SUITABILITY OF »CASE« METHODS AND TOOLS FOR COMPUTER CONTROL SYSTEM

INFORMATICA 1/90

Keywords: case methods, computer control, tools

Janko Černetič
Institut »Jožef Stefan«, Ljubljana

Some questions are considered concerning the introduction of CASE (Computer-Aided Sofware Engineering) methods and tools into the development of computer-based control systems, with particular emphasis to process control and the domestic working environment. In the first part of the paper, the advantages of using CASE are discussed: the general ones, corresponding to any computer-based system, and the specific ones, corresponding to computer control systems. In the second part of the paper, a short review of real-time CASE methodology is given and the main benefits and problems occurring during the practical use of CASE are mentioned.

PRIMJERNOST 'CASE' METODA I ORUĐA ZA SISTEME RAČUNARSKE AUTOMATIZACIJE. U radu su razmatrana pitanja u vezi sa uvođenjem CASE (Computer - Aided Software Engineering) metoda i oruđa pri razvoju računarskih sistema vođenja, s posebnim osvrtom na sisteme procesne automatizacije i na domaće uvjete rada. U prvom dijelu rada spominju se prednosti CASE; najprije one općije, koje važe za bilo koji računarski projekt, a zatim i specifične prednosti za projekte računarske automatizacije. U drugom dijelu rada spomenute su glavne koristi i problemi pri uvođenju tih metoda i oruđa.

INTRODUCTION

In the last few years, the acronym CASE, denoting Computer- Aided Software Engineering, has become a significant keyword for the modern software engineering community. According to E.J. Chikofsky (1988), "CASE is primarily a production - oriented integration technology to meaningfully improve software and systems development". In fact, CASE integrates methods, computer-aided tools and an appropriate working environment. It is effectivity- and production-oriented, therefore it can be seen as a sort of engineering.

In this paper we wish to consider briefly the above-mentioned trends, particularly in the context of using CASE within the development of computer-based control systems. The interested reader should also refer to our associated paper (Rihar and Černetič, 1989) which will give more details on the practical use of CASE tools.

Before we begin, let us explain why CASE is interesting in the area of modern control systems. As a rule, such systems are all computer-based and are being classified into the broad category of real-time systems (Hindin and Rauch- Hindin, 1983). If it is said that "the tar pit of software engineering will be sticky for some time to come" (adapted from Weiss, 1985, citing Brooks, 1982), this is the more true in the case of real-time and control systems, respectively. In this situation, CASE is giving some hope to the worried project managers.

Because a national development project has been launched also in Yugoslavia (IJS, 1988), with the aim to improve the current engineering practices in the development of process control systems for the chemical and other processing industries, our opinion is that we can not afford blindly to ignore CASE.

2. SOME FEATURES OF 'CASE'

In brief, the main features of CASE are the following. First, they are historically based on the well known methods of structured systems

analysis and systems (and software) design (De Marco, 1978; Page-Jones, 1980). Second, they are characterized by: a systematic definition and analysis of the problem, lucid graphic representations of the concerned system, black-box simplification and multilevel hierarchical structuring of system functions and strict criteria to assess the quality of resulting design solutions.

Third, CASE methods are - at least principally - independent of computer type, programming language and sort of application. The corresponding strategies are recognizing the need for iterative system development, whereby they are separating the design phase from the analysis and specification of requirements. In addition, they introduce procedures for the verification of resulting documents with regard to completeness, correctness and consistency.

All the above-mentioned features of CASE methods and strategies can be properly extended to CASE (software) tools. They support the methods by means of efficient graphics, friendly human interface, their extensive data processing and storage capabilities, as well as options for easier documentation.

From the features mentioned above, one can quickly derive the basic benefits of CASE in the development of general computer-based systems or corresponding software. Right at the beginning of the project, the designers are supported in the development of concise functional specifications, which can be verified almost automatically. Then, at the design phase, they can derive a sound system design from the specifications, following formalized rules and guidelines. Any changes in basic system specifications or design improvements can be introduced in a controlled and ordely manner, whereby all corresponding diagrams and documents are much easily modified than with the "paper-and-pencil" method.

In summary, then, it can be concluded that the use of CASE results in a system and software which is of a good quality, has an updated and clear documentation, and is easy (i.e. cheap!) to maintain.

3. 'CASE' ALSO FOR CONTROL SYSTEMS?

