CASE and Repository Technologies for a Multiuser Environment

Richard Barker

Payoff

CASE technology entails a combination of structured methodology, graphical workbench, code generator, reverse engineering tool, and the all-important multiuser repository. This article examines trends in CASE and repository technologies for the distributed development environments of global companies. The focus is on customized method support, portable generators, requirements for internationalizing systems, work group concepts, and integrated project management. An underlying open CASE architecture that joins all these facilities is also described.

Introduction

CASE and repository tools are essential for helping IS departments deal with rapidly changing business requirements and changing technology. Another concept that has gained favor, given the emergence of this technology, is the distributed development environment, in which the organization can see what each development unit is doing at any time. This article discusses the use of Computer-Aided Software Engineering and repository technologies to build applications for the multiuser environment. Among the characteristics CASE tools must exhibit are an openness to multiple methods and portability. Work group concepts for distributed development environments and the role of artificial intelligence are also discussed.

Customized Methods Enforcement

During the past 30 years, a wide range of structured engineering methods have been introduced. Most of these structured life cycle methods use a variety of well-known techniques, such as entity-relationship modeling, data flow diagrams, State-Transition Diagram, and function dependence diagrams. The use of these methods around the world is quite varied; the actual method used in a company is often determined by common sense.

Although some companies have standardized and adopted a single methodology, most users choose a hybrid that is based on one or more proprietary methods tailored to the needs of their organizations. There are also many companies that have, throughout the years, acquired different methodologies that are used in different parts of the organization. The requirement in this case is to let people use the method of their choice, to have it customized to a degree, and to enable multiple methods to work in parallel using the same information held inside the repository.

The same logical data requirement can be represented differently, depending on the method. In some cases, alternative representations are easy to provide by tailoring the object representation on the diagram. With a data model, for example, it is useful to show that one type of data (e.g., an employee entity) may inherit all the properties, relationships, and capabilities of another (e.g., a person entity). This is called subtyping. In Oracle Corp.’s CASE*Method and James Martin's Information Engineering methodology, subtyping is shown on a diagram by putting boxes within boxes (e.g., the employee box would be shown inside the box for the person entity). The Chen-Merise method, however, would
create separate boxes related to the first box to describe these subtypes. More complex differences have to do with the semantics of the objects themselves.

These semantic differences are difficult to maintain within a shared repository. In the near future, alternative techniques that are offered by different methods vendors but that share a common semantic model will be able to work together. When a semantic difference exists, the tools will have conversion capability between different representations. Eventually, full coexistence will be possible when the underlying repositories become powerful enough to maintain multiple semantic models at the same time.

**Portable Generators**

Currently, most Computer-Aided Software Engineering tools hold a definition of the business requirement using the conventions of a single method. This definition is then used by data base and program generators to target a particular operational environment. A new breed of portable generators will be able to work with a requirement that is constructed by a variety of alternative methods. These tools will also be able to generate code in one of two ways.

One breed of new generators will be able to generate code using fourth-generation languages, which are themselves inherently portable. Such a generator would disregard the operating system and window management system to a large extent, because these can be dealt with automatically by the portable languages used. If the generator targets a block-mode environment, layout and usability can be tailored to that lowest-common-denominator environment. If it targets a Graphical User Interface, however, layout or usability can be tailored to an environment that uses a mouse and contains select boxes, radio buttons, and other multimedia components.

The other approach is to use a logical module that is targeted at one or more generators, each of which is tailored to a particular target environment (see Exhibit 1). For example, it is possible to take a logical module and target it at Common Business Oriented Language generation for Canadian Independent Computing Services Association on an MVS environment. This same logical module can also be targeted on a generator for C, which takes into account the constraints and facilities available under Windows 3 on a microcomputer environment. As long as the generators can achieve 100% generation of code within approved standards and constraints, this mechanism is perfectly acceptable.

**Portable Generated Applications**

**Regeneration of Code**

Although generators are also emerging that have the ability to regenerate code, they will be unable to generate everything that a human being can envisage. Software engineers and designers will always be needed to tailor the appearance of the user interface and the aesthetics of key programs to best meet a specific user's needs or to take advantage of some technology not yet known to the generator.

