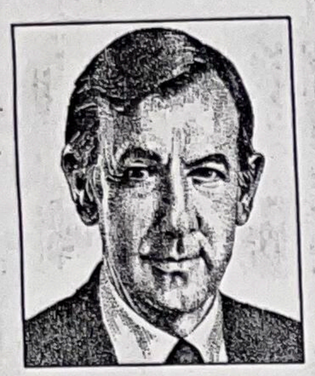
APPLIED INTELLIGENCE

Reverse-Engineering Gives Old Systems New Lease on Life



JAMES MARTIN

Powerful new concepts and techniques are becoming available to migrate the billions of lines of existing code into more easily expandable, maintainable systems.

In most enterprises, computing systems have large quantities of old code, most built by hand in an unstructured fashion. The data usually has not been modeled, with little or no interest in achieving data compatibility between different systems.

These systems typically were designed before today's principles of good database design were understood. The data isn't normalized and is unrelated to the data-administration process. As caretakers of these systems know all too well, these systems are fragile and expensive to maintain.

Traditional maintenance of programs is an unsatisfactory and expensive process. It has been compared with the attempt to repair a wooden boat at sea: New planks can be replaced only by using existing planks for support. The process must be done in small steps or the boat will sink. And sooner or later, the boat must be brought to a shipyard and rebuilt.

Using computer-aided software engineering (CASE) tools, old systems can be rebuilt from restructured design information stored in the CASE repository. They can't all be rebuilt quickly, because this would involve too much work. The best that can be hoped for is a steady, one-at-a-time migration of the old systems into the cleanly engineered form.

In corporations that have successfully implemented development methodologies based on integrated CASE (I-CASE) tools, these are two development worlds. The I-CASE world has the ability to evolve systems, continually improving their design and regenerating code. The systems are cleanly engineered, easy to change and have stable data models.

Alas, there is also an underworld of poorly structured old systems lacking data models. In many corporations, more programmers spend their time maintaining poor-quality code than those who work in the I-CASE world.

The problem is rather like the slum-clearance problem in a city. Despite a city's new center, elegant architecture and efficient street plan, existing slums and old, crumbling buildings still need to be maintained. Planners hope for a steady migration from the slums and their replacement with well-designed, new facilities.

Many corporations have attempted a major conversion of a file system to a database system and have failed. Usually, the reason is that the project consumes much more work than anticipated because so much has to be reprogrammed.

Often, the attempt to convert is killed by the people controlling information-systems (IS) finances. The conversion process itself creates no new applications. Management perceives a great deal of effort and expense with nothing to show for it. There is a long and serious application backlog.

In one organization after another, including some of the most prestigious data-processing organizations, the attempt to make a major conversion has failed. Yet the cost of maintenance in the non-I-CASE world escalates daily.

One of America's best telephone switches ran into so many software-maintenance difficulties that its maintenance costs exceeded $1 million per day. The switch would never have been built if that figure had been forecast.

A recent study by the U.S. Air Force estimated that unless maintenance productivity is improved by the year 2000, 25 percent of the U.S. draft-age population will be required to maintain the Air Force's software!

Fortunately, new tool sets to facilitate the complex process of rebuilding systems are available. The goal is to reverse-engineer old systems into a cleanly structured form, using tools that automate the tedious parts of this process. The programs are captured and represented in a CASE format so they can then be modified as required using CASE tools. Data definitions are captured and converted into stable data models stored in the CASE repository.

The figure shows a reverse-engineering step. The code of the old system is restructured with an automated tool and entered into an I-CASE tool so it can be analyzed and redesigned, and become part of the I-CASE evolutionary life cycle.

In connection with this reverse-engineering step, three terms are used:

Restructuring: conversion (with an automated tool) of unstructured code into fully structured code.

Reverse-engineering: conversion of unstructured code into high-level design specifications and (automatic) entry of these specifications into an I-CASE tool where they can be improved or redesigned.

Re-engineering: modification of the design of a system, adding functionality where required, and (automated) production of code for the improved system.

If reverse-engineering is done as shown in the figure, old systems can be rebuilt as evolutionary systems. Eventually, the messy underworld should disappear.

