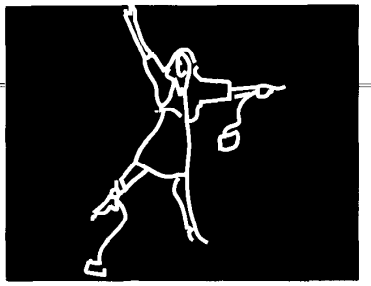

A Hypertext Based Sofware-Engineering Environment

JACOB L. CYBULSKI and KARL REED
Amdahl Australian Intellegent Tools Program

◆ *Tools from information management and software development combine to provide document developers with maximum flexibility plus a guarantee of correctness and consistency.*

The recent explosion in computer use has put pressure on software developers to use software-engineering methods that adhere to approved life-cycle models, specific types of deliverable documents, and detailed project organization and control techniques.[1] Many believe that these methods can effectively deal with the explosion, but if development costs are to decrease, they must also be automated.

As early as the 1970s, software researchers attempted to improve development efficiency by automating and integrating aspects of software development.[2] Initial efforts in CASE focused on source-code formatting, structured-programming support, and test-case generation. The emphasis has since shifted to creating development environments that support flexible or customizable process models. These environments comprise tools for interactively constructing system descriptions that rely on standard diagraming techniques and allow code generation. Integrating such tools into a uniform CASE environment is not possible, however, unless certain conditions are met:

◆ You can use different document classes at various development phases.

◆ Documents produced by one tool can be used by other tools farther along development.

◆ You can efficiently and conveniently traverse (navigate) among diverse documents that describe the system and its components. Figure 1 shows how software engineers might navigate through these documents.

◆ All documents can be guaranteed to be coherent and consistent throughout the software's life cycle.

For example, when viewing a process symbol in a dataflow diagram, you may

0740-7459/92/0300/0062/$03.00 © IEEE

wish to display the code associated with the process, display its representation in a state-transition diagram, and inspect the relevant section of a (standard) system requirements specification or a corresponding paragraph in the feasibility description. To do this, you would have to be able to access a variety of documents simultaneously.

These conditions suggest that a CASE system should have mechanisms for constructing software documents, for creating navigable links among them, and for navigating through the repository's components. A straightforward way to give a CASE system these features is to integrate it with a system that already has them.

CASE AND HYPERTEXT

We decided to integrate our CASE tools under an extended hypertext system, an information-management concept that evolved quite independently of CASE. Table 1 shows how we relate the features of hypertext to CASE components. In hypertext, information fragments in various forms are grouped and linked in a way that lets you search and browse through them nonsequentially.[3] By extending this environment to accommodate an integrated CASE tool set, we could guarantee correctness and consistency and provide maximum flexibility. This approach has been adopted by only a few significant CASE projects like Tektronix's Neptune project[4] and the University of Southern California's System Factory Project. (For more details on this project, see Peiwei Mi and Walt Scacchi's article on pp. 45-53.)[5,6]

The resulting environment, called HyperCASE, is an architectural framework for integrating the collection of tools. It is being produced as part of the Amdahl Australian Intelligent Tools Program, which is a joint project between Amdahl Australia, La Trobe University, and Prometheus Software Developments. The system provides a visual, integrated and customizable software-engineering environment consisting of loosely coupled tools for presentations involving both text and diagrams. HyperCASE's objective —

