MARKET SMART

Considering CASE: Write the Fine Print

A consumer's guide

Front-end CASE, back-end CASE, I-CASE -- with so many companies promoting a large number of CASE (computer-aided software engineering) products, it might be helpful to scrutinize some key characteristics of CASE tools. By discussing these factors, parties interested in acquiring CASE can accelerate vendor evaluation.

BUYER BEWARE -- PREPARE

Before meeting with potential CASE vendors, you should resolve several internal issues. First, determine the hardware and software environment in which the CASE tool will be expected to execute. Will it run on workstations, on a mainframe, in a network, on a local area network, or on multiple platforms? What are your operating systems, teleprocessing monitors and application languages? If you're not sure, you are not prepared to talk to a vendor. Of course, if you're willing to be persuaded by the vendor, you can be sure they've each got a solution that will fit just right.

Next, assess the level of vigor the software engineering staff is willing to accept and the willingness of management to support the introduction of change. The sophistication of CASE tools varies widely; no single product will meet everyone's expectations. Since choosing a single robust product set will eventually spawn the most quality and productivity results, do not be inveigled into selecting a middle-of-the-road product which appeases the majority of the staff through an averaging scheme. Select a product that challenges at least 80 percent of the targeted software engineering population.

CHATS WITH A VENDOR

Always be specific concerning the characteristics that are needed and when. Getting lost in strategic directions, future releases, and evolving products is too often the rule and not the exception. And consider the whole package: The toolset, the vendor's ability to provide training and consulting support, ongoing maintenance costs, equipment upgrades, and internal support costs. Be cautious of vendors that offer pieces of the package and "associates" who supplement the remaining pieces. Consider training on-site, in your environment, unless attendees will be unable to break away from daily tasks.

TIPS ON TOOLS

As you review vendors and products, several factors are critical and deserve close attention.

- **A methodology** should include a set of rules which is comprehensive and verifiable through consistency and completeness checks. This definition is not well suited to some vendors who describe their diagramming technique as a methodology. Indeed, methodology encapsulates the entire systems development lifecycle from planning through construction maintenance.

  Also, to be a CASE component, the methodology should be automated, rather than a series of leather-bound three-ring tomes resting in the software engineer's bookcase. An automated methodology provides reassurance that the product is integrated.

  In a CASE tool, more rigor is generally better than less as it implies completeness and self-enforcement. Of course, the appropriate level of rigor will still depend on the sophistication of the software engineers who will apply the tools. Still, a methodology that is "flexible" enough to allow system changes at several phases in the lifecycle lacks rigor. In contrast, a rigorous system will guarantee that any system change is introduced in the preconstruction phases of the lifecycle. Increased rigor reduces the opportunities for unintended changes to the system.
- **An encyclopedia** (sometimes referred to as a repository) contains all the metadata for CASE-developed systems, such as analysis and design specifications; edit criteria; element, record, and table definitions; code module location and reuse relationships; cross-referencing query capability; impact (change) analysis; and product documentation. By providing versioning capability, the encyclopedia allows different levels of generated software to be adequately tested before they are granted production status.

  The encyclopedia of a truly integrated CASE product will not require that metadata be reentered across phases or within phases, thus avoiding many opportunities to misdefine predefined data.

  Another test of a product's integration is the ability to move between the mainframe and the workstation for workstation-based updating. In addition, the centralized encyclopedia (either on a mainframe or file server) needs to maintain access control as developers simultaneously access it. Data integrity between the mainframe and workstation encyclopedias should be enforced by the encyclopedia rather than by manually-invoked import/export rules. Dependency on import/export rules indicates both that the product does not enforce its own integrity and that it is not integrated across the lifecycle phases. Measure the encyclopedia's "activeness" by how well it enforces the methodology for all its entries and updates as they occur.

MISMATCHED PIECES

Many vendors' current product sets aren't integrated throughout the entire software lifecycle -- from planning through retirement. An integrated product will not require software engineering activities to be repeated in subsequent development phases and will have a consistent interface.

Be cautious of vendors that offer pieces of the package and "associates" who supplement the remaining pieces

The consistent interface should include the following features across the entire product line: similar screen layout and coloring, identical function key invocation, consistent application of encyclopedia rules; consistent menu presentation and selections; oneness in appearance and function of pull-down menus and options; identical icon representation and function; and mouse support throughout the phases. An integrated CASE toolset will give the appearance and performance of a "single tool."

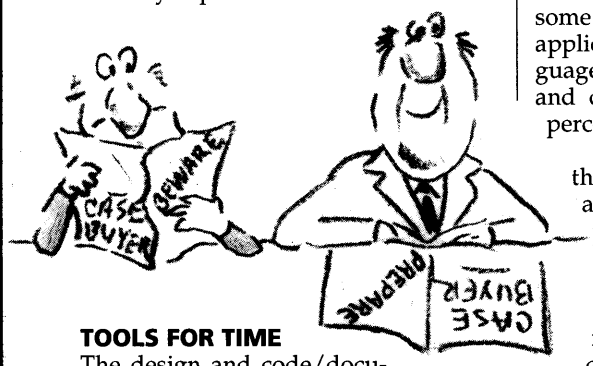
Unlike their integrated cousins, "interfaced" CASE products rely much more on manually-invoked processes to supplement development tasks. For example, a product set that requires "import/export" of data between phases or tools is clearly only interfaced.