The control systems, being a specific subset of real-time systems, must incorporate all the quality attributes associated with general computer-based systems, but, however, still some more. Similarly as with real-time systems, there are a few specific functional and performance requirements (Pressman, 1987) which are not easy to satisfy, namely:

- response time constraints,
- data transfer rate and throughput,
- interrupts and context switching,
- resource allocation and priorities,
- error handling and fault recovery,
- task sychronisation and
- inter-task communication.

As Pressman puts it for the case of real-time systems, "each of these performance attributes can be specified, but it is extremely difficult to verify if systems elements will achieve desired responses, if system resources will be sufficient to satisfy computational requirements, or if processing algorithms will execute with sufficent speed". All in all, "the design of real-time computing systems is the most challenging and complex task that can be undertaken by a software engineer. By its very nature, software for real-time systems makes demands on analysis, design, and testing techniques that are unknown in other application areas" (Pressman, 1987).

For the case of process control systems, we may admit that their performance requirements usually are not so severe, regarding only the system response time and data throughput rates (Hindin and Rauch- Hindin, 1983). Unlike most other real-time systems, they are complex in terms of the extent of communication with their environment (process operators, sensors and actuators) and because of some sophisticated (advanced) control algorithms.

Nowadays, the control algorithms for the processing industries are being designed and partly verified by simulation, using separate CACSD (Computer- Aided Control System Design) packages. In a sense, there is a specific approach in deriving the functional requirements of modern process control systems which include advanced control techniques, such as e.g. optimizing control. At the time being, there are no indications that CASE and CACSD can somehow be "married", but, in our opinion, this process must take place in the near future.

Nevertheless, side by side with CACSD, CASE seems to be the right "tool-box" also for process control systems, particularly because there are some methodological approaches addressing the above mentioned peculiarities of real-time systems. We will mention them shortly in the next paragraph.

4. A SHORT REVIEW OF 'CASE' METHODOLOGY

In a short article about the benefits of CASE for software engineering managers, Collard (1988) states that in the past few years, CASE has evolved from a concept to an industry. It is hard to believe and imagine such an explosive progress without knowing that the fundamental methodology behind CASE already has quite a history. With the inclusion of some important keywords and bibliographical references, a short past and future evolution of general-purpose CASE can be given in Fig. 1.

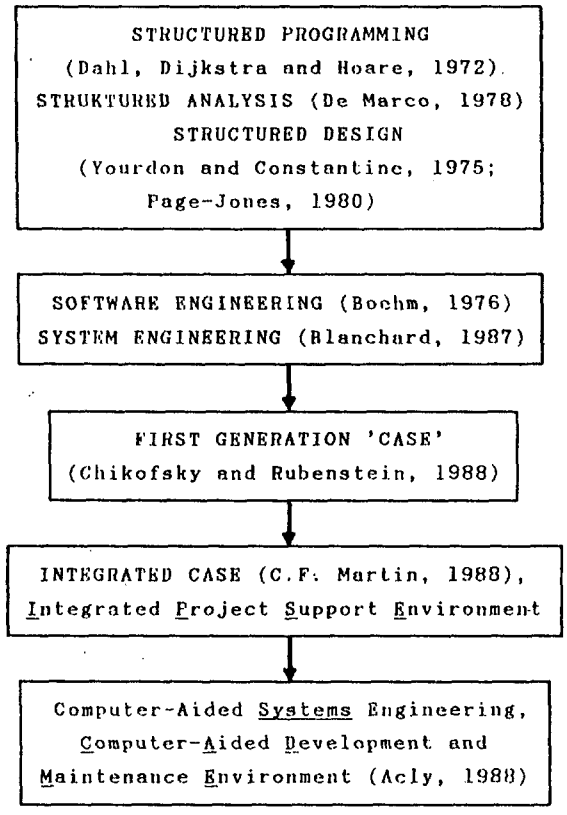

Fig. 1. The evolution of CASE methodology

As the interested reader can obtain some good surveys of general CASE methods in this journal and other readily accessible literature (e.g. Gyorkos and coworkers, 1988; or Pressman, 1987), we can limit ourselves to mention real-time CASE methods and tools.

In general, there are two approach directions to real-time CASE, the obvious and the original. The obvious and the more frequent one is extending the general CASE methodology by existing elements, to cover the specific real-time system requirements, mentioned in chapter 3 of this paper. In contrast, the "original" direction is developing an entire new framework to deal with real-time problems.

