How to Test and Compare CASE tools

Tapani J. Kinnula
Jalal Matini

Swedish Institute for Systems Development
Box 1250, S-164 28 KISTA, Sweden
Januari 1989

Abstract

This paper describes how a practical study of computerized tools for systems development could be prepared and made. The paper suggests the use of a detailed checklist combined with a description how the practical evaluation of CASE tools should be executed. The purpose is to ensure that the evaluations of different CASE tools will be made in similar ways and that results can easily be compared when selecting a strategy for computer aided software engineering.

1. Introduction

The complexity of design of computer systems is increasing. Practically all efficient corporations are moving to a higher level of automation in their data processing. This implies higher requirements on the computer systems being developed. Consequently, the need for more efficient methods and tools for developing and maintaining computer systems is increasing. During the last few years, a large number of tools for analysis and design of computer systems have been born to daylight. These tools, called CASE tools, intend to automate parts of the systems development process. There is, however, a great variety among CASE tools today. Some of them are merely graphic editors with a simple database for saving the design information, often can these specifications be transferred to 4GL-tools or database systems for further design and implementation. Some tools, in turn, are sophisticated workbenches with a large amount of automation in analysis and design processes and with possibility of generating code from design specifi-cations.

An important factor related to CASE tools is the method (or methods) which the tool supports in systems analysis and design. An enterprise may have to decide whether to invest in a method-specific CASE tool and learn the method it use, or whether to choose a tool that can be customized to support the method used by the enterprise. To make this important decision, detailed information about the tool and supported methods is needed. Further requirements for specific functions, as prototyping, code genereration and interfaces to other products, must be considered when choosing a tool. Of course, there are other aspects too when selecting a CASE tool, but they are beyond the scope of this paper. The purpose of this paper is to describe how reliable and detailed information about CASE tools can be captured by studying and assessing them in practice.

1.1 Justification

There should be no doubt that the best way to learn a CASE-tool is to use it in practice. Reading information from vendors and developers may form the basis of preliminary selection, that is, make it clear what tools do not satisfy the basic requirements. However, to choose the right tool, detailed information is needed about several tools, their advantages and the problems that can appear when using them in practice. Putting a CASE tool into practice is an efficient way to discover the practical problems and how serious they are. In particular, when studying several tools in practice in similar ways, important experiences for comparing the practical values of these tools are obtained.

The purpose of a practical study is to obtain experiences and relevant features of the studied object and to formalize the results. The resulting report forms a basis of the final assesment of the studied object. When studying CASE tools, the report should provide information about all important tool features and their role in practice. Detailed and identically structured reports on several tools provide a good basis for comparing tools and making the final selection.

Of course, preparing and executing these practical studies of CASE tools is an expensive and time consuming task. Many enterprises are confined to put up with vendor information and experiences from other corporations, when making their decision. User experiences are, however, seldom written down and formalized in order to make them easily accessible for other people. Therefore, well formed reports on practical studies and evaluations of CASE tools provides an important way of getting valuable and detailed information about the tools from the practical point of view.

To make studies of different CASE tools and resulting report as equal as possible an inspection protocol could be used. An Inspection protocol is a kind of check list of tool attributes including the description of how these attributes should be interpreted and studied. Using the protocol during the practical test work and report writing ensures well structured and comparable reports that can be used in the selection process. Figure 1. illustrates our approach for testing and comparing CASE tools.