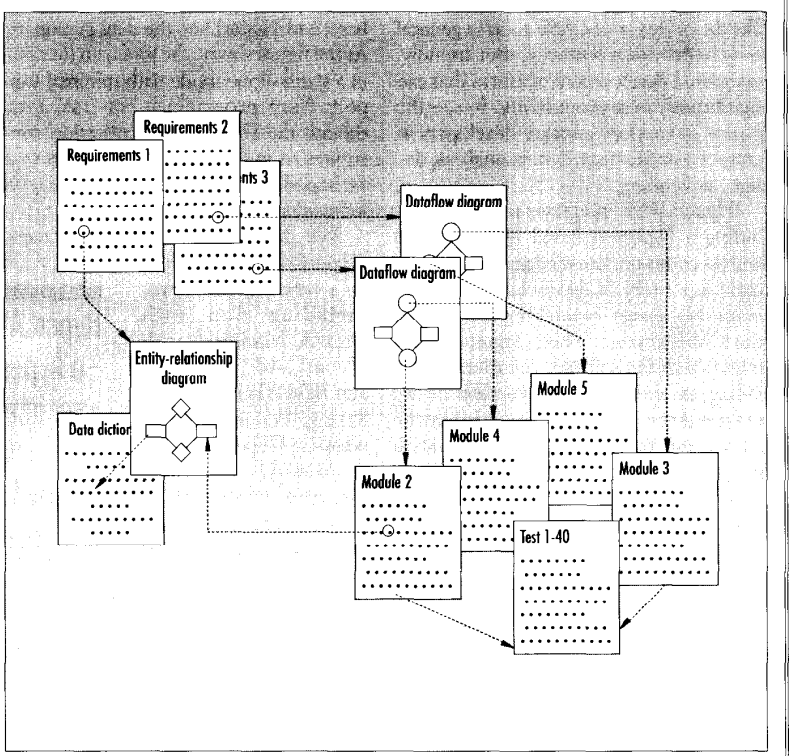

Figure 1. Navigating among software documents.

TABLE 1
HYPERTEXT APPLICATIONS TO CASE

Hypertext	CASE
Document authoring	Diagram editors Text-oriented tools
Browsing and navigation	**Traversing through program slices and refinement levels and across semantic terms**
Document aggregation	Module libraries Data-structure groups
Virtual structures	**Code generation**
Dynamic computation	Runtime results
Revision management	**Software-configuration management**
Group work	Project-team development
Extensibility/tailorability	**Multiple methodologies**
Concept annotation	Design-decision recording
Consistency checking	**Validation** **Verification**
Completeness assessment	Project-plan tracking

like the objective of CASE tools in general — is to provide a powerful, user-friendly, integrated development platform that can significantly raise productivity. Its specific goal is to support software developers in project management, system analysis, design, and coding.

HyperCASE integrates tools by combining a hypertext-based user interface with a common knowledge-based document repository. It also includes extensive natural-language capabilities tailored to the CASE domain. These are used in the interface to the software repository, providing an alternative to hypertext information management and interdocument navigation. You can also analyze English input during informal system-requirements specification, allowing a significant degree of automation for design and concept reuse at the earliest development stages.

Figure 2 shows HyperCASE's three subsystems: HyperEdit, the graphical user interface; HyperBase, the knowledge base; and HyperDict, the data dictionary. As the figure shows, the tools can function in a stand-alone mode with minimal support from proprietary systems, even though the HyperCASE collection constitutes an integrated system. In this way, we hoped to maximize the opportunity for independent tool development.

We developed a number of early HyperCASE prototypes on a variety of platforms, including Sun with Unix/X, Macintosh with HyperCard, and Macs and IBM ATs with C and Prolog. We are now developing HyperCASE on an IBM AT with Open Desktop. HyperEdit is being implemented under X Windows using the Open Software Foundation's Motif tool kit, HyperBase is implemented using Prolog, and HyperDict is designed as a Prolog data store in Ingres. We expect the final system to be implemented on Amdahl's UTS system (Amdahl's version of Unix based on System V).

HyperCASE combines a hypertext-based user interface with a common knowledge-based document repository.

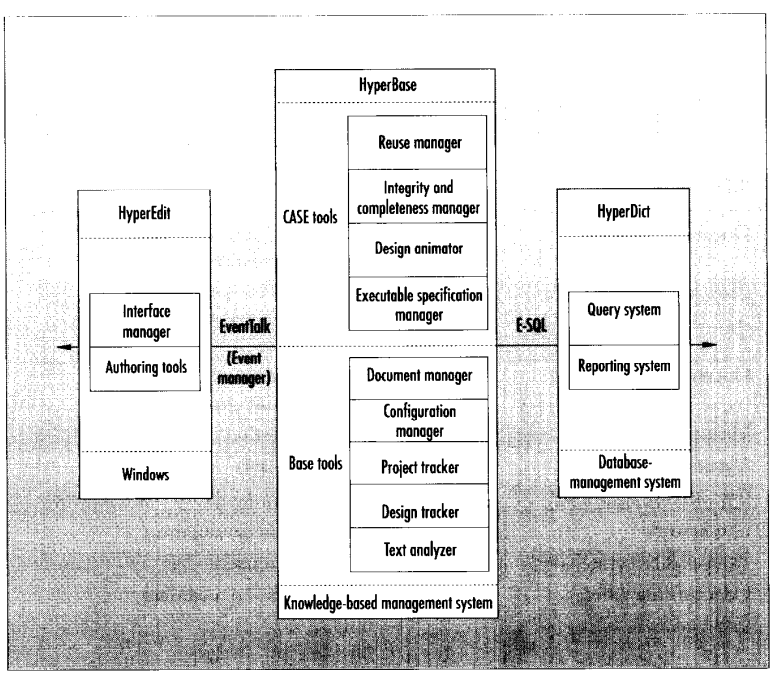

Figure 2. HyperCASE architecture.

