Expectations for a Fourth Generation Language

Darius S. Baer, IBM Corporation

Abstract

A fourth generation language (4GL) is identified by its ability to provide front-end processes for the end-user or programmer who needs facilities for data input/output, data management, report presentation, graphics, or statistical analysis. These processes are often compiled procedures and functions that are invoked when needed by the user. The extent to which the services and facilities are supplied by the 4GL differentiate it from third generation or high level languages such as COBOL or FORTRAN. These services and facilities allow the programmer to specify ***what he wants*** the system to do rather than ***how to do it***. We may expect a 4GL to provide improved programming **productivity** (a shorter development cycle), reasonable system **performance** (as measured by batch or interactive response times), increased **function**, much greater **ease-of-use**, **extendibility** (within and outside the 4GL), **flexibility** to handle different applications, malleability to individual **style**, and high **interface capability** to other software and hardware.

Cost benefit analyses may be applied when comparing 4GLs. The benefits are measured in terms of information value to the organization and usability of the language. The costs are measured in terms of time and effort expended for learning and using the software both in development and in application.

A 4GL should provide a language that views the world of data the way the user does rather than requiring the user to change their perspective of the world. The challenge for a 4GL is to give a users what they want as well as what they may need. As long as the 4GL user can specify what the system should do, we may **expect** the 4GL to have built-in procedures to fulfill the request.

Introduction

It is not uncommon for any of us to have suffered the frustration in the use of a language or application program that we may have wanted to destroy the terminal with a sledge hammer. This paper grew out of a desire on my part to understand why I found some software tools to be very productive while others caused me great pain and agony. Shouldn't we, as users, have some input as to the types of languages we use and how they will work? What can we reasonably expect from a computer language to best meet our computing needs?

Users of computer languages often do not think about the expectations they have for that language until it fails to meet one of those expectations. This paper will address the expectations for components of a fourth generation language (4GL) in order to assist end-users, data analysts, programmers, managers, and others in their decisions of which languages to use and/or how best to use an assigned language. As we contemplate six of the major areas in which a 4GL might be used, we may ask to what extent the language can do that work for us. Those areas include:

- Data input
- Data management
- Data analysis
- Data output (including reporting)
- Graphics
- End-user interfacing (including the use of windows, default screens, etc.).

In order to qualify as a true 4GL, a language ought to handle each of these areas, and do so without sacrificing productivity, performance, function, or ease-of-use. Furthermore, we would expect the language to be incrementally easy to learn such that minimal training would provide some gain while further training would increase productivity. The less the user needs to learn about ***syntax*** and the easier it is for him to learn the syntax he needs to know, the more likely it is that the language will be used successfully and productively. Human beings think in an ***associative*** manner. Although machines operate in a single step process, it is possible to construct a language that thinks more along the lines of a human rather than requiring the human to think along the lines of a computer. So, we have

the goal of searching for and expecting 4GLs to fulfill our wants and needs in the DP environment.

Generations of Computer Languages

- 1st - Machine language
- 2nd - Assembler language
- 3rd - COBOL, FORTRAN, etc.
- 4th - SAS, AS, etc.
- 5th - Expert systems, natural language processors, etc.

Figure 1. **Generations of Computer Languages.** A listing of the five generations of computer languages

Computer languages have been evolving gradually as have computer hardware. Figure 1 provides a list briefly describing each of the five generations of computer languages. Each level has different attributes upon which the succeeding generation has been built. Machine language is truly the language of machines, that of bits and bytes. Assembler language contains somewhat more syntax than machine language, but still revolves around the manipulation of registers. Third generation languages provide considerably more latitude and flexibility with both richer syntax and well-developed control structures. The user has been removed from direct manipulation of the bits and bytes. However, the user is still required to define the majority of the structure and flow of the process. Fourth generation languages solve a lot of those problems. The user may request many functions and procedures by specifying a small number of parameters. The system has begun to operate more on human terms. Fifth generation languages, in their extreme, would be expected to operate totally on a human level with parallel processing, natural language processing, associative thinking, and fuzzy logic. The computer would be continuously polling the environment for changes and would be evaluating probabilities as to the best decision to make. In areas of deterministic computation, there would be no probabilities, of course.