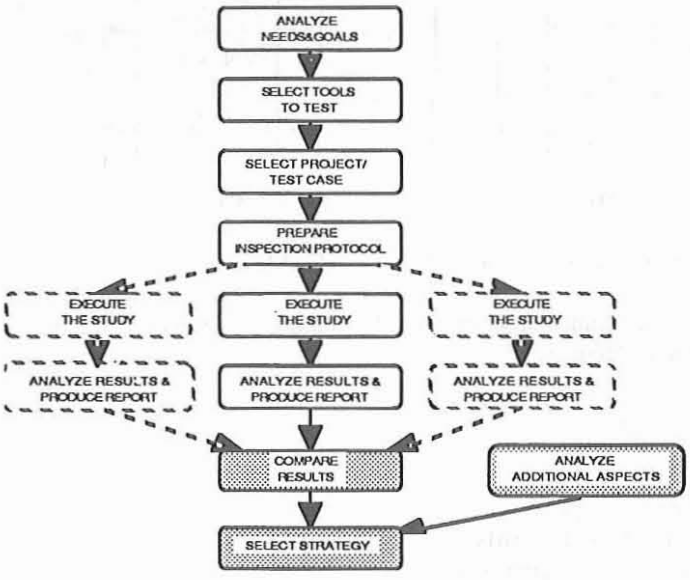

Figure 1. Testing and comparing CASE tools. Several tools can be tested simultaneously and the resulting reports form a basis for selection of strategy for CASE technology. Note, that the grey activities indicate activities of the enterprise, while the white activities could be performed by independent institutes or other enterprises as well.

2. Preparing a Practical Study

Before getting into work and studying CASE tools, some preliminary work should be done. The first thing is to identify needs and goals: what are the problems with the current systems development environment? How to solve these problems? what goals do we want to achieve, now and in future? When the needs and the goals are known it is easier to decide which tool type is suitable, and how to form tool studies.

When studying a CASE tool, it is necessary to know what features and lacks to look for. The requirements for the tool should be analyzed and the related tool features should be consedered when forming the inspection protocol and applying it to the tool. This ensures that no relevant aspects will be omitted during the practical study The protocol also provides a well-defined structure for the resulting report.

2.1 Identifying Needs and Goals

To be able to decide what to do, one has to know what's wrong. The most usual reasons to buy a CASE tool concern productivity in systems development and the quality in resulting systems. Knowledge about why the productity is too low and why the systems quality is not good enough is necessary when selecting a strategy for CASE. For example, implementing a method with many weak points in a CASE shell will not succeed without profound and costly improvements of the method. In the following we present some questions of concern when analyzing needs and goals.

- what are the current problems
- when and how does the problems appear
- what are the basic causes for the problems
- what are the consequences of the problems
- how can these problems be eliminated
- how can a CASE tool solve the problems
- what else do we want to achieve with a CASE tool

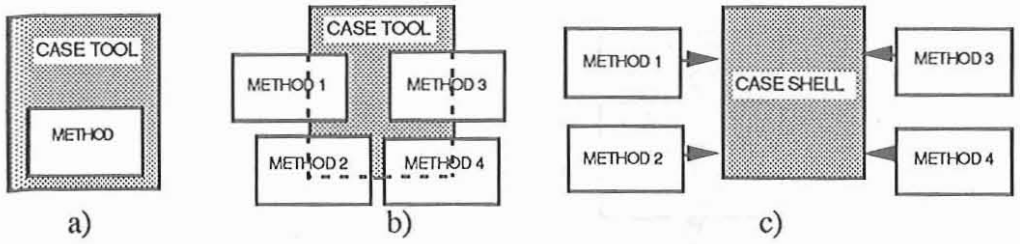

Fig 2. Different types of CASE tools. a) method-specific CASE tools have built-in methods, b) a CASE tools supporting several methods, but none of them completely, c) a CASE shell has no built-in method support, but allows implemention of different methods and method specific techniques.

- what kind of CASE tool is suitable
- what are the basic demands for the tool
- what methods should the tool support
- what specific features are desired currently
- what features are needed for future requirements

Answers to these questions provide a foundation for a preliminary evaluation of CASE tools. Tools not satisfying the basic requirements can be detected and omitted from further consideration. There are lots of brief analyses and surveys of commercially available CASE tools that can be helpful in this stage.

