

THE CASE FOR USER-CENTERED CASE TOOLS

CASE tools need to become more useful if they are to be applied to the practice of software production.

The interest in CASE tools is based on expectations of increased productivity, improved product quality, easier maintenance, and making software development a more enjoyable task. Current experiences, however, suggest that CASE technology hardly delivers those promised benefits. After more than a decade, CASE tools have not been widely used [1, 11]. At the same time, the CASE market is rapidly growing. A survey of the CASE tool market showed the annual worldwide market for CASE tools was \$4.8 billion in 1990 and grew to approximately \$12.11 billion in 1995. Why are CASE tools dearly bought but sparsely used? This phenomenon is worthy of careful attention.

We analyzed our own experiences with designing and using CASE tools in industrial software projects. We also investigated common tool adoption practices. We examined CASE technology from the perspective of technical and non-technical issues involved in software development. Our main conclusion is that current CASE tools are far too oriented on software modeling and construction methods, while other factors that matter to programmers receive little attention. We imagine the creative, problem-solving aspects of software development and perception of a software project from a process, rather than method, perspective. We observed that method-centered CASE tools are not attractive enough to the users. To better meld into the software development practice, CASE tools should adopt a programmer's mental model of software projects. In particular, CASE tools should support *soft*

aspects of software development as well as rigorous modeling, provide a natural process-oriented development framework rather than a method-oriented one, and play a more active role in software development than current CASE tools. Separation of development methods from other aspects that contribute to successful software projects does not benefit CASE users, whether programmers or project managers. In this article, we analyze problems that impede wide adoption of CASE tools and propose remedies to some of the problems. We direct this article to managers involved in CASE tool adoption, CASE tool users, and CASE tool developers. Although we focus on CASE tools that support software development according to some methods (such as structured analysis/design or an object-oriented method), we believe our observations apply to other types of software tools, too.

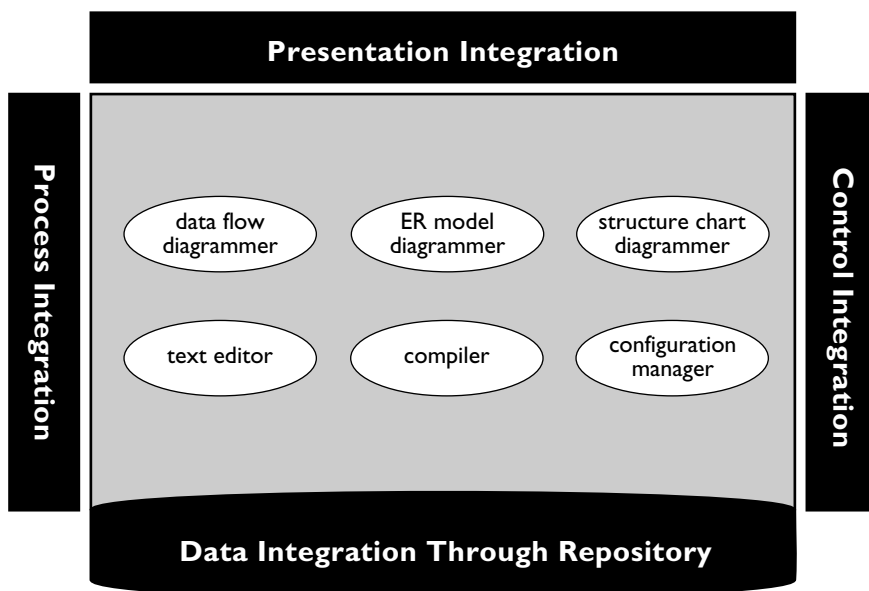


Figure 1. Dimensions of CASE tool integration

A CASE tool is a computer-based product aimed at supporting one or more techniques within a software development method (such as a structured or object-oriented method). There are CASE products that address multiple methods and Meta-CASE products that generate CASE tools from method specifications. Upper-CASE tools focus on the system analysis and logical design phases, while Lower-CASE tools focus on the construction of software systems. CASE tools can be integrated together to form a more sophisticated CASE environment. This can be done in several dimensions, such as presentation integration, control

integration, process integration, and data integration [5], as shown in Figure 1.

