Evaluating OO-CASE tools: OO research meets practice

Danny Greefhorst, Matthijs Maat, Rob Maijers
{greefhorst, maat, maijers}@serc.nl

Software Engineering Research Centre - SERC
PO Box 424
3500 AK Utrecht
The Netherlands

Abstract

Current object-oriented CASE tools are useful for recording and gaining insight into OO models. They offer extensive support for especially the analysis and design of object-oriented software. The possibility to generate skeleton code motivates development teams to construct a good design before coding. Developers subsequently add the remaining code; CASE tools then offer support for keeping model and code consistent. Extensive possibilities for maintaining models and producing documentation are also provided. Contemporary CASE tools thus provide for more efficient communication within development teams and therefore facilitate overall maintenance.

Support for UML is widely spread. This standard offers a notation, not a process: no clues are provided on how to think and work in an object-oriented fashion. CASE tools do not fill this gap; the tools do not help developers decide when to take which steps to create a design. As for creating a good design, tools lack active support for recording and reusing design knowledge and experience. Developers thus still have to rely on their own instincts to develop good OO designs.

This paper describes the results of evaluating four leading OO-CASE tools, based on requirements derived from current object-oriented modeling research and practice.

1. Introduction

Turning problems into working systems is part of the craft of Software Engineering. Although in the last decades much has been accomplished in developing software and supporting software development, still no recipe exists for creating flexible, maintainable systems. Lack of focus on architecture (structural aspects such as reusability of components, flexibility and maintainability) is to blame for this.

One way to raise the level of software development is to shift focus from the implementation to the specification of systems. Using standard architectures, patterns, components and code generation, the development process could be improved [3]. Therefore, incorporating the use of such abstractions capturing proven modeling knowledge and experience into the system development process is crucial, as is visualized in figure 1.
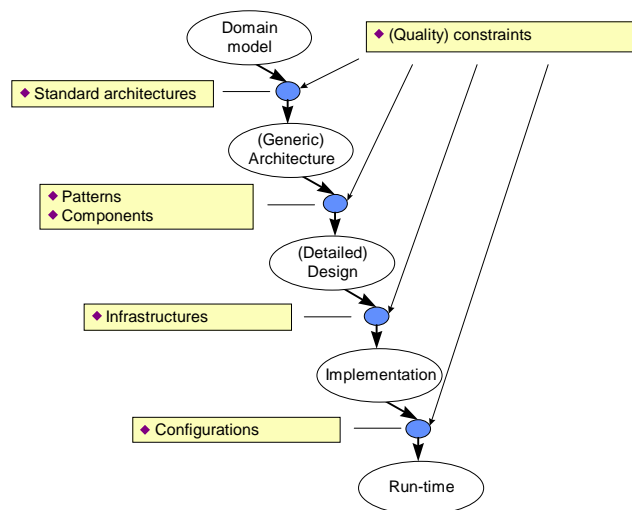

Figure 1 - Developing software using standard solutions

Software development takes places by finding a fit between the problem to solve, a standard architecture to use and an infrastructure. The solution is first described at a high level of abstraction by parameterizing or instantiating a standard architecture. In the next step this architecture design is translated into a concrete design, using standard components or patterns wherever possible. A working system finally is obtained by translating the design to an implementation for a certain infrastructure and a specific configuration. Usage of standard solutions at all levels of abstraction is crucial for guaranteeing quality. For now, experience seems to be the only way to determine whether a solution is 'good' or not.

This view on object-oriented modeling has consequences for the automated support [2] of object-oriented development; OO-CASE tools should not ignore the importance of standard solutions, crucial for OO modeling. Even more, OO-CASE tools should really aid in development, not hinder: more well-known requirements with regard to CASE in general also apply to OO-CASE tools. These requirements can be grouped in five main themes:

- architecture, patterns and frameworks
- development process and methods
- roundtrip engineering
- metrics
- usage aspects

With these five themes in mind, four leading OO-CASE tools (selected by visibility in the Dutch market, focus on analysis and design and availability for a wide range of target implementation environments) have been evaluated in order to determine the current state of the art in automated object-oriented modeling and development.