Early in the CASE movement (around 1986 and 1987) not even a handful of vendors could claim a full-lifecycle product. In an effort to compete with these tools, many front-end and

back-end CASE vendors merged to gain each other's capabilities. While mergers continue to occur, integration of encyclopedia capabilities, the developer interface, and methodology have been slow to follow.

FROM CRADLE TO GRAVE

For full lifecycle coverage, a CASE tool will include components for each development phase, all of which have an active link to the encyclopedia for realtime access and integrity checks among data structures and processes. Only realtime access between all the CASE components and encyclopedia can provide the necessary integrity enforcement. Also, integrity needs to be applied at both local and global levels. Local-level integrity validates new data elements and processes within the system under development; global integrity assures consistency for all systems defined to the encyclopedia.



TOOLS FOR TIME

The design and code/documentation generator components of the CASE toolkit address the most time-intensive software engineering activity and determine how well the finished product satisfies requirements. For optimal resource use, the design component should be workstation-based and mouse-driven.

In those cases where the design tool is supported by a prototyping tool, the prototyping tool's tie to the encyclopedia and its functionality should be consistent with the design tool. Since processes are captured in the design phase, no generated code can be created without being captured in the design component and encyclopedia, so that all code and documentation remain consistent and current.

RESULTS VARY

A full lifecycle product will generate some degree of code and documentation. Some products generate the equivalent of Cobol copy libraries that contain file descriptions and record layouts. Even the most unsophisticated products can complete this task.

Other products generate skeleton programs wherein Cobol identification, environment, and most of the data division are generated. Processes completed in the design phase using tools such as Warnier-Orr are converted to program code but are only as complete as subprocesses entered in the design phase. These code generators are as powerful as the functionality of the encyclopedia and the level of integration provided by the product.

Another group of CASE products can generate 100 percent of the application code, 100 percent of the time for a target environment. Be careful with this degree of code generation however. Frequently, in order to construct and/or test the generated system, additional products are required that are not included with the product; which causes loss of integration, and additional products to support at added cost. Lastly, some CASE products generate all the application code, data definition language, data manipulation language, and control language statements 100 percent of the time.

To assess a vendor's strength in the area of code generation simply ask: "Does the product generate 100 percent of the application 100 percent of the time?" Also ask how many complete applications generated with the product have never terminated abnormally. Of course, if the application for which you are considering CASE does not require that the code be correct or complete, there are a number of CASE tools that will help.

In transaction-driven environments that aim to minimize hardware upgrades, the need to generate code for a compiled, rather than interpreted, language is critical. Almost all CASE code generators for information systems environments generate Cobol code. Be especially cognizant of the few that do not, those which opt instead to generate proprietary fourth-generation language (4GL) systems.

NOT FOR PROTOTYPING

While quicker than hand coding, CASE-generated, compiled applications are not suitable prototyping tools. Imagine sitting with a customer prototyping a very small 20,000 line system on a workstation. As the customer requests a change to the system, you reply, "We can look at that change in about 15 minutes after I generate the system" -- a response that is not conducive to iteration. To support early requirements definition with the customer, prototyping should accommodate data entry, reporting, and menu screens; screen flows; edits; reports; and queries.

Documentation's currency is at least as important as its extensiveness. Enforcing changes in the design phase ensures that documentation is up-to-date. Application changes at the coding level perpetuate the primary cause of outdated documentation -- hardly enough time to change the code and no time to update the documentation. Useful system documentation should include as a minimum: process, screen, and report descriptions; processing flows; edit criteria; construction test results; and impact analysis.

NEGOTIATE FOR TRAINING

Heavy doses of training and frequent on-site consulting are required for successful CASE implementation. Include these needs in the negotiation process; they represent a substantial investment. Most CASE vendors supply their own training; other vendors believe that someone else might deliver better training. A CASE vendor who supplies the product and training could have an advantage due to integration, familiarity with future releases, and focus. Past experience indicates that consultants, with public and formal relationships with a single vendor, are more than willing to help implement alternative vendors' products.

BACK TO REQUIREMENTS

Reverse engineering, returning a product to an earlier phase (from construction to design), may eventually allow developers to return systems to the requirements and then to the planning phases. Reverse engineering, even more than forward-engineering, would attack the systems that gobble up 60 percent or more of IS staff resources.

Whether purchasing CASE tools for existing systems or for new developments, some care is indicated. If vendor promises seem too good to be true, they probably are. While CASE tools can offer productivity gains, their purchase, like that of any other product requires caution. *Caveat emptor!* ■

Joseph R. Schofield, Jr. is a computer systems consultant at Sandia National Laboratories in Albuquerque, New Mexico. His responsibilities include application of emerging technology into the software development process, and software quality and metrics implementation.

© United States Department of Energy contract number DE-AC04-76DP00789.