

FOURTH-GENERATION LANGUAGES ISSUE PAPER

BACKGROUND

Computers are designed to process, retrieve and store programmed information. Typically, they can only respond to electronic signals that correspond to binary numbers. Programming languages are designed to translate human language into commands that the computer can understand. There are currently five "generations" or levels of languages available [1]. Machine language, or First-Generation language, consists solely of binary numbers. Machine languages are computer dependent and therefore every type of computer has its own machine language. Assembly language, or Second-Generation language, which came into use in the mid-1950s, uses a shorthand notation of letters and numbers to communicate with the computer in place of binary groupings [2]. Assemblers are then utilized to translate the assembly language into machine language. Assembly language is still computer-dependent and therefore results in a unique assembly language for every type of machine. Higher Order language (HOL), or Third-Generation language (3GL), which came into use in the 1960s, consists of statements which more closely resemble the spoken language, and therefore are easier to read and write since they require fewer statements per function. Special programs, or compilers, must then translate the HOL statements to assembly or machine language. Expanding upon this, Fourth-Generation languages (4GLs), or Very High Level Languages (VHLL), also use instructions which resemble the spoken language, but they allow the programmers to define "what" they want the computer to do without necessarily telling the computer "how" to do it. Typically, the compilers, or interpreters, for 4GLs are not as efficient as HOL compilers in using available memory and processing speed. Finally, the highest level of programming, Fifth-Generation languages (5GLs), would involve a computer which responds directly to spoken or written instructions, or English language commands. There exist only a few 5GLs or "natural languages", and they are typically used in artificial intelligence applications.

DISCUSSION

Eighty percent of weapon system and AIS applications are written in 3GLs. [2]. However, 3GLs require a special program, or compiler, to translate the HOL statements into assembly or machine instructions. Compilers are not as efficient in terms of memory utilization or processing speed, so a tradeoff between performance and ease is often required when employing 3GLs. Additionally, 3GLs require a vast amount of code to provide the same functionality and are time consuming to debug. Hence, the necessity for the creation of 4GLs, which capitalize on advanced techniques of programming while simplifying the man-machine interfaces [3]. Currently, there are many 4GLs available, such as Oracle, Visicalc, FOCUS, RAMIS II, and DBase IV, and new ones are still appearing daily. The languages tend to fall into four functional areas: Query and Report Generators, Graphic Languages, Database management tools and Spreadsheets. As such, they typically have a very limited range of application. This issue paper will address the productivity differences, anticipated and experienced, between 3GLs and 4GLs.

Provided below is a detailed discussion of many of the attributes associated with 4GLs [3]:

Ease of use:

Because the syntax of 4GLs closely relates to the human-language syntax, it is easier to learn. Additionally, due to the non-procedural nature of many of the languages, the techniques for accomplishing things are also simple, while the results are fast and impressive. Fourth-Generation languages also aim to remove the use of unnecessary acronyms, thereby allowing users to expend their effort on the purpose of the application, vice being bogged down with extraneous requirements.

Limited range of functions:

4GLs are typically designed for a limited set of functions or specific applications. This is because the product then becomes easier to use than a full programming language. For example, Lotus 1-2-3 gained a large number of users quickly because it made it easy to manipulate spreadsheet data.

Restrictions of Options:

Higher level languages often restrict the options available to users of lower level languages, such as the capability to modify themselves at execution time. To compensate for this, 4GLs permit automatic verification before testing.

Default Options:

A user of a 4GL is not required to specify everything. Instead, a compiler or interpreter is capable of making intelligent assumptions about what it thinks the user needs. Therefore, while 4GLs often require that many parameters be specified, they also provide a default option if the user does not make a selection. This saves time and debugging efforts.

Monologue and Dialogue:

With 4GLs, a dialogue occurs between the user and the computer. This allows for more opportunities to catch errors as they are being made. The software may ask the user questions, signal errors, and flag inconsistencies as the application is being created.

Summary:

The objectives of 4GLs are: 1) to speed up the application-building process, 2) to make applications easy and quick to change, hence reducing maintenance cost, 3) to minimize debugging problems, 4) to be able to generate “bug-free” code from high expressions of requirements and 5) to make the language easy to use, so that the end users could solve their own problems [3]. Fourth-Generation languages require far fewer lines of code than would a 3GL, and they also employ a wide variety of other tools such as screen interaction, filling in forms or panels, and computer aided graphics. The goal is to allow the programmer to tell the

computers “what-to-do” and not have to worry about telling the computers “how-to-do-it” (i.e. non-procedural).

There are currently no standards for 4GLs, primarily because new ideas in language syntax, dialogue and semantics are appearing daily. Some 4GLs have already disappeared, and many of the best languages now come from relatively small, new vendors. As such, their application to Mission Critical Computer Resources (MCCR) systems, where obsolescence and supportability are key issues, is limited. Due to their limited range of application and their slow processing speeds, they will tend to be more prevalently utilized in Management Information Systems (MIS) developments.

QUANTITATIVE IMPACTS

There have been several studies conducted and published in the late 1980s which compare various 4GLs with 3GLs [4]. While interesting to read, the results are not definitive. Provided below is a synopsis of some of these studies and their associated findings.