HYPEREDIT

HyperEdit integrates many of the window-based, customizable graphics or text editors/browsers that a software engineer is likely to use in creating, modifying, and presenting software documents. More important, it can also generate such tools. HyperEdit — which comprises the interface manager, authoring system, and event manager — lets you construct, edit, and display a variety of documents, including requirements statements, dataflow diagrams, entity-relationship diagrams, structure charts, state-transition diagrams, Petri nets, flowcharts or Nassi-Shneiderman diagrams, source code, and test cases.

The main responsibility of HyperEdit's interface manager is to manage HyperCASE's graphical user interface under a native windowing system. The authoring system lets you construct software documents of hypertext buttons, which you activate while interacting with the document by typing or clicking or dragging the mouse. The event manager captures the button events and translates them into database updates, which it then sends to HyperBase or any other program via a special communication protocol, called EventTalk. In this way, HyperBase can interpret your actions in terms of the software design elements and interdocument relationships stored in its database.

Interface manager. The core of HyperEdit's presentation system consists of text and graphics primitives. These primitives let you easily express functions and flexibly create windows, dialogue boxes, menus, palettes, buttons, text, and graphics by using the mouse or keyboard.

Authoring system. HyperCASE offers several standard text and diagram editors that support major software-development methods. Each editor is simply an instantiation of HyperEdit's graphical environment. The instantiation is tailored to manipulate specific classes of objects with a range of attributes and behaviors. Authorized users can visually interact with HyperEdit's metaeditor to describe new object types and their subcomponents' shapes, sizes, colors, fonts, and styles; specify object behavior and the constraints governing the objects; and define the new editor's appearance and functions.

To construct a diagram or text, you

- compose software documents, of the editor-specific objects (like dataflow-diagram processes, stores, and entities);
- link them into object sets (with dataflows or into domains); and
- specify their logical attributes (like names) and visual attributes (like color, size, and position).

Authoring can be restricted according to the type of user. For example, analysts may be permitted to modify design documents without any restraints, while technical writers can be limited to altering the documents' visual attributes only, without affecting content. Programmers can be confined to *viewing* existing software design documents, with no ability to alter any document aspects.

The authoring system's browsing and navigational mechanisms include an extended button facility in which all diagram objects, regardless of shape and form, may act as hypertext buttons, thus becoming active document components. Button events are invoked not only when you click or drag a mouse, but also whenever a document's state changes as a result of modification, verification, or completion.

Other mechanisms include

- document retrieval by classification, content, or relationship with other documents in the system;
- document browsing with panning and zooming over individual documents or collections; and
- navigation among documents with history lists, route maps, bookmarks, and the like.

These capabilities give you rapid access to all relevant software documentation, which increases the chances of its reuse.

Event manager. All through the user's graphical interaction, Hyper-CASE maintains the data needed to describe developed diagram structures. Presentation records are kept in the HyperEdit repository. Conceptual information is kept in HyperBase or any other software that supports EventTalk.

The authoring system gives you rapid access to all relevant software documentation, which increases the chances of its reuse.

Although the presentation records and conceptual information (contents) are distinctly different representations, they have certain overlapping aspects. For example, a graphics field denoting an entity-relationship entity name will contain a string that will also be stored in the contents database. Consequently, information is replicated across the repositories of all currently opened HyperEdit sessions and HyperBase. For this reason we devised EventTalk, a special communication protocol that maintains the conceptual integrity and completeness of HyperEdit's multiple diagram images.

The main objective of EventTalk is to advise HyperBase of all user-instigated changes (as opposed to tool-instigated changes) to the document's content. Such changes may arise when a user creates, deletes, or edits a document's components (for example, creates or deletes entities, relationships, or attributes in entity-relationship diagrams). HyperBase can then validate these actions.