This paper describes the results of evaluating ObjectTeam (Cayenne Software), Paradigm Plus (Platinum Technology), Rose (Rational Software Corporation) and SELECT Enterprise (SELECT Software tools). A compressed version of the ambitious framework of requirements for OO-CASE tools used in the evaluation is presented, along with the support CASE tools currently offer.

2. Architecture, patterns and frameworks

The design of object-oriented software is not trivial. The problem lies in identifying the right classes and assigning them the proper responsibilities. Knowledge and experience are the only means for attaining a qualitatively good design. Stimulating reuse of standard solutions for common problems is therefore a necessity. Ideally, CASE tools offer active support, or even stimulate to reuse these solutions.

Standard solutions like architectures, frameworks and patterns consist of a solution structure and guidelines for their application. These solutions can be applied at various levels of abstraction. Standard architectures describe the structure of a system by decomposing it into components, roles and relationships. At a somewhat lower level of abstraction, patterns describe parts of a system as consisting of cooperating classes. A framework offers a partial implementation that can be supplemented or parameterized to easily develop a specific system.

UML [1], and specifically CASE tools, offer no active support for working with abstractions like architectures, patterns and frameworks. Although abstractions can be described and reused as UML concepts, active support

such as automatic recognition of abstractions and keeping them consistent is missing.

The concept of architecture is only available in Rational Rose and SELECT Enterprise as a graphical dividing line in a component or class diagram. The concept is missing completely in the other tools. Pattern support is also very limited. In particular, there is a third-party add-in available for Rose allowing definition and application of self-defined (or standard Gamma [4]) patterns, in the latter case by choosing from a list of available patterns. There is a predefined catalog of patterns available for Paradigm Plus, that can be applied using the scripting language. In both tools, recognizing the patterns and keeping them consistent is the responsibility of the designer.

Support for providing insight in existing reusable (partial) solutions, such as provisions for cataloging and searching them, is also not available. It is possible to archive a set of diagrams for later reuse, but it is up to the designer to locate the proper diagrams. SELECT Enterprise offers integration with the SELECT Component Manager that does support the suggested functionality at the component level.

3. Development process and method

Software development, object-oriented or not, consists of a number of steps ranging from requirements analysis, domain analysis, design, implementation, test to exploitation. The results of these steps are described in models that are interrelated. Logically these models can be considered as different views on the same underlying model. From this viewpoint every view, from analysis model to code, offers the developer a different look at the same software.

UML offers a standard notation for describing the various views, but does not describe the process of developing them. This leaves many questions unanswered, like: what is the next step to take? What are the relevant concepts within this step and to what extent should they be explicated? The reviewed CASE tools also lack such (active) support for a process; in most cases they only supply hypertext based documentation describing it.

Although views offer different viewpoints on a single model, it can be valuable to freeze models that are the result of a process step. ObjectTeam offers the most elaborate support for this concept by allowing steps and the accompanying models to be frozen and compared with one another. In other tools, views can only be represented by raising diagrams to a view status. No means are available for relating or switching between views.

CASE tools focus on the analysis and design steps, but can be integrated with specialized requirements analysis tools and development environments to cover most of the

process. The latter type of integration mostly relies on generation and reverse engineering of code and database definitions. Other development activities such as documentation generation, version/configuration management and project management are realized my means of dedicated interfaces to other tools. Only Paradigm Plus keeps conforming to the CDIF standard (CASE Data Interchange Format). GUI design is not supported in any of the tools. Version control can be performed at the class level in ObjectTeam, which includes an integrated version control system. Other tools share constructs on a more coarse grained level. Only ObjectTeam automatically integrates the source code in the version control system.

Although the CASE tools, excluding SELECT Enterprise, all provide a choice between various notations (i.e. OMT, Booch or UML), the focus is clearly on UML. UML 1.1 support is either currently available or expected soon. Although UML is an official standard, the tool suppliers have chosen to implement it only partially, thereby leaving out some diagrams and other notation constructs. On the other hand, other models that are not based on UML are included, for example to describe business processes or database structures. It is not possible to modify the underlying meta-model, at least not within the tools themselves.