Major changes have occurred in both the software and hardware arenas. DP shops seem more accepting of moving into new hardware than they do to using new software languages. There may be a reluctance to use new languages because of the cost of learning those languages and rewriting old applications. The benefits of using new software must be weighed against the cost of conversion no matter how great the advantages are for new development. This paper discusses the expectations of a fourth generation language because we are in the midst of a deluge of 4GLs. We ought to have techniques at our disposal to adequately evaluate these languages and choose the one appropriate for our needs and desires.

Fourth Generation Language Expectations

As long as 4GLs suffer in terms of performance or other areas of expectations, we will continue using third generation languages for production applications where performance is an essential requirement. However, as 4GLs become more capable and hardware provides more opportunity and faster response times, what previously was implemented by the programmer in lower level languages will be provided by procedures in 4GLs. Furthermore, we will also be able to run interpreted code as fast as we now run compiled code. The intent is certainly to change the programmer's tools. We are in the midst of the 4GL cycle, well on our way to the fifth generation.

The following list presents those **4GL expectations** that we ought to assess as we evaluate different fourth generation languages.

- Productivity
- Performance
- Function
- Ease-of-use
- Extendibility
- Flexibility
- Style
- Interface capability to other software and hardware.

High ***productivity*** refers to quick turnaround on application development as well prototyping and one-time data analyses. ***Performance*** demands that the system on which the software runs provide reasonable response time. ***Function*** refers to the variety of tasks that are available to be performed by the language for the user. ***Ease-of-use*** is self-explanatory and is a major criteria in determining whether a 4GL will be used at all. With ***extendibility***, we ask that the language allow us to solve our problems, no matter how complex or unique, through some manipulation of the language.

Flexibility is closely related to extendibility, but emphasizes the ability of the 4GL to deal with a wide variety of applications. ***Style*** provides the availability of doing the same task in different ways, thus allowing for individual differences between the users of the 4GL. ***Interface capability*** can be a critical element in the use of a 4GL in that use of other languages and applications may be a requirement, and may, therefore, determine whether a language can be used at all.

You may notice that in all the above descriptions of the 4GL expectations, all evaluations were subjective. What may be satisfactory response time for one shop or individual may be totally unreasonable for another. Furthermore, these items need to be weighed against each other. There will always be trade-offs that need to be made. None-the-less, there should always be certain expectations that we may reasonably have for 4GLs.

Fourth Generation Language Types

A computer language can be many things to different people. Many users do not realize the impact that they can have on the language they use and/or that different languages that, on the surface, appear to be the same in terms of function, usability, performance, etc., actually operate very differently. Figure 2 lists the four major types of 4GLs with their associated strengths and weaknesses.

Many 4GLs process code through an interpreter and then link to compiled code which is passed parameters which have been extracted through the interpreter. Other types of 4GLs are code generators. These languages expand the code written at a higher level into code that can then be processed or compiled by a lower level language. It is conceivably possible to have a 4GL direct compiler. However the costs associated with developing and maintaining this language could be prohibitively expensive, and the flexibility and ease-of-use might be difficult to attain. Optionally, a precompiler can be used. The programmer would then necessarily or just have the capability to use two languages, albeit in the same program. The precompiled code then would be linked into the compiled code.

TYPE	STRENGTH	WEAKNESS
Compiler	Faster performance	Decreased function, productivity, & ease of use
Interpreter	Increased function, productivity, & ease of use	Slower performance
Code generator	Faster performance and increased productivity	Limited ease of use and function
Precompiler	Faster performance and increased productivity	Limited ease of use and function

Figure 2. **Types of 4GLs.** Major types of 4GLs comparing strengths and weaknesses

The eight expectations that we may have of a 4GL are found to greater and lesser degrees in all types of 4GLs. A 4GL is NOT the same as a menuing system. In a 4GL, a program of sorts must be created. It is true that if the language were smart enough, it could theoretically query the user to such a degree that a program could be created that would then operate independently of the user. This may sound extreme, but code generators are part of the way there. Most 4GLs contain aspects of all the types listed in Figure 2. The assessments placed on the various expectations are, of course, subjective, and may vary according to your experience and perspective.

