Journal of International Information Management

Volume 2 | Issue 2 Article 7

1993

Orion: A menu-driveri source code generator for end users

Jay M. Lightfoot
University of Northern Colorado

Orion: A menu-driven source code generator for end users

Jay M. Lightfoot
University of Northern Colorado

ABSTRACT

Low software development productivity is a major concern for corporations. This paper describes one approach to improve productivity by allowing end users to build their own business data processing systems. The approach uses Orion, a menu-driven software development tool. Orion generates source level COBOL programs that compile and run without modification. The tool does not require end users to know COBOL or high-level query languages. Orion was used for nearly two years to build real business systems. The approach resulted in a significant improvement in software development productivity.

INTRODUCTION

Software development productivity is a key issue for the success of corporations in the future. The challenge of global competition dictates that reliable computer systems be built more rapidly than ever before (Alter, 1991). Unfortunately, traditional software development methods are not up to the task. According to one information systems manager, "the current software development environment [is] like the pre-Ford era of automobiles" (Freedman, 1991, p. 63). Every program is a custom product built by hand. The process is slow and labor intensive, neither of which promote high productivity.

A solution to this problem is to build domain-specific code generation tools that can automate the process (Rich & Waters, 1988; Barstow, 1984, 1985). These tools could deliver the expertise gained from prior development projects to the end user. If sufficiently automated, the tool would not require its users to be familiar with programming. This would allow those who are closest to the problem domain to develop systems to meet their needs. According to Barstow (1984), the best way to develop this type of tool is to build working models and test them on "real users who want real programs that can be run on real data" (p. 26). Orion is such a system.

Orion is a menu-driven programming tool patterned after the code generation approach to end user computing. Orion allows end users to "write" COBOL programs for business oriented systems without having to learn anything about COBOL or programming. These programs run without modification and are standardized for consistency and efficiency. This paper describes the development of Orion and its basic functions. It then discusses the merit of the approach to end user computing.

PHILOSOPHY UNDERLYING ORION

The Orion system is based on the observation that all the programs in a business system can be classified into one of the following four categories.

Inquiry. Inquiry programs read information from a data file and paint that data on the computer screen. The user is normally allowed random and sequential access to the data based on some key field.

Maintenance. Maintenance programs read information from a data file and display it on the computer screen in the same manner as an inquiry program. In addition, these programs allow the user to modify, delete, and add new data to the file.

Report. Report programs read information from a data file and print it on paper in various formats.

Special Processing. All business systems have a few programs that perform domain specific calculations and file operations (e.g., a general ledger posting program and a payroll calculation module). Each program of this type is unique to its domain and must be specially written for its function.

Most business system development consists of writing programs that fall into the first three categories. Based upon the experience gained from developing Orion, 80% to 90% of a business system can be classified as inquiry, maintenance, and report programs. All the programs written within each category tend to be very similar. Consequently, a tremendous amount of productivity can be gained by writing a generic version of each of the "universal programs" and using it as a template for future systems development. The technique of modifying existing programs of similar function is familiar to all experienced software engineers.

Researchers have investigated this general idea under the guise of software reuse (Burton, Aragon, Bailey, Koehler, & Mayes, 1987; Ledbetter & Cox, 1985) software redesign (Fischer & Heinz-Dieter, 1982; Fischer, Lemke, & Rathke, 1987; Neighbors, 1984), and specification reuse (Finkelstein, 1988; Maiden, 1991; Maiden & Sutcliffe, 1992). The concept has received this much attention because the idea of reusing programming components that are known to work is very appealing. Many corporate programming shops are burdened with a backlog measured in man-years; thus, any technique that improves programmer productivity is significant.

The Orion system takes the concept of reusable programs a step further. Orion automates the process of modifying the universal program templates. This allows end users to build their own domain specific applications without the aid of programmers. Users do not need to be familiar with programming nor are they required to learn high-level query languages. The entire process is menu-driven with copious help functions.

The second aspect of the Orion philosophy is that the system automates only those tasks that computers currently do better than people. Orion can generate a 2,000 line domain-specific maintenance program in less than a minute. It cannot automate the programming of tasks that fall into the special processing category. These programs, by their nature, require human intelligence as supplied by experienced programmers. Despite significant progress in the field of automatic programming (Barstow, 1979, 1985; Gomez & Wingate, 1989; Neighbors, 1984), much work must be done before computers can replace human programmers. Likewise, Orion does not attempt automatic program enhancement or maintenance.

Another philosophic concept behind the Orion system is that it produces programs that adhere to a narrow problem domain. The current state-of-the-art for successful code generation systems requires that they be limited to solving specific problems (Barstow, 1979, 1984, 1985; Rich & Waters, 1988). Because of this, Orion is designed specifically to generate the inquiry, maintenance, and report programs characteristic of business data processing.