Analysis of current problems may result in discovery of serious lacks in the currently used method, which indicates a need of a more efficient and formalized method. Which metod to choose depends on various things, but it should be pointed out that methods provided by specific tools often have a great priority. Some tools have strong connections to well-defined and formalized methods and therefore support a great deal of automation in different phases of systems development. It may be appropiate to examine tools in this class when new and efficient methods are needed. On the other hand, if the method used earlier turns out to be efficient and well defined, it is convenient to examine the possibilities of implementing it within a CASE shell.
Problems or special needs that do not have their origin in the method or that are related to certain kind of applications should also be taken into consideration when selecting and reducing the set of potential tools.

2.2 Test Case or Real Case?

There are to possibilities to evaluate a CASE tool in practice: a more or less ideal test case can be used, or the tool can be tested within a real systems development project. Which strategy to choose depends on available time and resources. It might be of advantage for an enterprise to have well structured reports based on test cases at hand when they need to make a quick decision or do not have the necessary resources to test tools within real projects. However, most enterprises cannot test a number of tools by themselves, no matter whether using a test case or a real case.

Test cases seem to be most convinient to be used by consulting enterprises or institutes that are specialized in testing tools for systems development in order to publish the results. Test cases also serve the simplest way of testing tools and tool prototypes for researh purpose. They can be especially formed to test specific features like expressibility (that is, how many types of details and constraints can be expressed by using a tool's description techniques). Testing tools with synthetic test cases is easier and can be carried out at any time, by contrast with real projects. Putting tools on real and heavy cases is however the most reliable way of getting information about the practical value of tools and methods when used in a specific environment. An other aspect that argues against test cases is that they are already conceptualized: conceptualization of the physical and logical environments is one of the main problems in systems development- and one of the main reasons to use CASE tools, while the

complete "case description" for a test case is nothing else than a conceptualization of a virtual environment. Evaluating CASE tools in real projects requires both financial and time resources, but selecting the right tool, and even more important, the right method, may result in considerable savings in productivity and systems quality. Therefore, enterprices with many heavy projects and complex computer systems probably serve their purposes best if they choose to put the most potential tools on real systems development projects when evaluating them in practice.

When using a synthetic test case, some important aspects should be taken into consideration. A good test case is not too comprehensive in time and effort while being characteristic and wide enough to cover the most important features on a specific area. What test case to select (or design) depends on the type of subject area and what features and problems to examine. The test case ought to be well-defined and described in great detail to allow people that do not have possibilites to interview the "real users" to be able to implement it in reasonable time. Unclear and ambiquous descriptions should be avoided. In real projects much time and effort are spended to solve problems with ambigous and unclear descriptions and modelling different kind of solutions to these problems. It is not relevant to burden a test case with these problems, because the are not "productive" enough with respect to consumed time and obtained results.

The target environment of the system and the system type should be reflected by the test case. For example, a test case specifying a database-oriented information system is not particularly suitable when testing tools for real-time control systems. The choice of the test case is of great importance in order to detect the right features in the right environment.

It is often possible to choose an existing test case and use it with only slight modifications. There are several test cases that are frequently used when testing and evaluating methods for systems development. These test cases are most often appropiate also for testing and evaluating tools for systems development. Of course, it happens that a new test case must be designed from the beginning, but it is probable that there are "real cases" that can be simplified and modified without too much effort.

2.3 Inspection Protocol

The inspection protocol presents and treats all relevant aspects of concern when studying a CASE tool. These aspects are normally related to tool attributes and how they should be treated when evaluating the tool or writing the report. The purpose is to provide a uniform and well-defined way of executing the study and equal treatment of tools in the resulting reports. Reports based on equal inspection protocolls have equal structure and organisation which make it easy to assess tools and compare them with each other. More specially, the features of particular interest, when making the final selection of the tool, can easily be picked upp and examined in detail. A carefully accomplished execution of the protocol also provides a means of indicating the weak points of the studied tools which helps developers to find out possible improvements to be carried out in future releases.