Study #1: Massey University - Correspondence School Information System [4]

Results:

- 4GL lines of code were approximately 85% fewer than 3GL (13,900 SLOC versus 93,600 SLOC)
- 4GL overall effort experienced a reduction of 77% in total manmonths required (61.6 manmonths (4GL) versus 262.3 manmonths (3GL)), while in contrast overall productivity decreased 37% (18.82 loc/day (3GL) versus 11.87 loc/day (4GL))
- Phase distribution for 4GL was more front loaded than typical 3GL developments (24% for feasibility and requirements on 4GL project vice 6% for 3GL project)

Bottom Line:

85% fewer lines of code, 77% fewer manmonths, however with an associated 37% decrease in productivity (hrs/loc)

Study #2: Matos and Jalics Experimental Analysis [5]

Bottom Line:

29-39% increase in productivity

Study #3: Capers Jones-Applied Software Measurement Surveys [6]

Results:

- 3GLs averaged from 4 function points (FP) per staff month (sm) to 12 FP/sm and 91 source statements per function point which resulted in 364-1092 statements/staff month
- 4GLs averaged from 8-18 FP/sm and 20 source statements per function point which resulted in 160-360 statements/staff month

Bottom Line:

4GL shows average increases in productivity per function point of 100-150%; however, it appears less productive based on statements per staffmonth.

Study #4: Harel and McLean Study - UCLA Graduate School of MIS [7]

Results:

- 3GL to 4GL delivered source instructions (DSI) varied by a factor of 0.9:1 to 27:1
- 3GL to 4GL man-hours (MHRs) varied by a factor of 1.5:1 to 8:1
- 3GL to 4GL productivity (DSI/MH) varied by a factor of 0.5:1 to 5:1

Bottom Line:

On average, 60% fewer lines of code (DSI), 60% fewer man-hours (MH), however, with roughly equivalent productivity (DSI/MH)

The above comparisons highlight the major difficulty involved in comparing 3GLs with 4GLs, which is that, because 4GLs tend to take less lines of code to provide the same functionality, they typically will require overall less effort (man-hours), but achieve no greater productivity (hrs/loc). To further exacerbate the problem, there is no established definition of a line of code in a fourth generation language, and this is complicated by the nonprocedurality of many 4GLs (i.e., the additional lines of code automatically generated that are associated with carrying out an instruction) [4]. In fact, often times to conduct these studies, the participants were forced to invent a definition of a line of code (for example - if the form data went down a screen in columns, each line in a column was counted as a separate statement if the line described a separate object). Additionally, in order to generate the 3GL to 4GL comparison, standard expansion factors were often times utilized (i.e., one 4GL line is equivalent to six 3GL lines - based on standard function points metrics), since the comparable 3GL program is hypothetical. A number of alternative metrics have been suggested to rectify the line of code specification problem, such as [8]:

- “Software science” or program information-content metrics
- Program control-flow complexity metrics
- Design complexity metrics
- Program-external metrics, such as number of inputs, outputs, files, inquiries and interfaces, or function points (a linear combination of those five quantities)
- Work-transaction metrics

However, no formal metric for defining 4GL work content has been developed and universally accepted.

RECOMMENDATION

Anticipated savings from the utilization of 4GLs depend greatly on the languages being utilized, the intended application of the 4GL, the personnel being utilized, their experience with the 4GL, and the metrics utilized to compare the productivities. Based on the varying amounts of savings realized in the studies cited above, and the uncertainty and inaccuracy surrounding this type of comparison, NCCA recommends that the analyst does not try to convert a 3GL productivity metric into a 4GL productivity metric, but rather collect, normalize and utilize historical sizing, effort and productivities for comparable 4GL efforts. In the event that the analyst is unable to collect comparable historical 4GL data, then NCCA recommends utilizing average 3GL productivities (i.e., assume no increase or decrease in productivity) from either the associated contractor or from the NCCA effort tool (if contractor data is unavailable), while verifying that the PM's software engineers have adequately sized the 3GL effort to reflect a reduction in the total anticipated number of delivered source instructions (since 4GLs require less statements per function). This will still result in an overall savings in anticipated man-hours.

Finally, as historical data becomes available, the NCCA SW team will periodically revisit and update this recommendation.

REFERENCES

1. Guidelines for Successful Acquisition and Management of Software Intensive Systems: Weapons Systems, Command and Control Systems, Management Information Systems, Version 1.1, STSC, February 1995, pp. 2-8 & 2-12.
2. Hook, Audrey A., et al., "A Survey of Computer Programming Languages Currently Used in the Department of Defense - An Executive Summary," Crosstalk, October 1995, pp. 4-5.
3. Martin, James and Leben, Joe, Fourth Generation Languages, Volume III: Products, pp. 6-20.
4. Verner, June and Tate, Graham, "Estimating Size and Effort in Fourth-Generation Development," IEEE Transactions on Software Engineering, July 1988, pp. 15-22.
5. Fenton, Norman and Pfleeger, Shari Lawrence, "Science and Substance: A Challenge to Software Engineers," IEEE Transactions on Software Engineering, July 1994, p. 93.
6. Jones, Capers, Applied Software Measurement - Assuring Productivity and Quality, McGraw Hill, Inc., New York, 1991, pp. 55 & 76.
7. Harel, E. and McLean, "The Effects of Using a Nonprocedural Language on Programmer Productivity," UCLA Graduate School of Management, Information Systems Working Paper, pp. 3-83, November 1982.
8. Demarco, Tom and Lister, Timothy, Software State-Of-The-Art: Selected Papers, Dorset House Publishing Co. Inc, New York, New York, 1990, p. 34-35.