In the following, we will briefly present three typical real-time CASE approaches (or strategies), covering the spectrum from original to the extended. Their authors are: a) Hassan Gomaa, b) Ward and Mellor, and c) Hatley and Pirbhai, respectively. All three have based their functional system representations upon the well-known "data-flow" diagrams and structured systems analysis/design (De Marco, 1978).

a) The Design Approach for Real- Time Systems (DARTS) of Gomaa (1984, 1986) seems to be the most original one, as it specifically addresses the most important problems of real-time systems, i.e.:

- concurrency and task design,
- inter-task communication and synchronization, and
- state dependency and transaction processing.

In the DARTS, an entire life-cycle project phase is devoted to the design of tasks, i.e. structuring of software modules into concurrent processing units. Inter-task communication and synchronization is defined by means of special task interface modules, and a corresponding graphical notation is introduced (Fig. 2).

Because many real-time systems incorporate some degree of transaction processing, Gomaa has introduced an original solution to the problem of implementing a transaction which is dependent not only on the incoming data but also on the current state of the system (the so-called State Transition Manager module).

Another such useful representation, called the Event Sequence Diagram, shows the sequence of actions that are expected to take place when an external event occurs. There are still some interesting guidelines in DARTS for project organization, planning and management, such as the system architect and incremental development concepts, the former being taken from Brooks (1975).

b) In the Ward/Mellor approach, (Ward and Mellor, 1985; Ward, 1986), the authors propose the following, in addition to the previously known data-flow system modelling elements:

- extended notation to include control processes and flows,

- formation rules for a transformation schema to restrict ambiguous descriptions,
- execution rules for a transformation schema which are loosely based on the modified execution of a Petri net and visualized in terms of token placement,
- execution plans to manage the execution rules,
- extensions of the data-flow system model representing methods for dealing with certain problems of control transformations,
- separation of essential (i.e. functional) and implementation system models,
- hierarchies of transformation schemas to simplify the representation of complex systems.

Typical for this approach is that the control flows and processes appear together with data flows and processes on the same diagrams (Fig. 3). Each control process (transformation) must be associated with a state machine which, in turn, is being represented by a state-transition diagram or a corresponding state-transition matrix. In addition to ordinary data stores, used in the schema to indicate storage delays among transformations, buffers are introduced for exclusively storing information about discrete signals, implying destructive reading of its contents.

c) The Hatley/Pirbhai strategy (Hatley and Pirbhai, 1987) has some features in common with the Ward/Mellor approach, although - up to now - it has been elaborated more in detail primarily for system specification. The main features in common are:

- representation of control processes,
- representation of state transitions,
- part of formation rules,
- separation of requirements from implementation and
- hierarchical representation of complex systems.

In contrast to Ward and Mellor, Hatley and Pirbhai represent data flow (DFD) and control flow (CFD) separately. Moreover, they have devised detailed guidelines for how to separate the so-called "requirements model" from implementation, that is the "architecture model". The latter is being represented by a modified requirements model, augmented with implementation-technology dependent system features, such as: user interface processing, input/ output processing and maintenance, selftest and redundancy management processing (Fig. 4).

As depicted in Fig. 4, the requirements model consists of the process model and the control model. In addition, both are supported by a requirements dictionary. The process model is developed in a top-down fashion, beginning from the "context data-flow diagram", which is progressively broken down into a multi-leveled hierachy of more specific data-flow diagrams, with increasingly greater extent of details. The bottom level of the process model is simply described by short narrative, tabular or diagrammatic "process specifications" (PSPECS).

The control model looks similar to the process model, with the exception that it is completed by timing specifications and control specifications. The timing specification give the system timing contraints relative to its environment, whereas the control specifications define "trigger" signals to activate or deactivate particular processes in the process model. On the other hand, the primitive process specifications optionally define the so-called "data conditions", essentially control signals linking a data-flow diagram with a corresponding control-flow diagram. The structure of the requirements model is given in Fig. 5.

5. BENEFITS, PROBLEMS AND LIMITATIONS

The most complete qualitative representation of CASE benefits can be obtained from the features given in chapter 3. In addition to these, we could still mention better user involvement. As a representative of a major CASE tools developer and vendor (Arthur Andersen and Co.) says, "Automated design and prototyping tools play a powerful role in encouraging users to participate actively in ... (systems) design (R. O'Mahony, 1987).