We have emphasized that the test case should reflect the target environment and particular problems of interest. That is valid for the protocol too. The protocol should reflect the tool type and treat corresponding features. In other words, the protocol is not general in the common sense, it is more or less tailored for different types of CASE tools. Of course, one could use a general and detailed protocol valid for all kinds of tools and only treat the aspects that are relevant for the current tool.

In the following text we present the protocol we used when studying CASE tools DEFT™ and IEW™. The protocol is formed for database-oriented informations systems and does not treat aspects like those of control systems or real-time applications support. The protocol is not especially appropiate for studying CASE shells either. However, it can easily be augmented with new type of aspects and attributes. Note that the tools attributes are not always given explicitly, but as questions or in descriptive way in order to avoid ambiquities and unnecessary confusion.

The inspection protocol

Terms like 'diagram', 'representation technique' and 'description technique' are usual in the following text. Normally, 'representation techniques' and 'description technique' include the term 'diagram', that is they are more general and include also different kinds of textual representations. For simplicity, however, we often use the term 'diagram' for all kinds of description techniques.

A. Presentation

- Give a brief description of the tool
  - developer and vendors
  - environments (PC, Macintosh, Mainframe, workstations, networks)
  - tool architecture (i.e. the modules/components that the tool consists of and main task for each of them)
  - tool type : method-specific, CASE shell, no specific methods

Note: no details here, just a brief orientation about the tool

B. Methods and techniques

B.1 Coverage

- What phases and stages of the systems devlopment process are covered by the tool?
- Motivate and exemplify your conclusion about the coverage (schematic figures may be very illustrative)

Note: most commonly used division: planning, analysis and specification, design, implementation, testing, installing and maintenance)

B.2 Techniques and diagramming tools

- Describe and explain the existing diagramming or description techniques
  - main usage
  - syntax and constraints (e.g. JSP syntax)
  - how syntax and constraint checks performed are performed
  - power/expressiviness of the description techniques
  - customizability (e.g modification of drawing symbols or augmenting the tool with new symbols)
- Illustrate with figures and examples of diagrams and suchlike

B.3 Integration between diagramming tools

- Describe and explain different types of connections between the diagrams
- Are the connections dynamic, that is, does a modification of a diagram imply related modifications in other diagrams?
- How useful are the connections in practice? (e.g you need not to specify objects more than once)
- can new diagrams be opened from the current diagram (e.g to allow more detailed specifications of a design object)?
- Can specifications be transferred (shared) between different types of diagrams or representations?
- Does the tool support automatic redrawing of the transferred (or shared) objects?

B.4 Method support

- What methods does the tool support?
- If the tool is customizable, what methods and techniques are supported or can be choosen
  on delivery?
- How method-dependent is the tool, or in other words, is the tool dedicated to a particular method (or methods) and does not allow any departures from the method?
- The systems development process is often divided into phases, and different methods often support different phases. If possible, describe how the supported methods and description techniques are related the the phases of systems devlopment process
- What are the consequences of the degree of method dependence? For example, a strong method dependence may limit the user's possibilites to express or model specific features
  and constraints. On the other hand, a strong method dependence may force the users to a formalized and efficient way of working and therefore indirectly improve productivity and systems quality.
- Does the tool dictate or constrain the way of using the provided description techniques (e.g. to allow proper analyses or efficient code generation)?
- Are accessible analysis and checking functions related to the selected method?

C. Database

When talking about a tool's underlying database a few aspects should be pointed out. Firstly, the database is the heart of the tool, almost all actions, as creating, modifying and deleting objects and analyzing the relationships between dem, are in some way influenced by the database and its capacity. Secondly, diagramming tools are most often just an interface to the database, and gives us a simple and efficient way of seeing and manipulating the objects stored in the database. Therefore, the interaction between the database and description techniques is of great importance. Thirdly, the capacity and usefulness of available analyses and controls is due to the contents of the database and how t is managed.