This programmer-added value has an associated problem, however: if programmers make changes to code that has already been generated, what happens if the business rules change and the program has to be generated again? New generators can accommodate code changes as well as layout, style, and other aesthetic changes made by a programmer and
yet still allow the regeneration of the code to take place should the business rules held in the repository change. This gives the combined benefit of the rigor of a repository-driven, generated system and the added value that a human can provide to make the system acceptable and appropriate to the user community.

**Creating Global Systems**

In many large organizations, the core or common systems may need to be implemented across state or national boundaries. The most obvious requirement is that screens and reports can be generated in more than one spoken language, yet this is not a simple task.

Within any language there are usually large regional variations as well as variations from country to country. The French spoken in France, for example, is different from the French spoken in Canada. Translation is not as simple as changing text, error messages, and help subsystems. Just changing the prompts can have a significant impact on an application: prompts tend to use far more characters in German than in English, for example, so such a change has an immediate impact on the layout of screens or reports. The standards used for dates, currencies, and other data formats also vary from country to country. Across Europe, different countries use different collating sequences.

Sophisticated applications systems often have warning messages that are tailored to the instances of data they are manipulating at any given time. Consequently, the repository and the generators that act on an error message must understand enough about the native language and have enough data to enable them to construct informative sentences that are fully intelligible to the user. In extreme circumstances, there may even be a requirement that some applications run concurrently in more than one language.

**Interface Design**

The problem of internationalizing applications is compounded during the design of interfaces and hypermedia. A simple example is an icon, which in theory is a shape or picture that is instantly recognizable for portraying a concept. In different countries, different icons are needed to represent the same concept. The repository may have to hold different icons for different countries, which can then be used automatically when screens are generated. Audio elements should be in the native language or dialect, using a male or female voice, depending on what is appropriate to the culture of the country in question.

Part of free-format text handling encompasses the ability to retrieve data on the basis of an understanding of the meaning of the words. Synonyms and homonyms, classes and topics, and proximity searching must be considered so that relevant data and information can be retrieved according to queries on textual information. In an international context, systems may need to hold within their data bases text from several national languages and have text-retrieval capability that can switch dynamically between languages.

Finally, localization may be a key issue in generating applications that are to be used in different states and countries. The generated rule bases that drive the processing must match corporate policies and the needs of the company. These rule bases may also need to be enhanced with the ability to have a local variant for the policies, rules, and legislation of a particular state or country. This is especially important for a payroll system, for example, that applies rules and legislation for many subsidiaries and countries within a global enterprise. Consequently, the repository and Computer-Aided Software Engineering generator must be able to hold, manipulate, and configure systems to cater to this more global style of application systems.
Many of the problems associated with building global systems apply not only to the user systems but to the development environment. There is theoretically no reason for a large multinational company with offices in Japan, France, and the US not to have each member of its staff working with a repository or CASE interface that works in the user's national language. Most of the facilities needed for internationalizing applications apply to CASE tools and their user interfaces and to the knowledge base and data that drive the underlying repository.

Workbench Directions

In the battle for the preferred Graphical User Interface and the preferred workstation of the future, it is unlikely that one standard and one supplier will emerge. The real requirement is to have Computer-Aided Software Engineering and other tools that can easily and transparently move from one window manager and one workstation to another. Consequently, these tools must be written against an international or de facto standard (for manipulating such environments generically) and then configured transparently to work with the chosen window manager and workstation at runtime. Several vendors are tackling these problems by producing generic graphics toolkits. Oracle Corp., for example, has a graphics toolkit product that enables the tools using it to transparently switch between Open Look, Motif, Windows 3, and NeXTstep and to manipulate drawing services, icons, radio buttons, and images.

Besides the user interface, there is also a requirement to handle ever-larger sets of data to cover an enterprisewide requirement. All limitations must be removed to effectively give systems engineers an infinite drawing surface for each diagram. Another requirement is the ability to have multiple subsets of such diagrams, each of which is targeted to a different subject area. Within a few years, it will be possible to use these diagrams in a multiuser environment and have full synchronization of concepts held on different diagrams in real time. Much of this capability is already available from some vendors. This level of sophistication is necessary to have integrated project management capability with what-if capability.