There is no correct answer for the appropriate mix of language types within a 4GL. Current hardware and software technology are major decision factors as are 4GL developer experience and attitudes. The particular attributes in Figure 2 have been rated in terms of relative productivity, system performance, ease-of-use, and function. The other four areas of 4GL expectations are more difficult to attach to a specific 4GL type. Depending on the needs of a user or programming shop, different 4GLs could be chosen. I recommend that you score your 4GL on the various areas using Figure 2 as an example of how to evaluate a 4GL. Score highly those areas for which the 4GL meets your expectations and give a low score to those areas in which the 4GL fails to meet your expectations. You might also weight the

various areas of expectation according to the values and needs of your particular shop. For example, performance might be of paramount concern in your shop. Therefore a 4GL which was based on a compiler might be favored. As long as you are aware of the choices available and the consequences of your choice, you will have made an informed decision.

Awareness of the types of 4GLs is a basic prerequisite to being able to intelligently choose the language you need. Always remember to set your sights high. You might not always get exactly what you want, but you will probably get what you need.

Human and Computer Perspectives

The philosophy of fourth generation languages demands that we understand the requirements of our tasks. We must be able to conceptually visualize the problem we are trying to solve.

The user may not know what he needs, but he knows what he wants! Look at the tools on your desk: phone, typewriter, pencil sharpener, computer. We need to be able to get results without becoming an expert. It is not a question of being lazy or incompetent. We just want to get our work done without reinventing the wheel each time. **YES**, you should understand what the system does for you. You just shouldn't have to work out the details manually. The language *must* address the desires of an audience. If no user wants to use the software, the tool can be fabulous, but it will remain unused.

An example familiar to many is the use of video cassette recorders (VCRs). VCRs come in all designs. Most functions, however, remain unused by VCR illiterate consumers. Bar code readers for use with VCRs to assist in setting recording times are coming into vogue. Perhaps that will fit the style and function of the VCR using public who need an integrated programming system. In this way, the system will have addressed the needs of the audience.

Two years ago, I presented a paper at SUGI 12 titled, "Preparing and presenting Management Reports using SAS/AF Software."[1] This paper emphasized the need for user friendly interfaces that would require only a ***conceptual*** rather than ***syntactic*** understanding of the application. When the system operates more from a human rather than computer perspective, there is a greater likelihood of use. Each user should be able to use the system in his or her own unique fashion. That is where I see 4GLs today, as extensions of our thought processes.

Both software and hardware user interfaces can be improved. From a hardware perspective, Peter Bishop[2] refers to WIMPs as the acronym for mechanisms used to make the end-user interface more flexible. WIMPs stands for Windows, Icons, Mouse, and Pointers. These mechanisms and how they are used can also be simulated by software in environments lacking WIMPs. There is a dividing line between the way computers and humans process information. Depending on the language used, the dividing line can be close to the way computers work, close to the way people think, or anywhere in between. Figure 3 provides an illustration of the interface between computers and humans. The goal is to move the interface line so that ***computers think like end-users***. This is the goal of 4GLs and 5GLs - to provide a bridge or intelligent user interface between man and machine. The degree to which the user interface is moved toward the human side is the degree to which we have accomplished the goals of the fourth and fifth generation languages.

Interaction Between Humans and Computers

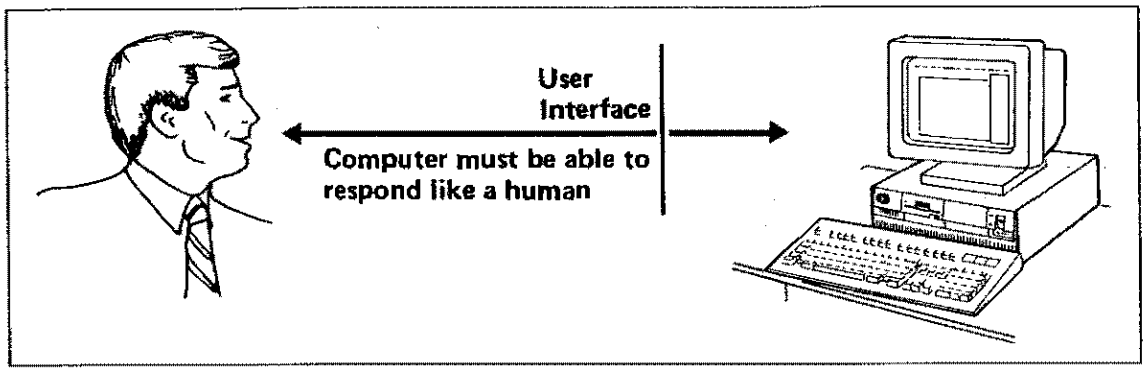

Figure 3. **Human Computer Interaction.** This diagram depicts the point where the interface between computer and human occurs depending on the language.