C.1 Data dictionary

We use the term 'data dictionary', but 'repository' or 'encyclopedia' would do as well!

- Database type (relational, hierarchical, etc.)
- What is stored in the data dictionary and what are the associations between stored data? Try to draw a conceptual schema on the database contents, if possible.
- Is the dictionary uppdated on-line or by command when editing in diagrams?
- Can the data dictionary contents be imported from and exported to other tools?
- Can dictionaries be split and combined (e.g subproject <-> main project)?

C.2 Data access and consistency in data representations

- Is the dictionary contents immediately accessible for viewing and editing (e.g through a special dictionary editor)?
- Do all modifications made in diagrams imply uppdating of dictionary contents (and vice versa)?
- Is the data dictionary always consistent with diagrams?
- Are the diagrams always consistent with each other? (Note: This aspect is partially treated in section B.3)

C.3 Analysis functions, reports and error discovery

Analysis functions, reports and error discovery are often related to specific types of diagrams. It might therefore be a good idea to treat these functions from the diagrams' point of view.

Since reports are an important part of systems documentation and form the basis of manual analysis, the tool should generate well organized and relevant reports. Unnecessary or inappropiate information should be eliminated by the tool or by user options.

- What kinds of analyses/error-checks are performed on-line (e.g. syntax checks or warnings for inconsistencies when the user tries to perform inappropiate actions?)
- What other kind of analyses and checks does the tool support? (some desirable analyses: find inconsequences between diagrams, find objects that are not used or are not defined, find stand-alone objects, i.e. objects that have no associations to other objects)
- Examine the report functions. Is the user reduced to make use of existing reports? Can the user customize them or define his own reports?

C.3 Data security

- What kind of security functions does the tool support? Some examples:
  - auto-save (memory -> disk)
  - back-up function (automatic or by command)
  - dictionary 'recovery' or 'repair' functions
  - data access security: passwords, user privilegies,
  - data uppdate control in multiuser/network environments

D. User environment

D.1 User interface and user interaction

- Describe briefly the user interface (windows, mouse-support, pull-down/pop-up menues, etc.)
- Describe briefly the user interaction, that is how the user communicates with the tool and vice versa (Examples: User enters data through a command-line or a dialog box, draws and moves objects with mouse, the tool shows messages on a message-line or in a message box appearing on the screen).
- How many diagrams or windows can be opened and elaborated simultaneously?
- Does the tool support the user in navigating between diagrams or between hierachies of specifications?
- Does the tool warn the user when performing dangerous actions?
- Can the user "undo" actions

Illustrative figures might help!

D.2 Help

- Describe the available help functions:
  - on-line help
  - context sensitive help (i.e. right help in right situation)
- How detailed is the help information
- How 'intelligent' help is available (e.g. the tool suggests some possible actions to take in a certain situation)

D.3 Multiuser environment

If the tool supports networks or multiuser environments, describe how this influences the situation of a user.

E. Features and facilities

E.1 Portability

- Can specifications be transferred to other tools/environments

E.2 Communication

- Examine communication capabilites
  - built-in communication facilities
  - communication with DBMS, 4GL-systems, other systems
  - network support

E.3 Transformation of data structures

- Can specifications (e.g. Entity-Relationsship diagrams, data structure diagrams etc.) be transformed into database schemas of a DBMS.
- Can database schemas be transferred back to the CASE environment and redrawn?

E.4 Code generation

- examine the code generating facilities. Some points of interest:
  - code type (pseudo,4GL, COBOL,C,etc or compiled)
  - target environments
  - completeness of code: executable code, skeleton programs or declarations of datastructures only
- how powerful/easy-to-use is the coding facility?

E.5 Prototyping

- what kind of prototyping facilities are supported?
  - simulation of user dialogues
  - execution/interpretation of specifications or state-transition diagrams
- Does the tool support implementation of small 'test systems' which can be tested and completed gradually?

