been tested show, as expected, significant similarities in the resulting code and UML-diagrams. The greatest differences are those connected to the individual tool’s main focus. MS VS is mainly a software development environment and as such does it lack many of the sophisticated modeling capabilities proper modeling tools have. MS VS 2005 .NET beta is also the first version of VS that encompass any modeling capability, and it only supports class diagrams (other modeling capabilities does not follow the UML standard and lays outside the scope of this article). Visual Paradigm differs from the main stream modeling tools in that sense that it is integrated with MS VS (2003), which enables real-time code generating. The UML-modeling itself is quite similar to the workflow in traditional modeling tools as e.g. Rational Rose, however. In that sense PowerDesigner is a more “clean cut” modeling tool.

Among other things MS VS the only tool that actually adds the necessary `main()`-method. It also enables real-time code generation within the same tool. It is all the more surprising that such a development oriented environment, as MS VS is unable to recognize data types that require additional using directives. Possibly could this drawback be attributed to the fact that the tested version is a beta-version, but it is none the less a more then noticeable drawback.

PowerDesigner formally has full support for all diagram types, and full forward and reverse engineering, but could not handle reverse engineering of the test application at hand. Concerning forward engineering, PowerDesigner does not add the necessary `main()`-method, but does as the only tool in the study add return statements in the empty method bodies. For the return data type `Int`, “Return 0” is generated, however, and the “DataSet” type gets the return statement “Return null”. PowerDesigner were also unable to handle the partial modifiers for the classes “gui” and “settings”.

Unlike PowerDesigner and Rational Rose is Visual Paradigm integrated with the IDE (MS Visual Studio 2003 in this case), which enables real-time code generation in a way similar to that of Visual Studio. This enables an improved work flow compared to PowerDesigner, but just like Visual Studio does Visual Paradigm lack to add return statements in the empty methods. Neither does Visual Paradigm add the necessary `main()`-method. In addition there is an internal fault in the tool which makes reverse engineering impossible. The latter problem unfortunately makes the tool too unreliable to make it practically viable in any iterative software development process until the bug is fixed.

This makes Microsoft Visual Studio .NET 2005 beta, despite its limited support for UML diagram types and drawbacks in code generation the overall most suitable tool for UML-modeling and forward/reverse engineering for the type of standard application considered. The most notable conclusions, however, is not the most limited tool being the most useful, but rather that no tool managed to generate proper code for the considered standard textbook example. The subject of development of UML-based code generating tools is thus far from closed neither in an academic nor industrial sense.

REFERENCES

- Baxter, I.D., Mehlich, M., 1997. Reverse Engineering is Reverse Forward Engineering, IEEE Proceedings of the Fourth Working Conference.
- Booch, G., Rumbaugh, J., Jacobsen, I., 1999. The Unified Modeling Language user guide Addison-Wesley Longman Inc., Reading.
- Booch, G., Rumbaugh, J., Jacobsen, I., 2000. The Unified Modeling Language User Guide, Pearson Education, Upper Saddle River.
- Code Generation Network, 2005. Generators, <http://www.codegeneration.net/generators.php> (2005-06-29).
- Faison, T. 2002. Component-Based Development with Visual C#, M&T Books, New York.
- Fowler, M., Scott, K., 2000. UML distilled, 2nd Edition Addison-Wesley, Harlow.
- Ekberg, L., Mankefors-Christiernin, S., Boklund, A., 2004. Software Code Quality with UML Design Models, Proceedings of the 4th Conference on Software Engineering Research and Practice in Sweden, Linköping, Sweden.
- Holt, J., 2001. UML for systems engineering – watching the wheels, The Institution of Electrical Engineers, London.
- IBM, 2005. IBM Software – Rational, <http://www-306.ibm.com/software/awdtools/suite/> (2005-05-25).
- King, M. J., Pardoe, J. P., 1992. Program Design Using JSP - A practical introduction, Macmillan, Basingstroke.
- Microsoft, 2003. Enterprise Solution Patterns Using Microsoft .Net: Version 2.0 : Patterns & Practices, Microsoft press.
- Microsoft, 2005. Microsoft: Visual Studio 2005 Beta Home, <http://lab.msdn.microsoft.com/vs2005> (2005-11-22).
- OMG, 2003. OMG Unified Modeling Language Specification, <http://www.omg.org/cgi-bin/doc?formal/03-03-01.pdf> (2005- 10-28).
- Sommerville, I, 2001. Software Engineering 6th edition. Pearson Education Limited, New York
- Sybase, 2005. 8 Sybase Inc, <http://www.sybase.com/products/developmentintegration/powerdesigner> (2005-05-25).
- Whalen, M.W., Heimdahl, M.P.E., 1999 On the Requirements of High-Integrity Code Generation, Proceedings of the 4th IEEE International Symposium on High-Assurance Systems Engineering, Washington.
- VisualParadigm, 2005a. Visual Paradigm, <http://www.visual-paradigm.com/product/sde/vs/> (2005-11-25)
- VisualParadigm, 2005b. Visual Paradigm, <http://forums.visual-paradigm.com/posts/list/84.html> (2005-11-19)