Many classifications of human workers interface with computers. They include managers, analysts, programmers, data entry personnel, and the vast array of endusers in any business requiring the use of data or information. All of these people can make better use of a system that communicates in a human oriented manner. See Figure 3. As the interface between humans and computers moves closer to the computer, more humans will be able to make good use of computers, and those who already use computers will become more efficient. Computer language is the software answer to improving the human computer interface. Satisfied users will be the result as you can see in Figure 4.

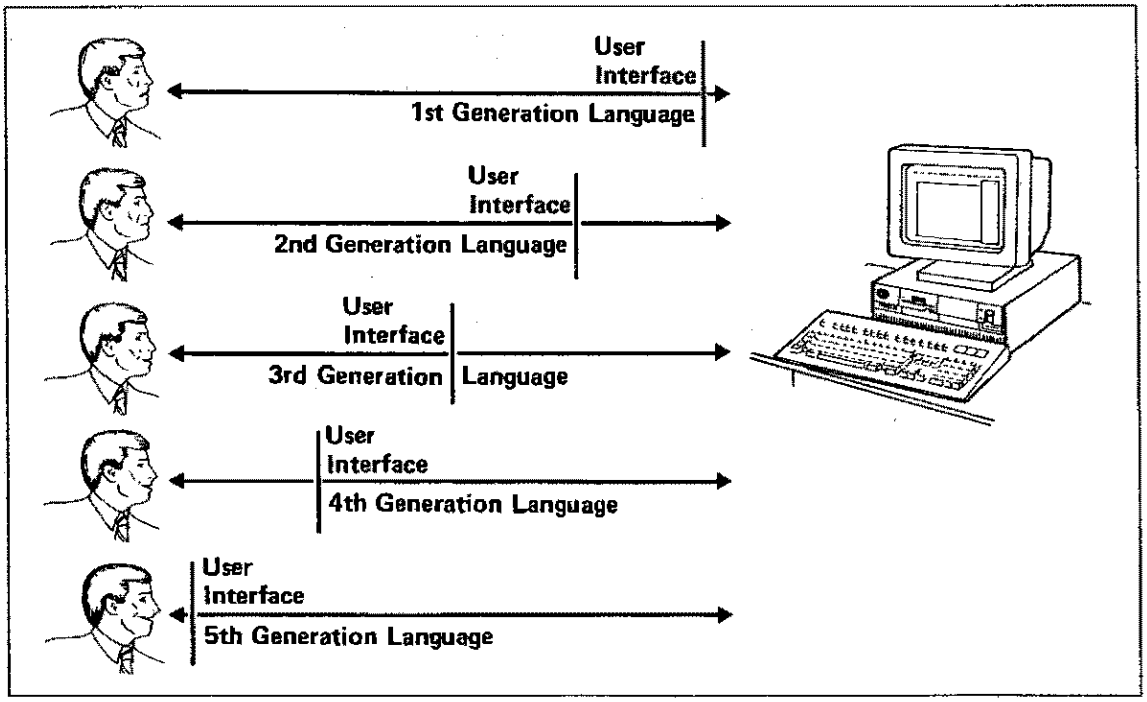

Figure 4. **The Effect of Computer Language on Human-Computer Interaction.** This diagram depicts the relative contribution of computer language to the interface between humans and computers

However, there are a number of problems connected with human-computer interaction (see Figure 5). Computers process information in a **monotonic** fashion while humans jump from one thought to another through ***association***. The ability ***to induce*** general principles from a subset of cases seems uniquely human, but may be simulated by computers using specially designed procedures. There is also the *language problem* where the computer speaks in precise notation while the human speaks in somewhat vague terms, though at a considerably more complex level. The computer tends to have a very structured and **syntactic** view of the world based on accumulation of facts. Humans base their views of the world on facts, also, but do so from an ***conceptual*** perspective. Computers tend to be **inflexible** to different users while humans may behave very differently when discussing the same problem depending on who they are interfacing to. Although humans can arrive at certain decisions ***intuitively***, computers can mimic that by presenting high-level answers and paths without requiring the user to proceed through all the data processing.