Note: powerful coding facilities eventually combined with reverse engineering might do as a kind of prototyping

E.6 Reusability of code and specifications

- examine reusability of object definitions and design specifications (including code). Some points of interest:
  - reuseability, both in the current and in future projects
  - libraries for saving specifications to be used in other contexts
  - support for searching among saved packages or libraries
- Reusing old specifications puts reguirements on analysis and error detecting functions. It might be an good idea to comment these functions from this specific point of view.

E.7 Flexibility

- Describe how the tool can be customized or extended.

E.8 Information control

When talking about information control, three types of control facilites can be considered and treated:

- Software configuration control: when the tool environment can be customized to user needs or different methods/techniques, the issue of compatibility between different tool customizations arises!
- Version/release control: compatibility between old and new versions of the tool.
- Information change tracking: detecting and managing alternative or different old versions of system specifications.

E.9 Project management support

- Does the tool provide project managing facilites?
- Can the tool be integrated with separate products for project management

F. Effects on the process of systems development

F.1 Productivity and quality improvements

Improvements in productivity and quality are often hard to estimate, since they depend - in addition to the tool- also on the lentght of the test period, the way of working, methods used, education in the method and tools. However, some well motivated approximations are desirable. We list some effects to consider when discussing these aspects:

- effects deriving from improvements in methods and organisation (often hard to estimate)
- better co-ordination of the project due to central dictionary
- more effective human communication due to illustrative and precise diagrams and automatic documentation
- improvements directly derivable from the tool capabilities (time savings, documentation, etc.)

F.2 Effects on systems life cycle

- What phases in systems life cycle does the tool act on and change?
- What steps in these phases are affected?
- How striking are the effects?
- What tool features are these effects due to?

F.3 Role of end users

- How can end-users be involved in the development process when using the tool?
- How does the end-user involvement affect the resulting system and its quality?

F.4 Education

Education is necessary to succesfully introduce a CASE tool and new methods in an enterprise. Later on, it is important to keep the knowledge up to date to ensure optimal use of the tool and methods. The following aspects should therefore be discussed.

- What are the desirable user qualifications?
- what courses on the tool and its methods does the vendor provide?
- What level of knowledge do these courses focus on (overview, more detailed, expert knowledge)?
- Is the tool/method of such complexity that it is advisable to have persons with 'expert knowledge' available in the enterprise?
- Considering the previous question, is there any risk for becoming dependent on external knowledge due to the complexity of the tool?

F. Hardware, software, documentation and costs

- Describe the hardware and software configurations in detail.
- What are the costs of hardware, software and education.
- Describe the documentation and its quality.

G. Conclusive remarks

G.1 Tool summary and assessment

- Summarize the tool and its disposition in the process of systems development
- What kind of enterprises/users can make the best use of the tool

- discuss advantages and disadvantages of the tool, but keep in mind that such things are often related to specific needs and requirements (i.e. to a specific enterprise).
- If the resulting report address a particular enterprise, discuss the value of the tool for that enterprise

G.2 Future improvements

- what improvements and changes will be carried out in future releases
- what improvements should be carried out? Motivate!

G.3 Subjective impressions

- Here you can tell about your private impressions.

3. Executing the Study

To carry trough the study without problems, it is advisable to examine the educational needs before strarting the study. It will certainly take a much longer time to learn a tool and its methods through using them in practice (trial-and-error-method) than a vendor course would take. Furthermore, good education is necessary when putting the tool on a real project, otherwise the project (and the study) will not be succesful.

Though not necessary, it as a good idea to get information about similar and commercially avalaible tools too. A survey of the state of the art in CASE area might help when assessing capabilities of a tool in proportion to other tools. However, vendor prospects are often fancy and have a greater bent for telling what the tool can do while omitting what it cannot do. To get more relevant and detailed information, it might be helpful to make use of the inspection protocol. The protocol serves as a checklist and a source of ideas when acquiring information about other tools.