A quantitative impression of CASE benefits can be derived from Fig. 6. Here data are depicted from a not so recent survey (1986) where the users by themselves have estimated the productivity improvement resulting from the use of Excelerator, the CASE tool of Index Technology Corporation (Chikofsky and Rubenstein, 1988). It is evident that, at that time, CASE had appeared to be the most valuable in the initial project phases. Other authors, e.g. Voelcker (1988), quote similar figures for productivity improvements.

But, unfortunately and in spite of proven benefits, it is reported that there are many problems associated with the introduction of CASE (see e.g. again Voelcker, 1988; or Shear, 1988). The main source of these problems is probably the novelty of CASE itself: "... there is nothing more difficult to take in hand, more perilous to conduct or more uncertain in its success, than to take the lead in the introduction of a new order of things ... " (a borrowed quotation from Pressman, 1987). No doubt that CASE is introducing "a new order of things" into the systems development process and, consequently, nobody likes to abandon his firmly established working habits, especially when this is associated with new learning effort and unclear future benefits. In the case of computer programming, the "moment of inertia" is still worse, as this profession is considered to be an intellectual art, impossible to fit in any ordered methodological framework.

Most often, this is the main problem behind the most common objections against CASE, just like the following few:

- if we do not begin with coding immediately, we will be late in delivery;
- the system user will not accept this way of doing;
- let the university people play with the "methods and approaches";
- we are working effectively without such guidelines;
- we will write system documentation later.

On the other side, there are some serious objections worth attention, because they are pointing to some general or specific limitation of (current-generation) CASE. Typical for this class are:

- productivity in system development is due mainly to good management.
- CASE is profitable only in "great" projects;
- how can I find a method suitable for my problem?

The first of these statements is absolutely true: CASE is no substitute for the skilled management of people, similarly as it cannot replace sound reasoning, although it supports both very well.

The second statement becomes true if the attribute "great" is defined in terms of system complexity. Indeed, CASE seems to be good for breaking down the requirements and the design of complex systems, such as typically are many modern computer-based process control systems.

The third of these objections, in the form of a question, can be resolved only by a good knowledge of available methods and tools.

In the context of our interest in CASE, we are perceiving two additional problems which, most probably, are still open. The first one has been already mentioned: there is a need to make a proper connection between the CACSD and CASE tools. The other is more specific: how to use or adapt CASE a) in our domestic social and working environment; and b) in typical research- intensive projects.

6. CONCLUSION

In our paper we tried to focus our reflections on the possible use of CASE in the development of process control systems. The statements given here represent a collection from the recently available foreign knowledge, mixed with some of our own opinions that were formed during a detailed study and numerous discussions of this topic.

This study has been started, and will be continued, in the framework of our efforts to find better ways of doing process control projects which will, hopefully, result in the advancement of our (control-science based) engineering profession and, second, in the conviction among our colleagues in industry, that it is worth-while to invest in domestic knowledge, instead of buying - and staying dependent of - foreign patents and licenses.

The preliminary results of our study indicate that CASE certainly is suitable for the development of computer-based process control systems, but, still some general and some specific problems concerning its introduction must be solved, before it can be used most effectively.

It would be to our sincere satisfaction if this knowledge can be of some value to any other professionals, dealing with the development in the demanding area of computer systems engineering.

Acknowledgment: The author is grateful to Dušan Osojnik, Rajko Kolar and Kordija Stiglic for valuable initial information; and further also to Marjan Rihar, Matjaž Šubelj and other colleagues for many stimulating discussions about CASE and software/systems engineering.

REFERENCES

Acly E., Looking beyond CASE, IEEE Software, March 1988, 39-43.

Blanchard B.S., Development of systems and equipment: Systems Engineering, Systems & Control Encyclopedia (Editor M.G. Singh), Pergamon Press, Oxford, 2, 995-1003.

Boehm B.W., Software Engineering, IEEE Trans. on Computers, C-25, 12(Dec. 1976), 1226-1241.

Brooks F.P., Jr., The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, Reading, MA, 1975, 1982.

Chikofsky E.J., Software technology people, IEEE Software, March 1988, 8-10.

Chikofsky E.J. and B.L. Rubenstein, CASE: Reliability engineering for information systems, IEEE Software, March 1988, 11-16.

Collard D., Selling CASE to your staff, Software Management, November 1988, 10.

Dahl O.-J., E.W. Dijkstra and C.A.R. Hoare, Structured Programming, Academic Press, London, 1972.

