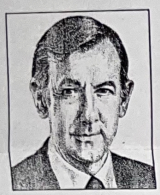
APPLIED INTELLIGENCE

Use of Automated Tools Crucial to RAD Life-Cycle Success



JAMES MARTIN

This is the fourth in a series of articles on rapid application development (RAD), a methodology designed to be much faster than traditional methods.

Improved applications-development life cycles are urgently needed to develop strategic applications more quickly, as well as to deal with the growing backlog of applications waiting to be developed. Information systems need to be retooled with techniques that can develop applications in months rather than years, days rather than weeks.

The RAD life cycle addresses these concerns through a combination of highly focused management techniques and advanced applications-development technology. Unlike more conventional development life cycles, RAD emphasizes the use of small, highly motivated teams of users and information-systems (IS) professionals, as well as extensive use of interactive, joint application-design techniques. Applications development is performed in an iterative manner using integrated computer-aided software engineering (CASE) tools capable of generating code for complete applications.

The success of the RAD life cycle depends greatly on the use of automated tools. Organizations that have achieved high productivity with RAD typically use tool sets, such as integrated CASE (I-CASE) tools, to rapidly build applications within an automated life cycle.

These integrated tools, developed in the late 1980s, permit entire applications to be specified on the desktop. Fully integrated CASE products provide a complete software-development environment that supports the entire life-cycle process.

A variety of tools—both simple and complex—have been created to help make IS development faster, cheaper and of higher quality.

Simple tools should be used to build simple systems—a complex tool may slow its development. (For some needs, a report generator or a spreadsheet tool is sufficient.) Most systems developed with RAD techniques, however, are likely to be complex and will require sophisticated tools.

In the early 1980s, fourth-generation languages were invented. Non-procedural languages made it possible to express the required result, rather than simply the way to achieve it. Structured Query Language (SQL), which provides a non-procedural way to access relational databases, became a standard. And other, more user-friendly query languages and report-generation languages proliferated.

Prototyping tools became important, enabling developers to build prototypes quickly and see how users reacted to them. Prototyping languages gave rise to iterative development in which a prototype was successively refined. CASE tools provided graphically oriented ways of expressing models and designs. Code generators, which could generate COBOL or other languages from high-level constructs, were also created.

Recently, these tools have been combined into powerful, integrated facilities. CASE tools for planning, data modeling, analysis and design were integrated with code generators. Prototyping capability was linked into the design tools. And non-procedural languages, including SQL and report generators, were integrated into the CASE environment.

The most important feature of I-CASE is the ability to generate code directly from the CASE design tool.

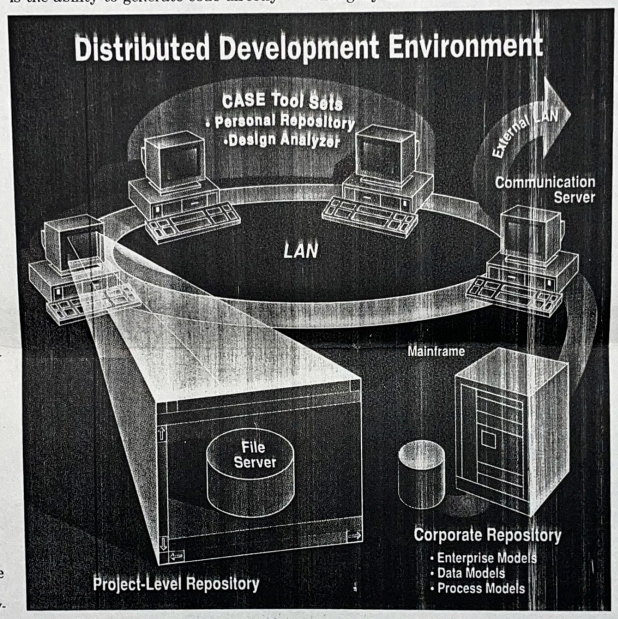
The figure shows a typical distributed I-CASE environment. Members of a project design team usually work within a LAN. Each member has on his or her desk an I-CASE tool set with its own repository and design analyzer. With the desktop tools, they can do planning, analysis, design and code generation.

Consolidation and analysis of specifications across the project is performed using a project-level repository that typically resides on a file server within the LAN serving the project.

In addition, there may be a central corporate repository, which is usually on a mainframe. The corporate repository typically stores enterprise models, data models and process models that are to be standard design components across the organization.

The developers, who can be miles from the mainframe, connected to it by telephone lines, download a subset of the central repository design information into their local project-level repository, where it is accessible via the LAN file server to all members of the design team.

Individual members of the team can then transfer this information to their desktop machine and work with it locally. A desktop design analyzer checks the integrity of what is built.



John Avakian

Simple tools should be used to build simple systems. Most systems developed with RAD techniques, however, are likely to be complex and will require sophisticated tools.

Periodically, the changes made to the subset are sent back to the project-level repository, where they are consolidated with specifications from other members of the design team, under project-management control. A design analyzer then goes to work on the consolidated specifications, detecting any discrepancies among the different analysts' work.

A critical characteristic of an I-CASE tool (as opposed to a CASE tool) is its ability to generate executable programs. A code generator is driven by the design workbench.

The tight integration of the analysis and design tools with the code generator results in much higher productivity than do uncoupled tools. The integrated tool set is the basis for RAD.

In advanced I-CASE tools, all of the functions associated with planning, analysis, design, consolidation of specification, analysis of specifications, code generation, database generation, documentation generation and project management can be performed within a LAN using a network of desktop computers.

No longer is there any need to interact with a remote mainframe computer, except to access shared, corporate-level design specifications.

Formerly, code generators for I-CASE tools were located on a mainframe; now they are rapidly moving to PCs. Today's PCs are powerful enough for complete development, code generation and compiling of programs that eventually will be executed on a designated host computer.

Using I-CASE tools with desktop code generators is generally faster than accessing a mainframe for code generation and compiling.

A developer should be able to design a system (or subsystem), generate code for it, test it, modify it and regenerate it as quickly as possible. In addition, he or she should be able to do this on a desktop machine, completely debugging the logic on that machine and generating the linkages to the mainframe databases, the network and the operating system.

Once fully tested on the PC, the code is handed over for execution and testing on the host machine on which it will eventually run.

Complexity Calls for I-CASE

The more complex the project, the greater the need for I-CASE tools. As the complexity of a project increases, the gain in productivity, relative to the traditional COBOL life cycle, also increases. Complex projects have many components that need to be integrated; thus, the integrating capability of the repository and design analyzer becomes especially important.

The key to building complex systems is to have small, autonomous teams working simultaneously with powerful I-CASE tools, their work coordinated with a model that is in the common I-CASE repository.

People who have learned to manage the RAD life cycles with I-CASE tools look back on the earlier methodologies with horror. Quality systems simply cannot be built quickly with the traditional methods.

The components of integrated CASE tools used with the RAD life-cycle process will be discussed in more detail in next week's column. ■

The concepts embodied in RAD are described in a new volume in the James Martin Report Series. For more information on this volume, call (800) 242-1240. For information on seminars, contact (in the United States and Canada) Technology Transfer Institute, 741 10th St., Santa Monica, Calif. 90402 (213) 394-8305. In Europe, contact Savant, 2 New St., Carnforth, Lancs., LA5 9BX United Kingdom (0524) 734 505.