After HyperBase is updated (for example, when an entity has been deleted), the knowledge-base rules may be triggered to make additional changes to the database to ensure document consistency (deleting all relationships connected to the deleted entity, for example). In this case, HyperBase will also send appropriate EventTalk commands back to HyperEdit to correct the displayed document. Changes in document presentation that do not affect document content (like repositioning or resizing graphical objects) are not immediately communicated to HyperBase.

EventTalk also advises HyperBase of all user actions that cannot be interpreted within the context of the displayed document. Such events may include browsing and navigating by clicking on a diagram component like a dataflow diagram process. These events require access to HyperBase's interdocument linking information and the details of the destination documents. For example, you may need information about all the links leading from the selected dataflow diagram process to the set of its refinement diagrams.

EventTalk gives HyperEdit an object-oriented view of HyperBase even though a relational schema may be in use. It provides mechanisms for transaction rollbacks in the long transaction model, which organizes the undo facility but which also aids document versioning. The EventTalk transaction logs are useful in implementing a project-tracking system to assist in project and configuration management.

HYPERBASE

HyperBase is a knowledge-based hypermedia repository of software documents. All user actions performed within HyperEdit are reflected in and possibly extended by HyperBase. As Figure 2 shows, HyperBase comprises Base tools and CASE tools. Base tools accommodate the mechanisms and structures to organize a generic hypertext system, while CASE tools help ensure that a document is consistent and complete and that design components are reusable.

Base tools. Base tools evolved from our initial experiments with HyperCASE. At first, we planned to use hypertext principles in the implementation of only one CASE tool, a design tracker. But we real-

ized as development progressed that hypertext systems could satisfy many CASE requirements — particularly the need for multimedia document presentation; interdocument analysis, browsing, and navigation; version change and delivery control; planning and tracking; and capturing reflections about the product and its development.

We discovered that commercial tools could not deliver all the required features affordably. Hence, we decided to develop a whole suite of CASE programs that could provide the basis of a hypertext framework. This suite, which we call Base tools, includes a document manager, configuration manager, project planner and activity tracker, design tracker, and text analyzer.

Document manager. Software documents produced by HyperEdit become an integral part of HyperCASE. The document manager analyzes, indexes, aggregates, and stores graphics and text attributes to enable interdocument linking and navigation over program slices and refinement levels or across semantic terms.

Configuration manager. The configuration manager controls the current state of the knowledge base and determines semantic and temporal dependencies in the project's structure. Its role is to apply heuristics to ensure that system descriptions, their versions, and the products they define are consistent.

With a few notable exceptions (namely, Neptune[4] and DIF[5]), most commercial hypertext systems seriously neglect the need for elaborate version management.[6] The simple revision models applied to ordinary documents are inadequate to maintain complex software documentation. Software documents involve a number of additional constraints:

- compositional and referential dependencies (that result from make files, for example);
- document generation through executable programs;
- document verification through parsing and compilation;
- delivery sets (baselines);
- development distribution from collaborative development;
- the need for private workspace; and
- prototyping.

We believe that incorporating elements of software configuration management into the traditional hypertext model will benefit the hypertext mechanisms themselves. This belief led us to develop HyperCASE as a comprehensive set of customizable revision-control hypertext tools.

HyperCASE's design tracker forces designers to document their decisions and the reasons for them. Managers can then trace the evolution of system concepts.

Project tracker. Hypertext's ability to organize software into a complex structure of internal and external documentation opens new opportunities for planning software-development and tracking-development activities. HyperCASE's project tracker provides a suite of project-management diagram editors in which a project manager defines the tasks, required resources, milestones, adopted standards, and project deliverables. The project team can then use the relationships and dependencies from such a project plan to determine the structure of deliverable documentation.

The project tracker also helps project development by tracking system activities like tool use, monitoring resources allocated to a given task, and checking the status of individual documents and their components.

Design tracker. A major reason for the high cost of software maintenance is that development and delivery systems don't record or preserve the history of design activities. Completed software-engineering systems, in particular, often omit intermediate design documentation. HyperCASE's design tracker forces designers to document their decisions and the reasons for them. An integral part of HyperCASE's navigation system, the design tracker lets maintainers and managers retrace the evolution of system concepts, follow the reasons used to implement them in either chronological or logical order, and identify previous problem-solving attempts — thus avoiding potentially dangerous and expensive code and documentation modifications.

Like the configuration manager and project tracker, the design tracker provides a way for developers to precisely record their reasoning patterns. This feature makes HyperCASE a suitable environment for researching software processes because it identifies discrepancies between a developer's explicit design reasoning and the development plan implied by tool use.