The final aspect of the Orion design philosophy is that the system is easy to use. Orion was intended to off-load as much programming as possible to non-technical end users. End users will not use tools that are overly complex. Accordingly, a significant amount of effort went into designing a simple, non-threatening user interface.

RESULTS OF USING ORION

Orion was used in an actual business setting for almost two years. During that time it was utilized to computerize several major business systems (viz., purchasing, receiving, budgeting, and personnel). Each of these systems required dozens of programs and well over 100,000 lines of COBOL source code.

The tool was a great success. The prime advantage of using Orion was the incredible productivity it provided. Users were able to generate 3,000 to 5,000 lines of working COBOL code in a single day. Since program generation was fast, users could prototype different ideas quickly and discard bad ones without remorse. Because the code was built using standard universal templates, all programs were standardized and tuned for efficiency. According to one user, the only bad aspect of Orion was that it generated systems faster than the problems could be defined.

OVERIEW OF THE ORION SYSTEM

Orion is composed of a menu shell, utility routines, and programs to generate each of the three supported universal program categories (i.e., inquiry, maintenance, and report). The generator programs have three "template files" that are used to structure the resulting code and control the operation of the generator. The entire system is written in COBOL and does not require any special screen drivers or layered products. The remainder of this section provides an overview of the major components of the system.

Orion Template Files

The Orion template files provide the structure and control information needed by the generator programs. The template files are separate from the generator programs to allow the templates to be updated, using a word processor, without modifying or recompiling the generator. This provides the flexibility to quickly incorporate new standards and efficiency measures into future systems without major maintenance.

Each template file is an ASCII text file made up of static COBOL statements and dynamic control statements. The static statements represent those parts of the universal program that do not change from one business system to the next. For example, the WORKING-STORAGE

SECTION for a purchasing inquiry program is almost identical to the same area of a budgeting inquiry program. Likewise, the error handling routines do not change from one system to another. The static statements represent the combined knowledge acquired from all prior system development projects. In effect, this is a working example of analysis and specification reuse.

The dynamic control statements are used to generate those parts of the programs that change between applications. For example, most of the INPUT-OUTPUT SECTION and the I/O statements must be modified for different problem domains. These statements are delimited by special processing codes that the generator program reads. Each processing code instructs the generator to perform a specific type of symbol substitution, and in some cases, iteration. The result of each control statement is a custom COBOL routine built using the file and field information specific to the application.

Orion Utilities

The Orion system has many utility routines. Most perform simple tasks such as copying programs from one area to another. Others are more important. The screen utility and the report utility are the two routines that have the most significant roles.

The screen utility is a program that reads a screen image text file and writes a table that defines that image to the inquiry and maintenance generator. A screen image is an ASCII file built by the user to depict the computer screen painted by inquiry and maintenance programs. Special symbols in the image represent the data type, data length, and alias of each data field. The screen utility reads the image and asks the end user to identify the data field associated with each screen element. The output of the utility is a table that provides enough information to build a program to produce the image. After program generation, the image file serves as documentation for the resulting program.

The report utility performs essentially the same function as the screen utility—except that it builds report tables. The input to the report utility is a report image file built by the user to represent the layout of the desired report. Special symbols are used to represent data type and data length. The report utility processes the image and asks the user to supply the data names associated with each report field. The output of the utility is a table that the report generator uses to build a corresponding report program. As before, the report image is useful for system documentation.

Inquiry/Maintenance Generator

A single generator program is used to build inquiry and maintenance programs. This is possible because an inquiry program is basically a maintenance program without the modification routines (i.e., inquiry is a subset of maintenance). A separate template file controls the generator for each of the two program types. As was stated before, the program templates control the generator. The maintenance template merely uses those generator routines necessary to build a maintenance program. Likewise, the inquiry template uses a subset of the generator routines. Viewed from this perspective, the template files are a type of high-level specification language.

The inquiry/maintenance (I/M) program generator requires two predefined inputs—a screen image table and a file layout. The screen image table is built using the screen utility. The file layout defines the application master file. Orion file layouts are the same as standard COBOL layouts with the exception of a special header section. The header states the prime and alternate keys (and their aliases) in a format that Orion can use. The file layouts are prepared by the programming staff and are kept in the computer's common data dictionary, so they are available to all users.

Building an I/M program is a three-step process once the predefined inputs are available. First, the user is asked to identify the file layout and the screen image table. Next, the user is asked to supply a name for the new program. Last, the user is asked to supply a comment to describe what the program does. Program generation usually takes about a minute. The user can then instruct Orion to compile and run the program.

