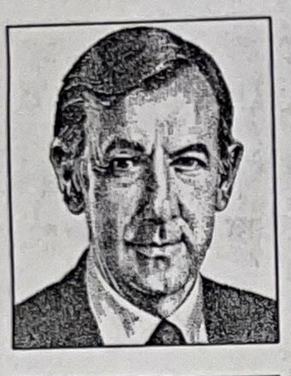
APPLIED INTELLIGENCE

Restructuring Code Is a Sound Investment in the Future



JAMES MARTIN

Among the most urgent issues in the CASE industry today is the need to develop better tools for restructuring, re-engineering and reverse-engineering existing applications code into more easily expandable, maintainable systems.

As outlined in previous columns, restructuring involves automatically converting unstructured source code into structured code; re-engineering is used to standardize data definitions, remove redundancies and anomalies in existing systems, and reimplement the application in a new form; and reverse-engineering is concerned with extracting high-level specifications from existing code in the form of data and process models.

Most computer-aided software engineering (CASE) tools currently don't provide facilities to restructure or re-engineer existing code; they are focused on automating the production of new applications. However, as much as 80 percent of the programming effort in many information-systems (IS) organizations is directed toward the enhancement of existing applications.

Tackling a Thorny Task

Forward-engineering from graphical design specifications using a conventional CASE tool is much simpler to enact than true reverse-engineering, which demands the extraction of specifications from existing systems. For instance, more than 200 CASE vendors support the conventional forward-engineering process, although very few vendors have tackled the thorny task of reverse-engineering existing applications. In fact, no vendor has yet solved this problem for both the data and process sides of an application.

Organizations are beginning to recognize the increasing importance of preserving the value of past software investment through re-engineering the existing software base. Many organizations have 10 million to 20 million lines of code that must be maintained and enhanced for many more years; replacing this large body of code using only forward-engineering techniques is impractical at best.

The restructuring of existing code is one of the simplest methods available to reduce the cost of maintenance. Using products such as Language Technology Inc.'s Recoder or IBM's COBOL Structuring Facility permits existing unstructured COBOL programs to be restructured automatically.

As shown in the figure, these programs take spaghetti-code input and produce cleanly structured code and documentation as output. The resulting structured programs are identical in functionality to the original unstructured programs; however, the structured code is typically 20 percent to 25 percent less expensive to maintain.

In operation, the restructuring tool executes a complex algorithm to automatically structure the spaghetti code. First, it reduces the program to an abstract control-flow design. This design is then analyzed and redesigned to achieve a cleanly structured control graph, which is then run through a COBOL generator to reimplement the structured design. Documentation is produced in the form of structure charts, control-flow diagrams and reports on problem areas in the code, such as unreachable code, endless loops and poor structure.

Specialized versions of these products are available to restructure IBM's Customer Information Control System program (CICS) commands, such as the Handle construct, which can cause hidden control-flow jumps whenever a particular type of error occurs. To provide an objective measurement of the quality of the code, additional tools such as the Inspector from Language Technology can be used to apply structure and complexity metrics to existing code.

Organizations such as the Hartford Insurance Group of Hartford, Conn., have used restructuring and metrics tools in support of an applications-maintenance center. The company has reduced maintenance costs substantially by using automated tools to analyze its portfolio of COBOL programs, identify poorly structured routines and automatically restructure these programs. Metrics tools provide an objective evaluation of the complexity and quality of each program module.

As illustrated in the figure, the restructuring process does not extract specifications from the restructured code; it stops short of making control-flow information accessible to the programmer.

Some restructuring tools do provide interactive access to the control-flow diagrams produced as part of the restructuring process. These tools, among them a new product called Chrysalis from Language Technology and the Via/Center product set from Viasoft Inc., extract control-flow information from the code and make this information available to programmers on line.

Using this information, programmers can evaluate the structure and organization of the program, how control is passed between modules and how logic and data flow through the program. The programmer can use this information to enhance and maintain the program at the structure-chart level.