De Marco T., Structured Analysis and System Specification, Yourdon Press / Prentice-Hall, Englewood Cliffs, 1978.

Gomaa H., A software design method for real-time systems, Communications of the ACM, 27, 9 (Sept. 1984), 938-949.

Gomaa H., Software development of real-time systems, Communications of the ACM, 29, 7 (July 1986), 657-668.

Györkös J., I. Rozman and T. Welzer, A survey of most important and outstanding methods for software engineering, Informatica, 12, 2 (1988), 24-30.

Hatley D.J. and I.A. Pirbhai, Strategies for Real-Time System Specification, Dorset House Publishing, New York, 1987.

Hindin H.J., and W.B. Rauch-Hindin, Real-time systems, Electronic Design, January 6, 1983, 288-318.

Humphrey W.S., Characterizing the software process: a maturity framework, IEEE Software, March 1988, 73-79.

IJS (Institut "Jožef Stefan", Ljubljana), Prijedlog za poticanje projekta "Računalska automatizacija procesa u kemijskoj i procesnoj industriji, Ljubljana, april 1988; accepted for funding by the Yugoslav federal committee for science and technology, under the code PR-24.

Martin C.F., Second-generation CASE tools: a challenge to vendors, IEEE Software, March 1988, 46-49.

O'Mahony R., Successful systems: new ways to involve users, The Consultant Forum, 4, 2 (1987, Digital Equipment Corporation), 11-13.

Page-Jones M., The Practical Guide to Structured Systems Design, Yourdon Press, New York, 1980.

Pressman R.S., Software Engineering - A Practitioner's Approach, (2-nd Ed.) McGraw-Hill, New York, 1982, 1987.

Rihar M. and J. Černetič, Some practical aspects of using CASE tools, Informatica, 1989.

Shear D., CASE shows promise but confusion still exists, Electronic Design News, December 1988, 164-172.

Voelcker J., Automating SW: proceed with caution, IEEE Spectrum, July 1988, 25- 27.

Ward P.T. and S.J. Mellor, Structured Development for Real-Time Systems, Yourdon Press, New York, 1985.

Ward P.T., The transformation schema: An extension of the data flow diagram to represent control and timing, IEEE Trans. on Software Engineering, SE-12, 2, (Feb. 1986), 198-210.

Weiss E.A., The permanent software crisis - recommended classics for those in the ever- sticky software engineering tar pit, ABACUS, 3, 1, Fall 1985.

Yourdon E. and L.L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, Prentice-Hall, Englewood Cliffs, 1975.

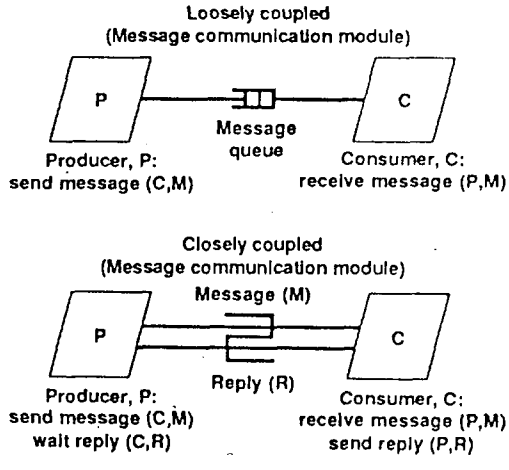

Fig. 2. a) DARTS notation for inter-task communication

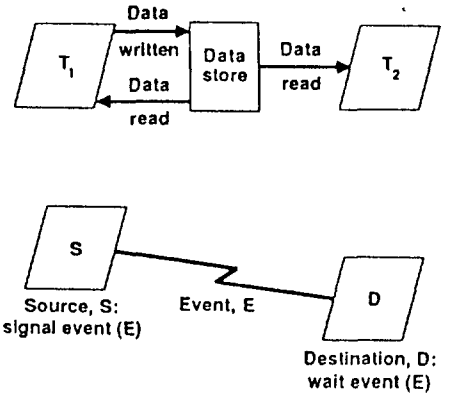

Fig. 2. b) DARTS notation for information hiding and task synchronization.