HUMAN	INTERFACE BY COMPUTER	COMPUTER
Associative thinking	Group single steps into procedures	Single-step data processing
Inductive reasoning	Simulate induction through procedures	Deductive reasoning
Verbal and textual language	Mimic human language	Numeric and binary
Conceptual	Supply syntax	Syntactic
Dynamic	Appear to be dynamic; pre-process static information	Static
Intuitive decision making	Present high-level final results	Process all data and calculate all results

Figure 5. **Human and Computer Perspectives.** A comparison of human and human perspectives on the data processing issue.

Conventional programming languages try to do two things at the same time according to Michael Jackson[4]. The languages try to let the programmer state what he wants the system to do, while at the same time requiring the programmer to specify how the system should do perform the task and in what order. Fourth generation languages ought to allow the programmer to specify what he wants the system to do while, in the background automatically deciding in what order and how the process should be accomplished. When the system handles the "how to" responsibility, the user then needs only to specify the modules in which he or she is interested. Modularity is a concept which is inherent at all levels of programming.

The question is whether the end-user or the language provides the constructs of that modularity. 4GLs should enforce the concept of structured programming by the use of syntax that strongly discourages the free use of GOTOs. Machine orientation is deeply embedded in our concepts of what a software language should consist. "Conventional programming languages, as commonly used, require the programmer to specify the exact sequence of instruction execution, whether or not s/he wishes to do so."[4] Using such languages burdens the programmer with tasks that can be taken care of by the language as well as contributing to sources of errors and future maintenance. We must be careful not to confuse "the problem itself with the mechanics of executing its solution."[4] Jackson proposes that a high level language be split into a programming side directed toward solving the programmers problems, and an execution side directed toward the problems of interfacing with the other system software and the hardware.

Even the programmer will be moving toward using functional and conceptual techniques rather than twiddling bits. There really is no requirement that in order to control what we want to have done we must twiddle bits. It is only that we have evolved from machine language and continue to do that with which we are most familiar.

By using up-to-date software tools, Dimitris Chorafas[3] proposes that a computer professional can help his company:

1. To keep up with the best technology can offer
2. To benefit from the cost-effective tools currently available
3. To provide the company with a competitive edge over its competitors

"Under no circumstances should the applications programming effort rest on 25-year-old so-called high-level languages."[3] Fourth-generation languages can be classified in five major groups:

1. Programming extensions to the operating system through command interpreters
2. Database management and query
3. New programming languages such as graphics
4. Productivity-oriented tools, through precompilers
5. Spreadsheet systems and integrated software

"Fourth-generation languages are not just end user facilities. They can serve both the end users and the computer specialists — each at a different level of sophistication."[3] They are also constantly evolving. 4 GLs provide procedures to automate application development and assist in the organization of data.

It is important to recognize the distinction between data and information. Data are a group of measurements. Information is the "intelligent," systematic organization of that data that is of use in some form of human endeavor. A true 4GL will help the user manage data and produce reports that contain useful information.

Advantages of Good 4GLs

We can understand better the advantages of a 4GL by comparing the use of a 4GL to a typical High Level Language (HLL) or third generation language (3GL) such as COBOL in a classical application development cycle. With the 3GL, a formal system design with detailed user specifications must be done. With a 4GL, prototyping can be done with a subset of the user's data in order to help the user define their requirements. 3GL application development could require **person months or years** while using a ***4GL*** might reduce that effort to one individual working ***days or weeks***. Although the 4GL may contain procedures, they exist for the programmer to invoke, whereas in the 3GL, the programmer must actually tell the system how a piece of work should be done. In other words, the programmer actually writes the procedures which are provided by the 4GL. In a 4GL, the user passes parameters to the procedures which are then executed. In many cases, a large amount of time is saved because a programming staff is not required, complete with all the requisite and continuous meetings. Rather, one or a very small number of programmers can complete rather large applications. Formal documentation of the code, required in 3GL becomes considerably lessened with the use of a 4GL in which the code, itself, is often the documentation. Additionally, the number of lines of code is often reduced by an order of magnitude, or more, which reduces the amount of documentation needed by the same factor. Along the same lines, maintenance and enhancements are reduced proportionately. Error handling is often improved and the 4GL often assists by default in helping to debug errors.