A typical CASE tool offers graphical editors to help developers model different aspects of a software system such as functional requirements, modular structure, data structure, or system behavior, according to the conventions of a given software method. In addition, CASE tools have a built-in knowledge of method rules such as how to draw a correct data diagram or how to generate a relational schema from entity-relationship diagrams. This knowledge allows CASE tools to check whether developers follow method rules and to automate some of the transitions between software development phases, including the generation of code. Most CASE products can be installed on a LAN to support team development, a CASE repository resides on the server and a proper locking mechanism controls multiuser access to stored project data.

Ivan Aaen discussed shortcomings of existing CASE tools and showed how they affect the adoption of CASE tools in [1]. The moral from his study is that CASE tools should offer some immediate benefits for the software developers, not just long-term advantages to the company. We should hope that future CASE tools will be more attractive to the developers. But what is it exactly that would make CASE tools more attractive to the developers?

We believe CASE tools must address all the major elements that constitute a successful software project. These are software modeling and construction methods, software process, and representation of knowledge about real-world processes supported by software systems. Each of the elements has hard aspects (technique, documentation, and rigor) and soft aspects (creativity, problem solving) that a CASE tool should support. Current CASE tools are particularly weak in addressing soft aspects of software development and we shall concentrate our discussion mainly on this issue. If a tool does not blend well into programming

practice (such as impede thinking and creativity), developers will be reluctant to use it even if the tool might provide excellent support for methods. When we initially evaluate a CASE tool for possible adoption in a company, we tend to examine how well a CASE tool supports technical details of a method we use. However, when we put a CASE tool into use, the evaluation perspective changes, becoming holistic. CASE users will experience the presence of a tool in their work and feel how well a tool helps them achieve software project goals. Tool vendors often claim that lack of training and proper organizational structure are the reasons why CASE tools become shelfware. Based on our CASE experience, we think the real reasons are method- rather than user-centered design of tools, weak support for soft aspects of software development, and too a narrow view of what makes a successful software project.

Current CASE tools are still in the method-centered stage. It is common to classify CASE tools based on technical criteria only [10]. Here, we discuss why

a method-centered approach impedes the adoption of CASE tools in industry and suggest remedies to some of the problems.

Support for Creativity in Software Development

The left part of human brain deals with numbers, logic, words, formal reasoning, and other elements of hard processing. The right brain specializes in soft and less tangible aspects of cognitive processes such as intuition, creativity, spatial imagination, color, rhythm, and emotions [2]. Most tasks require us to use both parts of our brain. No doubt software development involves a rich blend of hard and soft functions of our brain. Software methods define development phases, describe how to model software systems in a rigorous way and prescribe quality control procedures—all hard aspects of a software project. A soft aspect of software development is problem solving and the many elements related to it such as creativity, understanding, idea generation, free play of ideas, intuition, unconstrained thinking process, analogy, goals, rationale, interest, emotions and freshness or tiredness of human mind. Today's CASE tools are built to support methods. They do little to support soft aspects of software development.

Sometimes developers must concentrate on the business context, design rationale, goals and intentions of design, create alternative design solutions, experiment with them and evaluate the implications of decisions on the overall project goals. Developers hardly find the CASE tool a partner in such goal-oriented problem solving activities. They find CASE diagrams

too dry, notations and editors too restrictive, impeding rather than helping in their work. These problems are

Neural Network

A neural network contains a large number of simple processing elements called neurons with weighted connections among neurons. Weights of the connections encode the knowledge. By adjusting the connection weights, a network learns new facts. Learning rules determine how to adjust connection weights to optimize the network performance. The intelligence of a neural network emerges from the collective behavior of neurons, each of which performs only very simple operations.

aggravated in a team development situation whereby the whole design concept and rationale do not remain in one person's head. A well-designed user interface should create a metaphor that bridges the conceptual gap between a computer system and human thinking [9]. Such a metaphor is not the strength of current CASE tools.