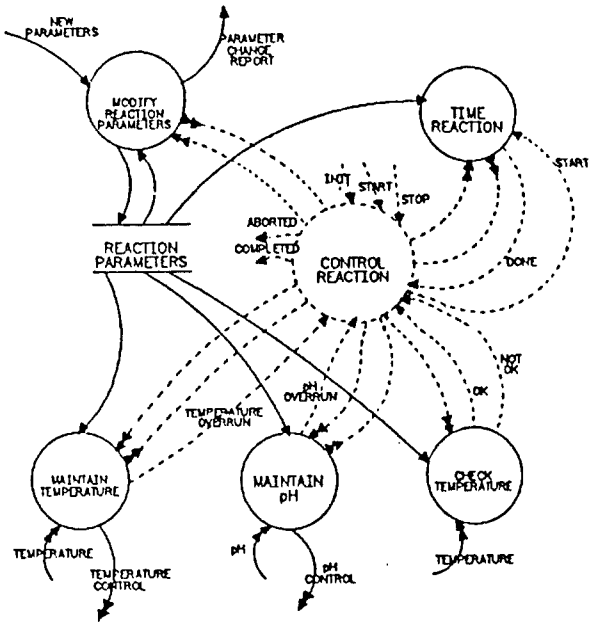

Fig. 3. Transformation schema, drawn in the Ward/Mellor notation, for a simple process control system.

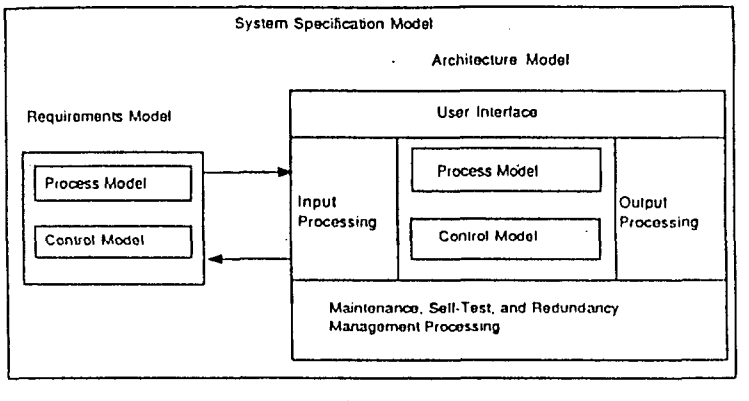

Fig. 4. Overall structure of the system specification model by Hatley and Pirbhai.

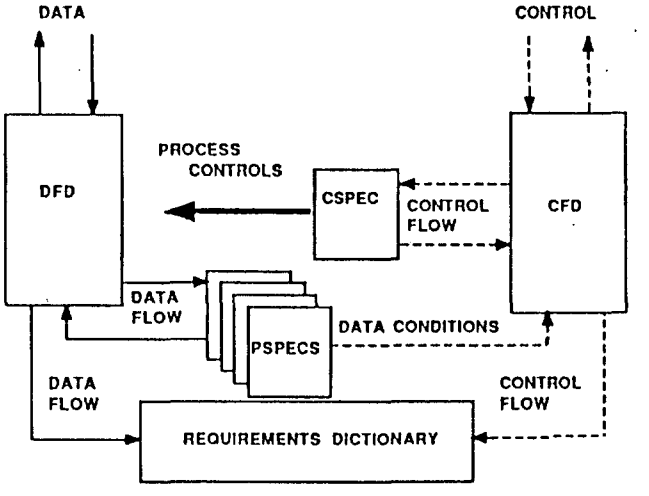

Fig. 5. The structure of the requirements model by Hatley and Pirbhai.

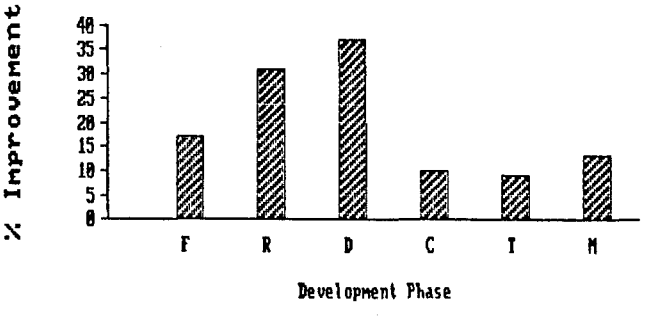

Fig. 6. Productivity improvement in particular system development phases (F - M) when using the CASE tool Excelerator (Data froma a user survey).

Legend: F = Feasibility study; R = Requirements specification; D = Analysis and design; C = Coding; T = Testing; M = Maintenance.