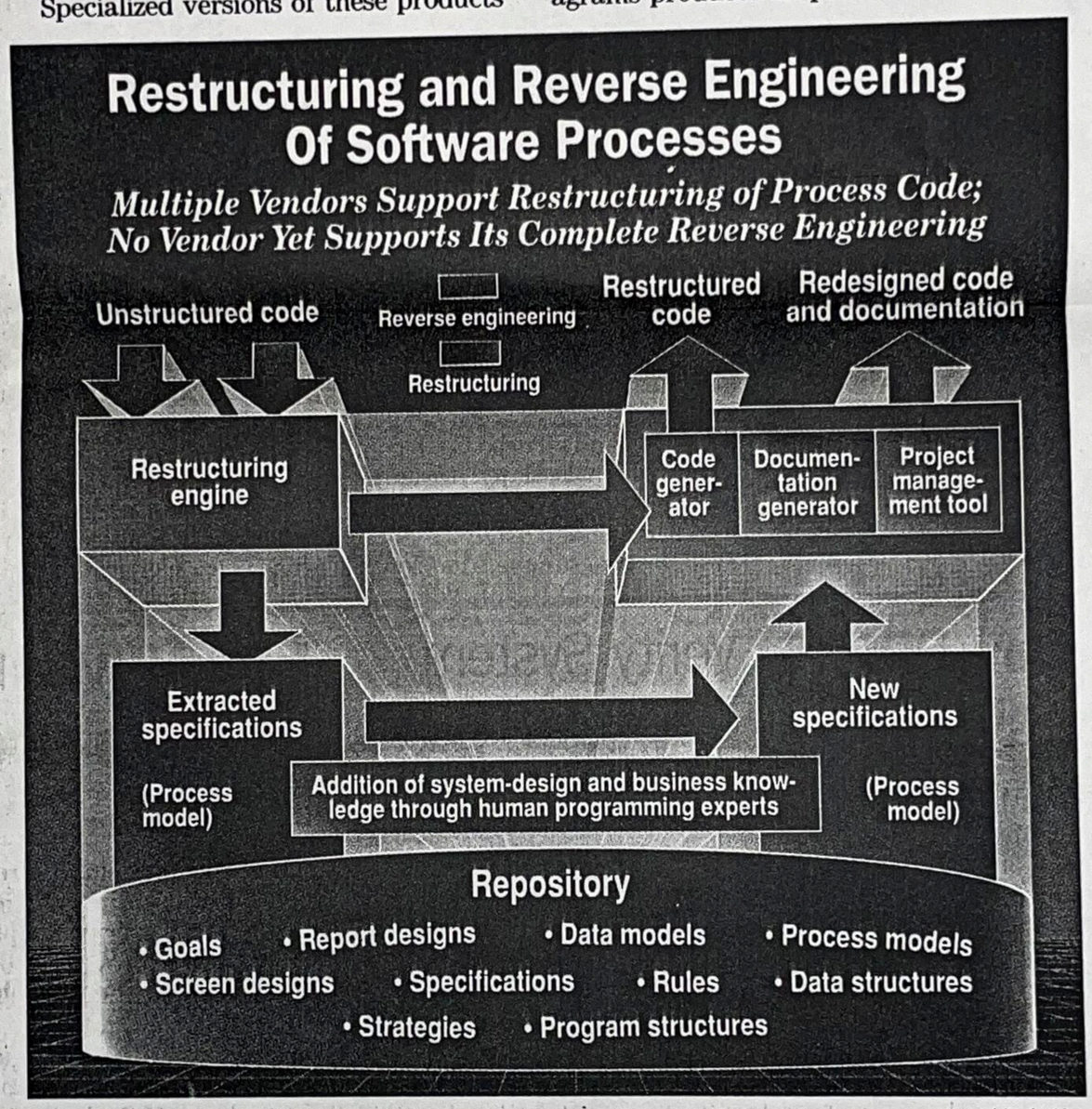
As shown in the figure, a desirable next step in the extraction of control information from existing code is to produce higher-level control abstractions, such as design tables, action diagrams and data-flow diagrams. The ultimate objective is to support reverse-engineering by extracting process specifications in the form of process models that are independent of the implementation environment. These process specifications can then be combined with high-level data models, which are stored in the repository of a CASE tool, and then enhanced at the design level and reimplemented in any supported environment.

It is desirable to build systems that will be easy to change in the future. A major goal of software engineering is to make systems easy to enhance and maintain. Applications built today should be fully based on normalized data models in a CASE repository.

To design systems that will ease future enhancement and migration, the following techniques are recommended:

- Use CASE tools for planning, analysis, design and construction.
- Use a code generator driven by the CASE design tools.
- Adhere to the principles and practices of information engineering.
- Create procedure designs that are independent of technology, which is likely to change.
- Employ a code generator that can translate the designs into implementations with different technology (for example, a tool that can generate code and data descriptions for different database environments).
- Use fully normalized data models.
- Use fully structured code built with an action-diagram editor that is part of a CASE environment.
- Use a relational database whenever possible.
- Use application standards such as IBM's Systems Application Architecture.
- Use a database-management system with field independence and features that enable changes to be made without rewriting existing programs.
- Avoid unusual hardware, operating systems or facilities that could make future migration difficult.
- Plan to do future maintenance by regeneration.
- Make upper-level management fully aware of the business reasons for using software-engineering techniques and automation in applications development.

Next week begins a series of articles on key industry trends in computer hardware, software, database environment, communications and methodologies that are having a profound impact on the computer industry. ■



John Avakian

Organizations are beginning to recognize the increasing importance of preserving the value of past software investment through re-engineering the existing software base.

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.