Internally, the I/M generator copies the static template statements directly to the new program. The dynamic statements are built via symbol substitution and iteration from information contained in the screen image table, the file layout, and end user queries.

Report Program Generator

The report generator requires an Orion file layout and a report image table as predefined input. A report building session begins by the system asking the user for the names of the file layout and the report image table. The user is then asked to supply the name of the resulting program and a comment describing its function. Following that the user is asked if the data should be sorted and, if so, on what data fields. Finally, the user is asked to list the data fields to use for control breaks (i.e., subtotals). Reports are normally generated in less than 30 seconds and can be compiled and run immediately.

The report generator performs the same type of internal operations as the I/M generator. That is, it copies the static template statements directly to the new program and performs symbol substitution on the control statements. The processing routines in the report generator are different from those in the I/M generator; thus, only the report template can control the report generator.

DISCUSSION OF THE ORION APPROACH

Orion was used to build real business systems in an electronics corporation for nearly two years. The approach was successful because it allowed end users to write their own systems without the aid of programmers. This speeds up software development and frees programmers to work on more technically demanding tasks. The code that Orion generates uses standard COBOL syntax and good program structure; hence, it merges seamlessly with the code written by the programming staff. When modifications are required, end users have the choice of submitting a change request to the programming staff or modifying the image table and regenerating the program themselves. The latter option was normally used in the electronics corporation because the maintenance backlog was always large. This allowed end users to do some of their own system maintenance.

The approach also has some negative aspects. Programmers are still needed to write code for the special processing functions. This ties end user system development to the availability of programming staff. Another problem occurs because of the remarkable productivity that Orion provides. In practice, some end users start building programs before they have adequately defined the function of the system. This leads to the problems commonly associated with poor problem specification.

The net effect of using the Orion approach is very positive. The system improves software development productivity dramatically. It also makes maintenance easier because all the programs it generates are similar. Once programmers understand the structure of the template files, they understand the structure of all the programs that Orion creates.

SUMMARY

Low software development productivity is a major concern for corporations. This paper describes one approach to improve productivity by allowing end users to build their own systems. The approach uses Orion, a menu-driven software development tool based on the code generation approach to end user computing. Orion writes source level COBOL programs to perform the common business data processing functions of inquiry, maintenance, and reporting. Orion does not require end users to know COBOL or a high-level query language. The tool was used in industry for nearly two years and resulted in significantly better software development productivity.

REFERENCES

Alter, A. (1991, March). Increasing the yield. *CIO,* pp. 23-25.

Barstow, D. (1979). An experiment in knowledge-based automatic programming. *Artificial Intelligence, 12*(1), 73-119.

Barstow, D. (1984). A perspective on automatic programming. *AI Magazine, 5*(1), 5-27.

Barstow, D. (1985). Domain-specific automatic programming. *IEEE Transactions on Software Engineering, SE-11,* 1321-1336.

Burton, B. A., Aragon, R. W., Bailey, S. A., Koehler, K.D., & Mayes, L. A. (1987). The reusable software library. *IEEE Software, 4*(3), 25-33.

Finkelstein, A. (1988). Re-use of formatted requirements specifications. *Software Engineering Journal, 3*(3), 186-197.

Fischer, G. & Heinz-Dieter, B. (1992). The nature of design processes and how computer systems can support them. In *Proceedings of the European Conference on Integrated, Interactive Computer Systems (ECICS* (pp. 73-88). Stresa, Italy, Amsterdam: North Holland.

Fischer, G., Lemke, A. C., & Rathke, C. (1987). From design to redesign. In *Proceedings of the 9th International Conference on Software Engineering* (pp. 369-376). Monterey, California.

Freedman, D. (1991, March). Leveraging tools. *CIO*, pp. 62-67.

Gomez, F. & Wingate, V. (1989). Automatic programming for end users: The TOAD system. *IEEE Transactions on Knowledge and Data Engineering, 1,* 398-405.

Ledbetter, L. & Cox, B. (1985, June). Software-ICs. *Byte*, pp. 307-316.

Maiden, N. A. M. (1991). Analogy as a paradigm for specification reuse. *Software Engineering Journal, 3*(1), 3-15.

Maiden, N. A.M. & Sutcliffe, A. G. (1992). Exploiting reusable specification through analogy. *Communications of the ACM, 35*(4), 55-64.

Neighbors, J. M. (1984). The Draco approach to constructing software from reusable components. *IEEE Transactions on Software Engineering, 10,* 564-574.

Rich, C. & Waters, R. C. (1988). Automatic programming: Myths and prospects. *IEEE Computer, 12*(8), 40-51.