During problem solving, one needs freedom in expressing ideas and a tolerant environment that would stimulate intuition and support concepts such as goal, decision, reason, consequence, impact, and variant solution. An environment should allow a developer to use many forms of expressing thoughts and move easily between different forms of expression.

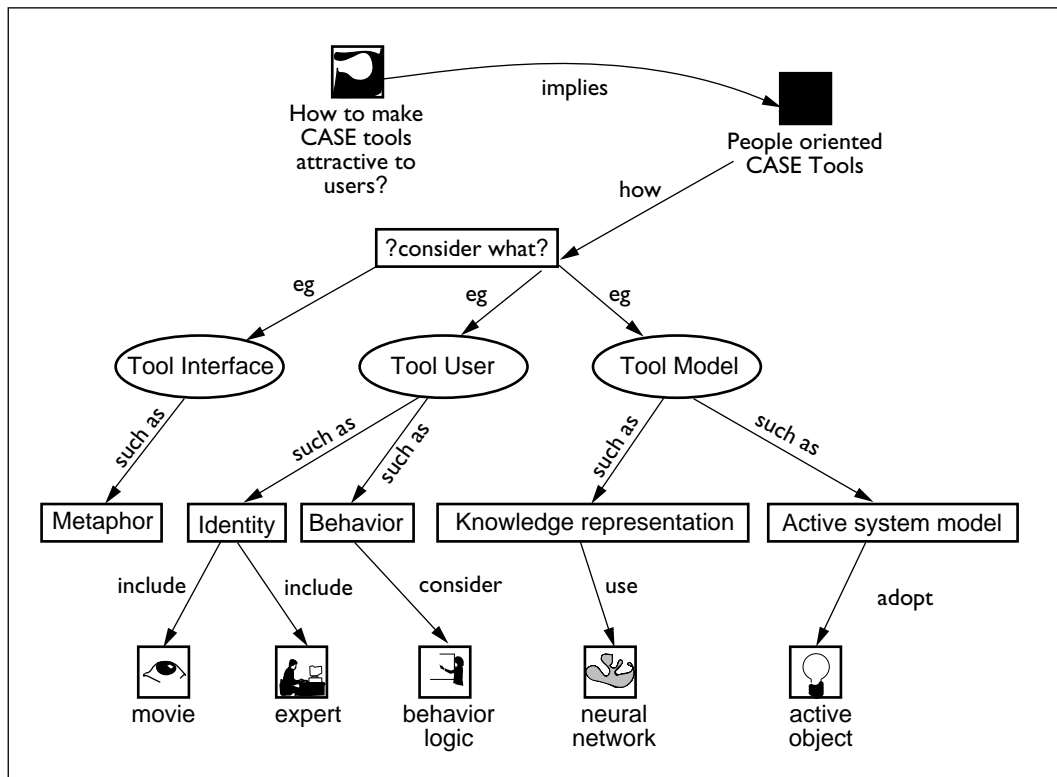


Figure 2. An example of idea presentation using Idea Processor

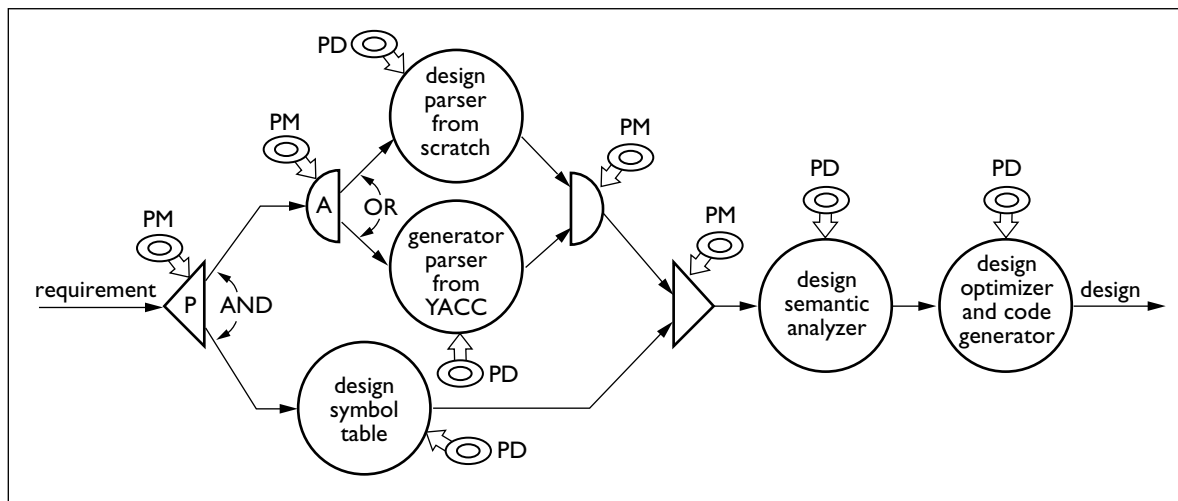


Figure 3. An example of a software process

A software process can be modeled in terms of concepts such as *activity*, *goal*, *problem*, *event*, *decision*, *rule*, *role*, *deliverable*, and *information*. A process is described by chains of activities. Humans and tools perform activities to complete the process. For each goal (for example, “to reduce system response to the maximum of 5 seconds” or “to fix a bug”), there is a scenario (or a number of alternative scenarios) that leads to achieving the goal. A scenario is defined in terms of activities to be performed and decisions to be made. Activities may be decomposed into lower-level scenarios. Activity execution is governed by rules. Rules describe what happens at decision points or indicate chains of activities that can be performed in parallel. Timing constraints are also modeled as rules. Roles model people responsible for activities. Deliverables are inputs to and outputs from processes, scenarios, and activities.

Figure 3 is an example of a software process for a compiler construction project. Small double circles indicate roles. PM and PD represent project manager and project designer, respectively. Triangle with P starts parallel activities. The half-circle with A starts alternative activities. Triangles and half-circles are decision points in the software process. For simplicity, we omitted many details, such as deliverables, from the figure.

An environment should recognize multiple appearances of the same concept in different forms and record relationships between concepts. Throughout the whole software project, developers should be able to refer to this intentional and motivational level of project descriptions.