It is difficult to get a grip on the dynamic aspects of a system during design. How do objects collaborate in executing a certain task? It is possible to describe the dynamic aspects of systems in UML using, for example, sequence, collaboration and state diagrams. SELECT Enterprise can simulate this dynamic behaviour by animating the messages that are exchanged between objects.

4. Roundtrip engineering

Object-oriented CASE tools could assist in switching between views and in deriving views. The only views that the evaluated tools can derive, are parts of the implementation: code and (sometimes using add-ons or other products) database schemes. Code generation provides a skeleton with declarations for classes, attributes and methods in which the developers only need to fill in the bodies of methods. The information necessary for code generation comes from the class diagrams. Some tools also use component diagrams; the other diagrams are not at all used for generation.

Language-specific properties of classes, attributes and methods introduce a language dependency in the model. Unless abstract data types are used, further language dependencies are caused by the attribute types necessary for generation, which are specified in class diagrams. In Paradigm Plus and SELECT Enterprise, which allow abstract data types, the developers need to eventually map the abstract types to language-specific types. During a session with the tools, it is possible to switch to another implementation language by adjusting the attribute types or the mapping to the chosen language.

The evaluated tools can also perform reverse engineering in the sense that they can derive class diagrams from source code. It is also possible to derive database models (not part of UML) or class diagrams from existing databases. Generation as well as reverse engineering is adaptable.

Method code is not represented in the tools; implementation is not a view on the model. This causes inconsistencies between the diagrams and the source code when the developers try to read adjusted generated code back into the diagrams or try to regenerate from adjusted diagrams (roundtrip engineering). For instance, when a developer changes the name of a method in a diagram and regenerates, most tools insert a new method from which previously written code is missing. Rose avoids this problem by including identifying numbers in the comment of generated methods. SELECT Enterprise provides insight into this problem by simultaneous visualization of model and code (Java Synchronizer).

5. Metrics

During the software development process, measurements can be gathered to support the developers in making qualitative statements and predictions about the design and the product. These metrics [5] can also be helpful in making the OO projects predictable and manageable. To measure is to know: making well-founded statements regarding predictability, quality and productivity only becomes possible if project data has been recorded and interpreted systematically. CASE tools potentially are the ideal measuring environments, because a lot of concrete data on project and design is already recorded.

In practice, the evaluated CASE tools provide only moderate support for determining metrics. Paradigm Plus and Rational Rose can report statistical information, but automatic interpretation is not supported. SELECT is the only tool supplier offering more intelligent support by means of a separately available tool called Estimator. This tool uses the model in SELECT Enterprise to make predictions on the expected time needed for implementation. Estimator does, however, not reveal the underlying calculations, which are based on experience figures. Therefore the results seem to be like 'magic'.

Active support for metrics is missing in all of the evaluated CASE tools. There are, for example, no mechanisms that can warn a developer when filling in or changing the model causes certain metrics to reach critical

values. When the number of relations between two classes becomes too high, generally a bad sign, the tools do not draw the developers attention.

6. Usage aspects

The user interfaces of the evaluated tools are suitable for handling complex models and large diagrams. It is possible to hide information in the diagram, for example attributes and methods of classes. During the reverse engineering of large quantities of classes, the developer can increase the comprehensibility by (possibly automatically) distributing the classes over a number of diagrams. Furthermore, it is often possible to include references to other (sub)diagrams, which keeps large diagrams comprehensible. In Paradigm Plus, the developer can perform similar operations on large models in a fast and simple way by means of a relational table representation of arbitrary design elements and their relationships.

Apart from Rose, all tools have a repository with which multiple developers can work simultaneously. All repositories can be accessed outside the tools using an open interface or simply because the tool uses a database.

Rose uses files to store model information. The developers can share this information by dividing it into packages, the unit of reuse and authorization. This means that changing a package can be done by one developer at a time, whereas in the other tools the only constraints are that only one developer at a time can change the layout of a diagram and that two developers can not simultaneously modify the same diagram element. In the latter, the design elements (classes, use cases, etc.) are the unit of reuse and authorization.