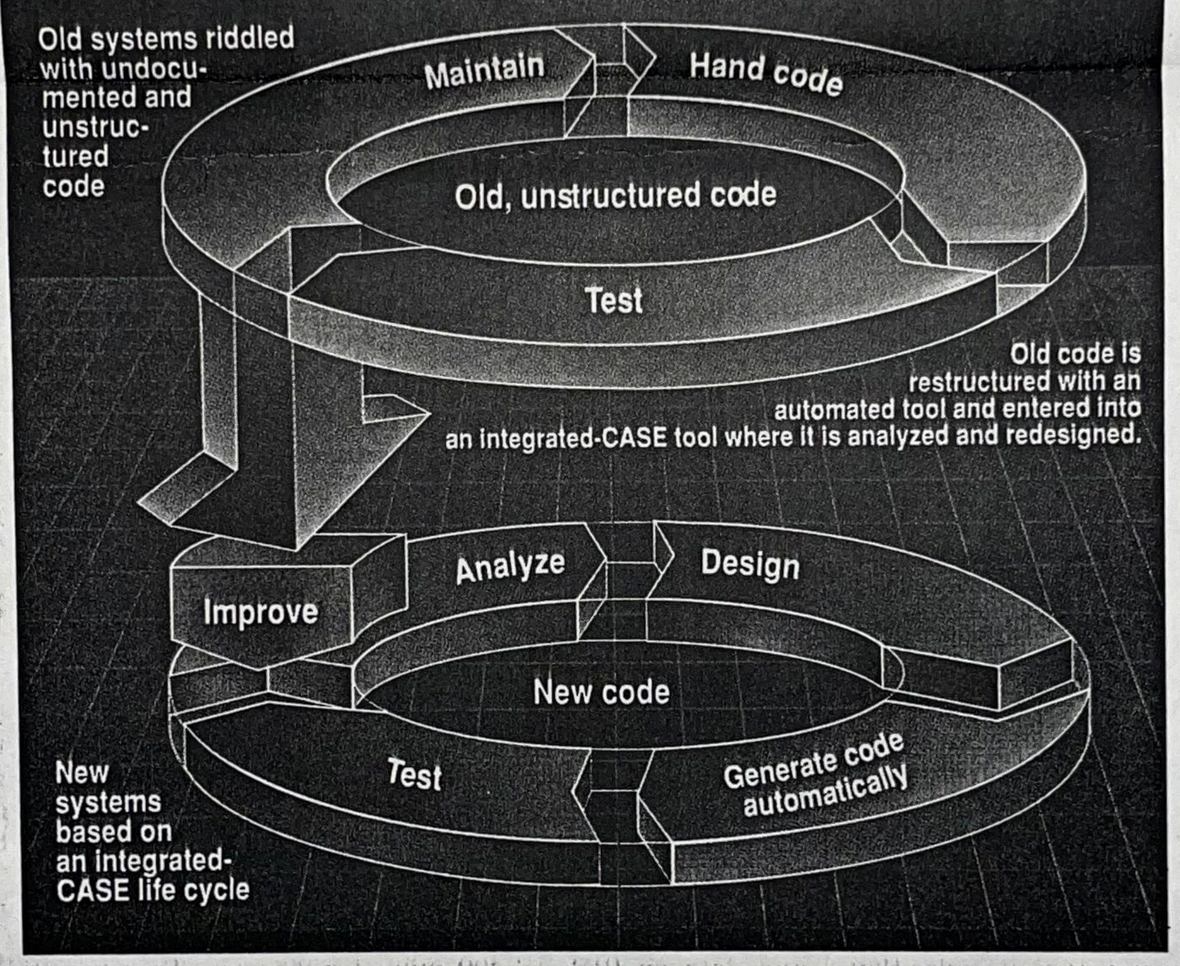
There are basically three approaches to the IS reverse-engineering and re-engineering problem:

- **Do Not Convert.** Allow applications to continue their existence unconverted but, when necessary, build a bridge to new systems using data models and process models stored in a CASE repository.
- **Restructure.** Quickly restructure the messy applications (preferably with automated tools), but don't rebuild them with CASE tools. The slum areas are improved, but not rebuilt to be part of the new planning. Often it's necessary to build a bridge to the new systems built with CASE tools.
- **Rebuild.** Reverse-engineer the old applications to conform to the data models and process models stored in the CASE repository. This is comparable to redesigning and rebuilding areas of the city.

In large IS installations, a mix of these approaches will probably be used.

How To Reverse-Engineer Existing Systems

Old Systems Can Be Rebuilt Using Restructured Design Specifications Stored in the CASE Repository



John Avakian

In one organization after another, the attempt to convert to a database system has failed. Yet the cost of maintaining old systems escalates daily.

To Convert or Not

Two questions should be considered before deciding to convert an old system. First, does it work? If it works well, there's a strong argument for leaving it alone. If it works inadequately, it should be rebuilt using integrated CASE technology. Second, does it incur high maintenance costs? If the application is fragile and expensive to maintain, then it's a candidate for restructuring or rebuilding with automated tools.

It may be appropriate to automatically restructure an old system and modify it by programming rather than rebuild it with reverse-engineering and I-CASE tools. If an application system works adequately and needs little maintenance, then its conversion should probably be postponed. Spend the effort on something else; there are so many other applications needed, such as a bridge between it and the new environment.

In planning data resources, it may be unwise to assume that old systems will be converted easily to the new, automated form. A realistic appraisal is needed of the costs and difficulties of conversion. The dismal history of uncompleted conversions should be weighed.

In their initial enthusiasm for new development methodologies, system designers often assume that the old systems will be converted and discover too late that they won't. It's safer to assume that many old systems will survive and plan a bridge that links them to the new world.

Next week I will examine the challenge facing vendors in building tools that support reverse-engineering. ■

The concepts embodied in reverse-engineering are described in the CASE 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.