An example of a tool that can effectively support thinking is Idea Processor.TM The tool is mode-less in the sense it allows a user to do anything at any time. It appeals to intuition through shape, size, color, and layout of represented ideas. Based on simple principles, the tool offers a rich set of functions that help one organize ideas, relationships between ideas, record rationale for why certain things are done and to trace the impact of changes. A developer can define the form of expression that fits his/her current need on the spot. Figure 2 is an example of idea presentation using the Idea Processor.

Suppose we model data pertinent to a given application. CASE tools support entity-relationship model-

ing and this is the highest conceptual notation to model data. While the ER notation is ideal for precisely formalizing data definitions and conceptual schema for an application database, it may not be ideal for capturing data requirements in the initial stage. This is particularly true if application end users rather than data modeling experts are to capture initial data. It is necessary to have a less constrained notation to sketch data entities to annotate them with additional information.

An unconstrained modeling environment should be an integral part of future CASE tools to assist developers in problem solving in all phases of software development. Ideas arising during problem solving would provide a context for understanding more technical and rigorous software descriptions. We believe future CASE tools will allow software developers to think and work at the level of application endusers whenever possible. This will be beneficial for software experts and in line with the recent trend toward end-user computing.

Active Tools

A CASE tool should provide more guidance for a novice developer than for an expert. However, most CASE tools appear almost the same to both novice and expert developers. Consider user interface, for example. Usually, for each function of a tool there will be a button or a menu item to access that function. Is this a good way to present the system functions to the developer? The answer might be *yes* for an expert developer, but *no* for a novice one who has only little knowledge of the tool functionality. A tool might allow a novice developer to request services and then expose functions relevant to a task in hand in an incremental way. Psychological tests reveal that human ability to cope with the complexity is highly restricted (7+2 principle). So, even in case of an expert developer, there is little advantage to expose many system functions at any one time.