The evaluated tools can all be extended to a high degree. Rose and SELECT Enterprise can be extended using an OLE interface and Rose is also adaptable using Rose script. For Rose, a lot of additional functionality is already available as an add-in. In Paradigm Plus, all functionality is accessible by means of scripting, from the repository down to diagrams. In ObjectTeam the menus and properties of diagram elements are adaptable without scripting. By scripting in Tcl/Tk, the complete functionality is adaptable, with the exception of diagrams.

The evaluated tools are suitable for larger projects as, apart from the usage aspects discussed, clients and servers of a variety of platforms can be mixed in the tools. Only Rose will be less suitable for large teams, due to its limited provisions for cooperation.

7. Conclusion

Measured against the ambitious framework presented in this paper, the individual CASE tools are not full-grown yet (see table 1). Especially in the area of more active and intelligent support of the various, difficult, aspects of OO analysis and design, the tool suppliers have many challenges to face. Knowledge and experience still reside only in the heads of developers, not in the tools.

		Absent	Possible	Extensive support
Architecture	• Design using standard solutions		✔	
	• Reuse		✔	
Development process	• Different views on model	✔		
	• Support for process	✔		
	• Integration with other tools			✔
	• UML notation			✔
Roundtrip	• Code generation		✔	
	• Reverse engineering		✔	
	• Roundtrip		✔	
	• Code as view on model	✔		
Metrics	• Gathering measurements		✔	
	• Active use of measurements	✔		
Usage aspects	• Team development			✔
	• Controlling model complexity		✔	
	• Extendibility of tools		✔	

Table 1   State of the art regarding OO-CASE tools

	Architecture		Development process		Roundtrip	Metrics	Usage aspects		
	Views	Patterns	Model simulation	Integrated version control	Synchro-nization	Prediction	Scripting power	Number add-ins	Relational manipulation repository
ObjectTeam (Cayenne)	✔			✔			✔		
Paradigm Plus (Platinum)		✔					✔		✔
Rose (Rational)		(✔)					✔	(✔)	
SELECT Enterprise (SELECT)			✔		(✔)	(✔)			

Table 2 Notable features of current OO-CASE tools (brackets indicate support is realized by means of a separately available product)

The conclusion that the CASE tools are immature is, however, a false one; the CASE tools can be very valuable in the development process. All of the tools are mature environments with extensive support for analysis and design of object-oriented systems. They are pre-eminently suitable for maintaining and communicating designs, also in large development teams.

CASE tools are being improved constantly, also towards the more ambitious parts of the sketched framework. For example, in the area of patterns, metrics and views, the tools occasionally offer support that goes beyond the passive recording of models (see table 2). Combined in one tool, these distinguishing properties would be a great step forward in computer-aided support of object-oriented software development. For the time being, however, the ideal *active* CASE tool still seems far away.

8. References

[1] H.-E. Eriksson and M. Penker, *UML Toolkit*, John Wiley & Sons, 1998.

[2] A.S Fischer, CASE — *Using Software Development Tools*, John Wiley & Sons, 1991.

[3] G. Florijn, "Software engineering en -architectuur", *Informatie*, januari 1997, pp. 32-39.

[4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns*, Addison-Wesley, Reading, 1994.

[5] B. Henderson, *Object-oriented Metrics*, Prentice Hall, Englewood Cliffs, 1996.

9. Justification

The authors are employed as researchers at the Software Engineering Research Centre — SERC. The evaluation results presented in this paper originate from an evaluation SERC has performed in collaboration with the suppliers of the evaluated CASE tools (Cayenne Software, Platinum Technology, Rational Software Corporation and SELECT Software tools). Using the framework presented in this paper, potential end-users were invited to examine a CASE tool by solving a real-life case (using Java and Forté) in a small development team, under supervision of SERC and the CASE tool supplier. A more detailed description of the framework used and the specific results concerning all four CASE tools is available from SERC.

10. Contact information

Software Engineering Research Centre

PO Box 424
3500 AK Utrecht
The Netherlands
Voice: *+31 - 30 2 54 54 12*
Fax: *+31 - 30 2 54 59 48*
E-mail: *info@serc.nl*
URL: *http://www.serc.nl*

Contact person: *Nanne Nauta (nauta@serc.nl)*