Text analyzer. HyperCASE's text analyzer lets users access HyperBase through a restricted form of English. Unconstrained user queries are an alternative way to access software documents across the HyperBase data structures that support hypertext navigation. This alternative is attractive because although HyperBase is intelligent and comprehensive, it is highly structured and thus inflexible.

The text analyzer also makes it easier to recognize references to reusable concepts in requirements statements and to provide mechanisms for automatically classifying requirements and indexing and linking text for hypertext navigation.

CASE tools. Because HyperCASE's tools vary considerably in form and function, documents must be continually monitored to ensure consistency, integrity, and completeness. Software-development tools for assessing a project's state and condition must deal with volumes of incomplete and, in some cases, incorrect information. Typical problems include partial specifications, unfinished designs, error-ridden code, patchy documentation, and dated schedules and plans.

To remedy these problems, the HyperBase subsystem uses recent advances in artificial-intelligence applica-

tions to software systems.[7-9] The subsystem organizes a sophisticated knowledge base of reusable software documents and their components, provides heuristic rules to check their integrity, and aids their execution and testing.

CASE tools include a reuse manager, an integrity and completeness manager, a design animator, and executable diagram descriptions.

Reuse manager. Important to reusability — the fundamental aspect of software development — is having a uniform declarative representation of all document components that lets you see multiple views of the same component.

HyperBase provides such a uniform representation, which it uses to devise a set of knowledge-base rules about software and design reusability. The reuse manager offers a way to index and classify analyzed texts of requirements statements, design diagrams, plans, and schedules that lets you retrieve software documents relevant to the problem at hand with only a partially completed design or requirements statement. It also gives suggestions on how to incorporate the reusable components into the system being developed.

HyperCASE provides complete functional and structural mappings from the requirements specification to the construction of source and binary modules.

Integrity and completeness manager. Throughout the evolution of a project, each software entity is necessarily described from several vantage points. Relationships in entity-relationship diagrams, for example, may be defined as files or records in a data dictionary but appear as dataflows or stores in dataflow diagrams. Processes initially shown in HIPO (hierarchical input-process-output) charts may be refined with decision tables or state-transition diagrams, used in dataflow diagrams, and finally laid down as activity charts or program code.

When handling multiple software descriptions, you must analyze document semantics, correlate document content, check design integrity, and define completeness. Such tasks are usually laborious and difficult, frequently requiring the use of complex logic and heuristics. The integrity manager provides the inference framework for devising rules that ensure semantic integrity and document completeness. It analyzes the syntactic and semantic contents of documents produced over the life cycle, including the highest design levels.

Design animator. As system development progresses, the design animator monitors the sequence of systematic refinements to modules and data structures. In this way, HyperCASE provides complete functional and structural mappings from the requirements specification; through the design of program logic, flow, and control; through the definition of the data structure; to the construction of source and binary program modules. Once the programs are compiled and linked, you can exercise the code using a standard Unix symbolic debugger and visually trace the execution progress across all the system's diagrammatical and textual descriptions.

Executable diagram descriptions. We are also working on ways to create a software-diagram specification that is constructed in much the same way a circuit diagram is built —made from standard module families, and requiring no further logical or functional elaboration for implementation. We expect the systems described by tools supporting this diagraming formalism to be truly executable from early specifications.

HYPERDICT

All HyperCASE documents are stored in HyperDict, a common data dictionary.

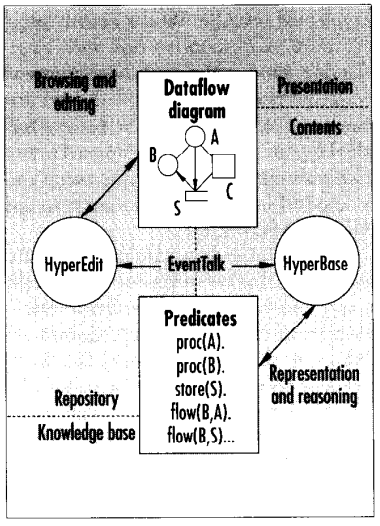

Figure 3. Document processing in HyperCASE.