This type of user interface requires a CASE tool to be active. Otherwise, hiding functions may appear to be a setback rather than progress. An active CASE tool is different from conventional one in that it can guide the users in the development process, figure out developers' intentions and customize tool functions to a task at hand. Most of the current CASE tools are passive in that sense.

Active CASE tools must be based on sound knowledge from different types of developers and their expectations. CASE tools are active at the method level (they can check consistency of software models and verify intermodel rules), but they are rather passive at the software process and developer levels. To provide active assistance for developers, a CASE tool should have a built-in knowledge of software methods, software process, application domain, and developer behavior. A tool must integrate and utilize these multiple layers of knowledge so that it can adapt to a development task in hand and to a developer's expectations. An active CASE tool should be self-learning and self-improving, and should be able to work and reason with incomplete information. Neural networks can be used for this purpose.

It is natural for a developer or project manager to

view a software project as a process. Software process modeling is a good way to integrate people, methods, tools, project management, and development activities. Figure 3 is an example of a software process for a compiler construction. Project planning, execution, and monitoring also requires a process perspective. There are some available process models and tools [3, 8]. However, current CASE tools provide very limited process-oriented facilities. Separating method issues from the work process will not benefit software developers or project managers.

We believe an explicit software process model should be built into a tool architecture and should drive the operation of an active CASE tool. A process-driven CASE tool would meld into the programming practice much easier than a method-driven tool.

Process modeling technology should be advanced to provide a good foundation for CASE tools. Most of the current process models are too strict and too weak in coping with the stochastic change of the process and the change of the constraints [6]. In reality, we often need to modify detailed process characteristics on the fly, during process enactment.

Process metamodels make such modifications possible. In OASIS—a process-centered environment based on a metamodel [8]—process models are objects organized into inheritance network. Processes inherit characteristics from metamodels. Customization and evolution of processes is achieved by defining variants and revisions of metamodel and process model objects. The metamodel method has the potential to alleviate the problem of dynamic process evolution, but this method is more suitable for CASE designers rather than for CASE users.

Most of the process modeling systems use a centralized control mechanism to check and enforce consistency of process specifications. This, however, does not reflect the needs of concurrent and distributed software engineering processes. Tolerating inconsistencies is often desirable in order to avoid unnecessary restrictions on the development process. Inconsistency

AN UNCONSTRAINED MODELING ENVIRONMENT SHOULD BE AN INTEGRAL PART OF FUTURE CASE TOOLS TO ASSIST DEVELOPERS IN PROBLEM SOLVING IN ALL PHASES OF SOFTWARE DEVELOPMENT.

management is a complex task. Local agents have to decide what checks to invoke, when to invoke them, and how to keep track of the results. Decentralized process models directly address these issues. CASE tools that support collaborative team software development would benefit greatly from decentralized

Modeling Real-World Processes

Software developers, particularly system analysts, should have easy access to models of real-world processes supported by software systems they build. Consider building software systems for business. The bottom line for evaluating the success of a soft-

ware system is how much value it adds to the business. To observe opportunities for value-adding software systems, one must have a good understanding of global, inter-departmental business processes. However, typical CASE notations for business analysis depict merely the static structure of business in terms of departmental struc-