Facilities for Work Groups

The individual facilities and features described in this article will not have the desired effect in a company unless the right people in the development project can use them effectively. There is a major move by several Computer-Aided Software Engineering and repository vendors toward what is known as integrated project management. This is just part of the support capability needed to enable small groups or teams of people to work together effectively.

In many large development environments, several hundred development staff members may be working in teams of 30 or 40 people to develop large systems over a two- or three-year period. As Computer-Aided Software Engineering technology evolves and expanded capabilities become more readily available, it is possible to expect the same output to be produced by teams of five or six people in three to six months. When this change in development projects occurs, individuals will find themselves working on more than one team simultaneously. A typical scenario is one in which a team leader on one product is responsible for a peer-group check of the design of another team; this same person may be writing an important piece of reusable code for a third team. To help people work together effectively in such an overlapping team environment, repository and
Computer-Aided Software Engineering technology will need to include some specific facilities.

**Interactive Multiuser Repository**

First and most important is the concept of an interactive multiuser repository, possibly distributed, so that all developers and managers can obtain an up-to-date picture of the entire development environment. They then need access control and filtering mechanisms so that they can easily get at the information they are allowed to use, be restricted from information to which they are not privileged, and be protected from making inadvertent errors that would affect their colleagues.

**Integrated E-Mail**

Communication within and between teams is quite critical. Integrated electronic mail and other electronic communications devices will become the norm, allowing team members to talk to other team members electronically and enabling the CASE and repository tools to alert team members to problems that have been identified. The different tools may also communicate with each other through electronic mail facilities.

**Integrated Project Management**

The aspect that will change the style of development work most dramatically is the concept of integrated project management. A workstation interface may, for example, display three diagrams, two of which may relate to a multiuser repository-based project management capability that can handle overlapping projects and resource use. This type of system would be able to compare the details of a user's work and allocations of the user's time for multiple projects, depending on the skills and roles the user is performing. This capability would make it possible to reschedule three projects but adhere to the schedule for the fourth most critical project, following the change in availability of key personnel or other resources. This level of sophisticated project management interface requires, in its own right, many of the capabilities described previously. Even greater benefits will result as these capabilities are integrated with other CASE tools. The project management interface may even display a third diagram, a data flow diagram, with the repository helping to coordinate the project management and Computer-Aided Software Engineering diagrammer being used.

For example, the designer may begin the day by asking the repository, through the project management interface, what task should be worked on. If the developer is told to complete a data flow diagram, when the developer has invoked the task, the system automatically loads the relevant data flow diagrammer from the chosen tool (preloaded with the appropriately named diagram) and readies the developer to complete the task. The developer moves the mouse to activate the data flow diagram and completes it before changing the status of the diagram to COMPLETE. The simple act of changing the status triggers the project management system, and if it finds that this is the final deliverable to this task, the task is implicitly completed as well. The project management system then checks the dependency network of tasks and sends an electronic mail message to the person who is to execute the next task online. If the dependent task is an independent quality check of the data flow diagram, a copy of the diagram can be mailed along with a reminder.
This development environment will radically change the way development teams conduct their daily work. Such an environment provides instantaneous access to a multiproject repository, up-to-date progress on tasks, quality measures, and other controls. Many people may perceive this style of working as a threat to their personal careers; however, this use of technology will enable progressive IS departments to work more productively, and they will be the ones that can most responsively meet the needs of their user community.

**Artificial Intelligence**

The final technology that must be considered is that of artificial intelligence. Many IS professionals would say that it will still be several years before AI technology becomes fully effective anywhere other than in particular niche environments. Part of the reason for this opinion concerns the difficulties of setting up the rule bases on which such systems are driven. Other problems concern the technology's portability and its general acceptance in the commercial sphere. Some form of computerized intelligence can, however, be made readily available through development tools and, for that matter, within generated systems.