Figure 3 shows how data dictionary updates are triggered by EventTalk transactions, channeled through the HyperBase knowledge base, and finally translated into the statements of embedded primitives in a high-level query language like Embedded SQL. This approach guarantees that HyperBase rules will check database integrity constraints or invoke them when necessary, thus keeping the database logically consistent. Several types of database operations — like reporting, querying, backing up, or recovering — do not alter the database contents and may be performed freely without any fear of constraint violation.

HyperCASE is our attempt to integrate a collection of disparate applications — namely, a CASE tool set — under a more general information-management and presentation paradigm, hypertext. We believe that this integration can improve concept reuse, simplifying the implementation of the CASE system and increasing its users' power and productivity.

The choice of tools has, in our view, been vindicated by exercises such as that

conducted at CASE '90, the Fourth International Workshop on Software Engineering. During the workshop, users and implementers catalogued the desired features of CASE systems. We believe that the HyperCASE project addressed a considerable number of the issues users considered of immediate importance to any future CASE research.

Our initial goals were to focus on front-end issues that we believe to be critically important yet lacking in sufficient attention from existing research teams. We have not, therefore, proposed the development of code generators or support for specific methods. The project team anticipates a number of opportunities for future development, providing direct support for the software tools being marketed by both Amdahl and Prometheus. This will take the form of tailored diagraming and design-capture capabilities that reflect the needs of individual development systems.

We expect HyperCASE to significantly decrease maintenance efforts as well as make it easier to cope with the large document collections typical of software development. We also expect the combination of natural-language processing, a design-reasoning record, and project tracking to substantially improve the economy of software development because it promotes design reuse and enhances project control. ♦

ACKNOWLEDGMENTS

We acknowledge the direct financial support of Amdahl Australia and of both La Trobe University and Prometheus Software Developments, as well as the assistance from the Victorian State Government. We also gratefully acknowledge the moral support and encouragement of Tharam S. Dillon and the members of the Amdahl Australian Intelligent Tools Program team: Kevin Alldritt, David Cleary, Mel Hatzis, Daniel Jitnah, Austin McClaughlin, Jane Philcox, Arthur Proestakis, Bev Teague, and Chris Wignall.

REFERENCES

1. R. Rock-Evans and B. Engelien, *Analysis Techniques for CASE: A Detailed Evaluation*, Ovum Ltd., London, 1989.
2. R. Rock-Evans, *CASE Analyst Workbenches: A Detailed Product Evaluation*, Ovum Ltd., London, 1989.
3. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Sept. 1987, pp. 17-40.
4. J. Bigelow, "Hypertext and CASE," *IEEE Software*, March 1988, pp. 23-27.
5. P. Garg, "Abstraction Mechanisms in Hypertext," *Comm. ACM*, July 1988, pp. 862-870.
6. W. Scacchi, "The USC System Factory Project," *Proc. Software Symp.*, Software Engineers Assoc., ACM Press, New York, Jan. 1989, pp. 9-41.
7. P. Carando, "Shadow Fusing Hypertext with AI," *IEEE Expert*, Winter 1989, pp. 65-78.
8. P. Garg and W. Scacchi, "Ishys: Designing an Intelligent Software Hypertext System," *IEEE Expert*, Fall 1989, pp. 52-63.
9. P. Puncello et al., "ASPIS : A Knowledge-Based CASE Environment," *IEEE Software*, March 1988, pp. 58-65.

Jacob L. Cybulski is deputy director of the Amdahl Australian Intelligent Tools Program and a lecturer in software engineering for the computer science and engineering department at La Trobe University. He is also a consultant and independent software developer. Cybulski's interests include artificial intelligence applications in software engineering, specifically interfaces and knowledge-based systems.

Cybulski received a BAppSci and MAppSci in computer science from the Royal Melbourne Institute of Technology. He is a member of the IEEE Computer Society, ACM, and American Association of Artificial Intelligence.

Karl Reed is director of the Amdahl Australian Intelligent Tools Program and a senior lecturer in software engineering for the computer science and engineering department at La Trobe University. He is also a senior visiting fellow in information technology, industry structure, and industry policy at the Royal Melbourne Institute of Technology. His research interests include general software-engineering issues, computer architecture, and industry policy.

Reed received an Associate Diploma from the RMIT in communications engineering and an MS in computer science from Monash University at Clayton. He is a fellow and honorary life member of the Australian Computer Society and director of its technical board.

Address questions about this article to Cybulski at Dept. of Computer Science and Computer Engineering, La Trobe University, Bundoora Victoria, Australia 3083; Internet jacob@latcs1.lat.oz.au.