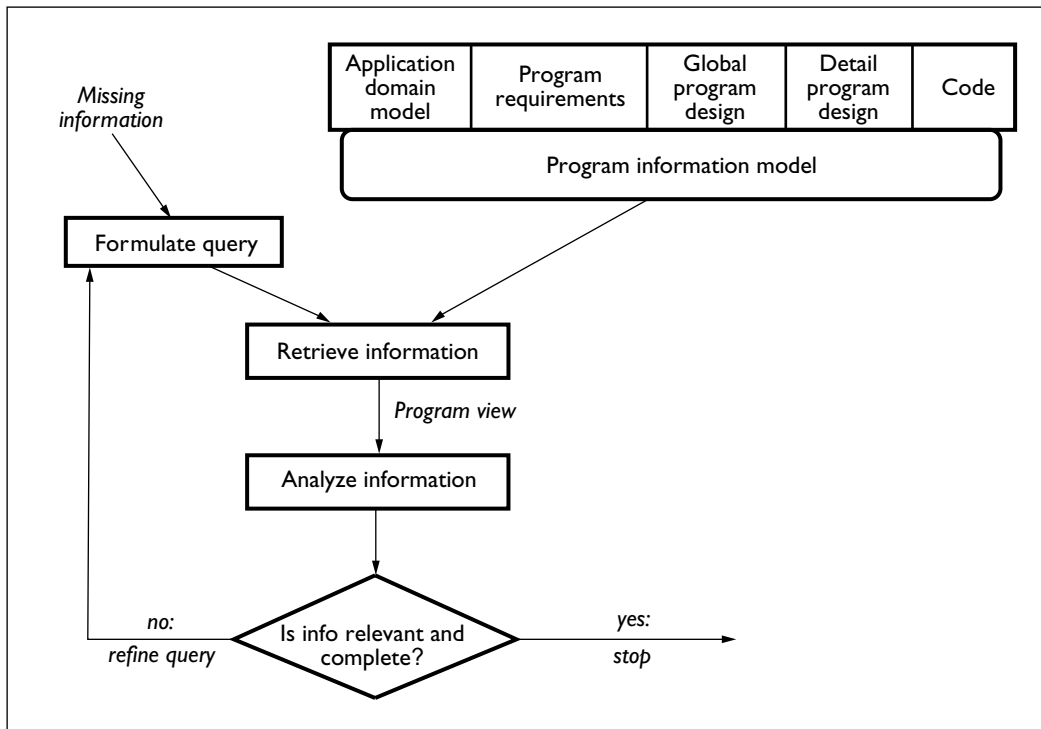


Figure 4. A model of a software developer's behavior

process models. A process model, such as ViewPoint [9], is an example of a step in this direction.

To avoid reinventing the wheel, active CASE tools should contain a knowledge base of standard solutions to well known software development problems. By suggesting solutions to routine tasks, active CASE tools would allow developers to concentrate on essential problems that require truly original solutions. Reinventing solutions to problems that others have already solved is not a sign of originality, unless our objective is to improve on the existing solutions. We can identify common design patterns in application domains. There are standard ways of dealing with accountancy, payroll, or customer-order processing problems. An active CASE tool should offer a wide range of predefined, application domain-specific templates for reuse. Screen painters, report generators, and database schema generators, are good examples of generic facilities that automate common tasks in designing database applications. CASE tools should provide similar facilities at the level of different application domains.

ture, functions performed by departments, databases used, and business systems required to automate operations of departments. This is not enough to build systems that radically improve business operations. A system analyst should be able to model workflow, interdepartmental communication channels, information sharing, determine why certain things are done, assess the importance of various activities in a given business process and their impact on achieving business goals. Such business models are created in business reengineering projects [12], but rarely precede routine analysis of software requirements. If CASE tools support the modeling of business dynamics, software system analysts would have easy access to the information so vital to the success of a system and there would be less disappointment with systems not adding enough value to the business.

User-centered CASE tools focus on the behavior of developers who work on software projects. Therefore, we must analyze and understand what software developers do to the extent we can build models of developer behavior. Then, we can design tools around such models. We illustrate behavioral models on the exam-

ple of analysis of programs for understanding.

Understanding software is essential to any development or maintenance activity. Studies show that maintenance programmers spend almost half of their time analyzing code for general comprehension. In big development projects, understanding software models and design decisions creates similar problems. An important aspect of understanding is looking for missing information.

Brooks observed that developers build models of a program that span from application domain down to the syntax of a programming language [4]. To understand a program, developers often look for information related to application domain and program requirements. They search through the code to see how various application domain concepts and user requirements are reflected to understand relationships between program objects such as procedures and variables. Often, developers deal with the problem of missing information. Looking for possible small subsets of information relevant to a task in hand is the heart and soul of program understanding. Therefore, to model developer behavior, we introduce a concept of query. Queries address situations when some piece of information a developer needs is not readily available.