Implementing the test case should be carried out as any systems development project. Of course, since the number of persons involved is probably small, no heavy project administration is necessary. The main exception is the protocol. When applying the tool and new methods, it is an good idea to keep the protocol in mind and write down experiences continuously. The protocol is helpful when formalizing experiences, and even more important, it gives ideas of what features and lacks to look for. Having the protocol as a guide and a notebook when examining the tool also reduces the risk of not taking some important aspects into consideration.

When writing the report, the organisation of the protocol should be followed, since it makes it easy to formalize and structure the obtained experiences. Furthermore, a standardized report to be used when comparing tools is one of the main advantages of the use of an inspection protocol. Knowledge about other tools and methods helps you to form a balanced and objective description of the the tool. It also makes it easier to analyze the problems that arose during the implementation work; whether they were due to shortcomings of the tool or insufficient knowledge.

4. Making the Decision

Although our focus is on the the practical study and the inspection protocol and not on how the final decision on investments in CASE technology should be made, we present some aspects worth considering.

- how well the tool satisfies the current and future requirements
- current hardware/software environment

- plans for future hardware/software environment
- future development of CASE tools (research results)
- changes needed in the organisation (new tools, new methods, new responsibilities)
- educational needs and investments

Additional aspects besides these and which strategy to choose are discussed more detailed in [Bubenko88].

5. Summary and Concluding Remarks

We have presented a way of testing CASE tools in practice. The idea is that detailed and relevant information is obtained about a CASE tool through putting it on a sythetized test case, alternatively a real systems development project, and analyzing its capabilities and features with assistance of an inspection protocol. The protocol serves as a checklist of tools attributes and what to examine including explanations and justifications of a variety of tools features. The protocol is supposed to be used both as an inspection guide during the practical work and as a skeleton for resulting reports. This ensures more equal studies and standardized reports that provide detailed and reliable information about tools and methods and form a basis for precise comparisons between tools.

This paper is mostly based on our work on studying commercially available CASE tools and our experiences from practical studies. We conclude with some practical advises worth considering when making practical studies on CASE tool.

- prepare carefully: analyze your needs and expectations, choose right tool type and test case (if you are going to use one)
- examine and complete your protocol with desired features and explanations
- have a good training in both tools and methods before you start, discuss with experienced users if possible
- make detailed notes during the practical work, use the protocol as a guide
- analyze your experiences and discuss the problems and the protocol with other users, compare your experiences
- compare with other similar tools, you might get some ideas
- check evething that is unclear with vendors or developers to avoid missunderstandings
- let the vendor or developers comment the report, you might have misunderstood or missed something, and moreover your critics might give them ideas of future improvements

REFERENCES

Bubenko88 Janis A. Bubenko jr, Selectin a Strategy for Computer Aided Software Engineering (CASE), SYSLAB, 1988

Brodie87 M.L. Brodie, Automating Database Design and Development, Tutorial, Nov 1987, Authors address: GTE Laboratories Inc, 40 Sylvan Road, Waltham, MA, 02254, USA

Floyd86 C. Floyd, A comparative Evaluation of Systems Development Methods, in [Olle86]

Martin88 C.F. Martin, Second Generation CASE Tools: a Challenge to Vendors, IEEE Software, 1988

Olle86 T.W. Olle, H.G. Sol, C.J. Tully (editors), Informations System Design Methodologies: Improving the Practice, Elseview Science Publishers B.V. (North Holland), Amsterdam, 1986

Pressman88 Roger S. Pressman, Making Software Engineering Happen, Prentice Hall, 1988

Rock89 Rosemary Rock-Evans, CASE Analyst Workbenches: a Detailed Evaluation, Ovum Ltd, 1989

Starts87 The Starts Guide: a guide to methods and software tools for construction of large real-time systems, NCC Publications, 1987