The simplest form of intelligence is the appropriate use of prompts and reminders on interactive screens, which can be easily enhanced by the use of help and tutorial facilities to remind users of alternatives when they are using this particular part of a development or operational system. A simple extension can be provided by integrated constraint management of errors or warnings, which could also be enhanced by lists of allowable alternative actions.

Computerized intelligence can also be provided by powerful rule-based utilities that make key decisions about how tasks ought to be done. Examples of these include automatic data base design, automatic screen layout, and adherence to chosen installation standards.

A new form of intelligence is emerging in the form of business templates for requirements and design. For example, one personnel system is generically very similar to any other: all would contain people, jobs, roles, and organization units. Repositories may be preconfigured with a range of business templates that hold the data models, function models, and other representations for one of these generic business areas. With the expertise of world-class business analysts and designers used to build such templates, it would then be up to the company to simply find the differences between the templates and its users' needs and to make corresponding modifications before generating the system.

Many other forms of computerized intelligence will appear during the next few years as vendors strive to find different ways of solving old problems. Certainly the classical expert system capability will continue to be used for pattern matching to enable the Computer-Aided Software Engineering tools and repositories to identify areas of commonality or reusability.

All these forms of computerized intelligence will require the support of repositories (to handle sophisticated rule bases and metadata) and of server and workstation technologies (to provide the tremendous computer power to enable this level of sophisticated processing to be carried out fast enough to be fully exploited by the systems engineers using these facilities). Luckily, at the rate workstation technology is changing, there is, in many senses, an infinite amount of storage and processing power available for this style of task. (In the past three years, the average user workstation has changed from a 3-MIPS device with 4M bytes of main storage and a 100M-byte disk unit to a 50-MIPS workstation with 32M bytes of main storage and 500M bytes of local disk space.)
An Open Case Architecture

The key to successfully enabling these repositories and Computer-Aided Software Engineering facilities to work together in an open environment is the underlying architecture of software that joins it all together. Exhibit 2 shows an integrated open Computer-Aided Software Engineering architecture.

Integrated Open CASE Architecture

At the bottom of the exhibit is a repository, which would typically be held on a relational or object-oriented data base management system. This repository would be portable to multiple hardware platforms and would have distributed data base capability. Ideally, it would also have an open interface to other similar repositories on both a file-to-file transfer and interactive basis.

Held within the repository is all the data that is required by the systems engineers and the tools that use the data. In addition, the repository holds the national language rules and configuration details about targeted software and hardware environments as well as the business templates and other knowledge bases to drive the facilities described throughout this article.

The various repository services would be used to support a range of tools (including workbenches from different vendors, powerful generators, and reverse engineering tools) and would encapsulate other state-of-the-art technologies, such as text engineering, fourth-generation languages, and a variety of data base management systems. The tools would support one or more chosen methods, so the templates held within the repository may also include methods templates to help drive and control these tools. To provide the service to the systems engineers, a Graphical User Interface is required to transparently enable the tools and technologies sitting on top of the repository to be represented in their chosen window manager.

Independent of all of these tools and services, the repository would also contain the templates to drive project management and Configuration Management services. These templates can be used to coordinate the use of tools and other resources during the building and maintaining of sophisticated systems.

The real challenge with this architecture is to define the interface between each of the layers. These open interfaces then should be adopted by the international community and by the different vendors.

Conclusion

The evolution of Computer-Aided Software Engineering tools and sophisticated repository services, as described in this article, may depend on whether the different vendors decide to work together or whether they attempt to lock the customer into their own solutions. To date, even with standards that have been around for some time (e.g., Structured Query Language), only a few companies have fully complied, and even then they often have supersets of requirements that enable them to differentiate their products from those of other vendors.

The only answer to these serious issues is the international standards bodies; they can create a powerful yet lowest-common-denominator standard, which can then be required by companies when they request bids for products. Only if this occurs will IS departments have the freedom of choice to use the tools, methods, techniques, hardware, and user
interfaces that best meet their technology aspirations and budgets and that enable them to support their organizations in these changing times.

**Author Biographies**

Richard Barker  
Richard Barker is a senior vice-president of Oracle Corp. responsible for the company's development method. He runs several software development units, including those that create CASE tools, text retrieval software, and human resources applications packages built using CASE.