A program view retrieval cycle depicted in Figure 4 starts when a developer needs to perform a certain task, make a decision or meet a goal, and finds that some information is missing. A query formally describes a needed program view. Having retrieved a view (from the sources represented by the *program information model* in the Figure 4), a developer judges the relevance of the view to the problem in hand and, eventually, refines the query. The model, though very simple, is useful in analysis of capabilities of tools that support program understanding. In particular, we can discuss the following issues:

- What types of program views do developers need?
- What type of program information is needed to obtain these views?
- Retrieval of which program views is particularly labor intensive?
- Which program views can and which cannot be retrieved automatically?
- What would be required to make automation possible?
- If a given program view cannot be retrieved automatically, is there any other way that a tool can help?

Models like this one should be studied to provide sound foundation for designing user-centered CASE tools.

Conclusion

We've analyzed problems impeding adoption of CASE tools and suggest remedies to alleviate some of these problems. The main point we make is that current method-centered CASE tools are not attractive enough to the users. To better blend into the software development practice, CASE tools must be more user-oriented, and support creative problem-solving aspects of software development as well as rigorous modeling. The tools should also provide a natural process-oriented rather than method-oriented development framework and play a more active role in software development than current CASE tools. Future CASE tools should be based on sound models of a software process and user behavior. **G**

REFERENCES

1. Aaen, I. CASE Tool bootstrapping (how little strokes fell great oaks). *Next Generation CASE Tools*. K. Lyytinen, and V-P. Tahvanainen, Eds. IOS, Netherlands (1992), 8–17.
2. Axon Research. Axon Idea Processor, User manual, 1996.
3. Bandinelli, S. et al. SPADE: An environment for software process analysis, design, and enactment. *Software Process Modelling and Technology*. A. Finkelstein, J. Kramer, and B. Nuseibeh, Eds. Research Studies Press Ltd., England, 1994, 223–247.
4. Brooks, R. Towards a theory of the comprehension of computer programs. *Intern. J. of Man-Machine Studies* 18 (1983), 543–554.
5. Chen, M. and Norman, R. A Framework for Integrated CASE. *IEEE Softw.* 3 (1992), 18–22.
6. Cugola, G. et al. How To deal with deviations during process model enactment. In *Proceedings of the 17th International Conference on Software Engineering*. (1995), 265–273.
7. Engels, G. et al. SOCCA: Specifications of coordinated and cooperative activities. *Software Process Modelling and Technology*. A. Finkelstein, J. Kramer, and B. Nuseibeh, Eds. Research Studies Press Ltd., England, 1994, 71–102.
8. Jamart, P. et al. A reflective approach to process model customization, enactment and evolution. In *Proceedings of the 3rd International Conference on the Software Process*. 1994, 21–32.
9. Leonhardt, U. et al. Decentralized process enactment in a multi-perspective development environment. In *Proceedings of the 17th International Conference on Software Engineering*. 1995, 255–264.
10. Lyytinen, K. and Tahvanainen, V.-P. Introduction: Towards the next generation of Computer Aided Software Engineering (CASE). *Next Generation CASE Tools*. K. Lyytinen, and V-P. Tahvanainen, Eds. IOS, Netherlands, 1992, 1–7.
11. Russell, F. The case for CASE. *Software Engineering: A European Perspective*. R. Thayer, and A. McGettrick, Eds. IEEE Computer Society Press, Los Alamitos, Calif., 1993, 531–547.
12. Spurr, K., Layzell, P., Jennison, L. and Richards, N., Eds. *Software Assistance for Business Reengineering*. 1993, John Wiley, NY.

Idea Processor is a trademark of Axon Research.

STAN JARZABEK (stan@iscs.nus.sg) is a senior lecturer in the Department of Information Systems and Computer Science at The National University of Singapore.

RIRI HUANG (huangr@iscs.nus.sg) is a postdoctoral fellow at the Department of Information Systems and Computer Science at The National University of Singapore.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.