Chorafas talks about the automobile and the computer as being the 2 most evident impacts upon society. However, as of 1986, over 90% of the American population were auto-literate, while only 5% were regarded as computer-literate. Computer literacy can be accomplished in two very different ways. We can insist that users learn the language of computers or we can insist that computers learn the language of humans. The question is who is to serve whom. If computers are to serve humans, then the hardware and software should be designed to do exactly that. We should, therefore, welcome rather than fight change.

4GLs are powerful system software because they provide, for example, default data management, programming, and reporting functions. This, in turn, results in:

- Decreased Development Time
- Decreased Development Costs
- Increased Software Quality
- Improved Decision Making Capabilities
- Increased Availability of Information

Therefore, development time could conceivably be reduced by two orders of magnitude (100 times) because of the reduction in lines of code and reduction in staff required to produce the code. The more complex the application becomes, the more important becomes structure. The structure can be imposed by the programmers developing many modules or by the language reducing the complexity of the structure to the programmer(s).

When comparing 3rd and 4th generation languages, we must take into consideration development time compared to the value of the completed application. Figure 6 does just that. The formulas that were used to derive the curves for the 3GL and 4GL lines were arbitrarily created in order to fulfill my concept of how 3rd and 4th generation languages provided value per unit time. The equation for the 3GL line is $Y = e^{-(X/25)} \times 54$. The equation for the 4GL line is $Y = e^{-(X/3)} \times 18$. You may notice that for low and

medium application value, the 4GL far outperforms the 3GL. If you were to continue the graph beyond that presented, the 3GL would outperform the 4GL before both curves flattened out at the asymptote lines. These curves are representative of functions in which e is raised to a negative power.

4GLs, by their nature, should be very productive in the early and middle phases of development. Eventually, the user may run into problems with system performance, interface capability, or any of the other items listed as 4GL expectations earlier in this paper. 3GLs, on the other hand, will have a much greater startup cost. If the procedures needed for the application are uniquely designed and built, there could be a higher value obtained by using a 3GL for **very** complex and large applications. As 4GLs evolve, however, 4GLs will be able to be used for larger and more complex applications negating the need for 3GL use.

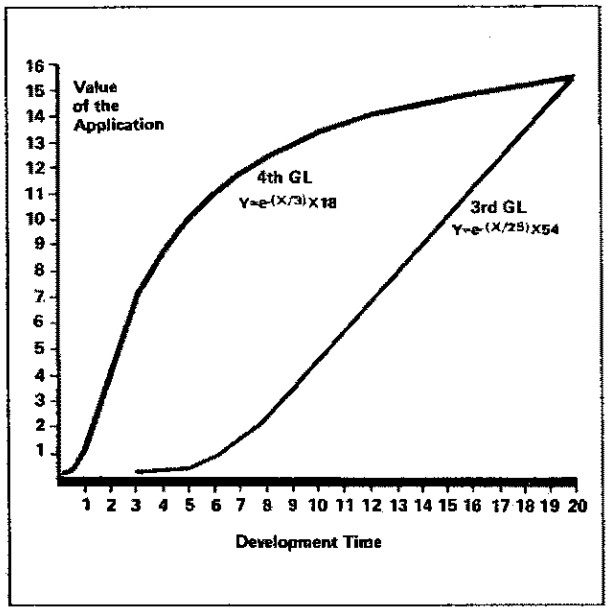

Figure 6. **Value Comparison of 3 & 4GLs.** This compares 3rd and 4th Generation Languages in terms of time required to develop an application depending upon complexity of that application.

Evaluating 4GLs

We need to move from a position where a majority of the work effort in a company is spent in administration and trivial matters to a position where the majority of time is spent in making decisions and obtaining the information to make those decisions. It is considered trivial to be doing work that could either be automated and/or is irrelevant to the current task at hand. 4GLs can bring the end-user to the computing environment.

Solutions for the end-user can come in two forms. First, an easy to learn language can incorporate many of the required processes into **defaults**. Second, flexible **menu systems** can provide a system in which the user does not need to learn any commands, but rather is led through the system by answering questions for which the various answers are readily available. This latter form was described in detail in the paper I presented at SUGI 12[1]. In that paper, I emphasized that by writing an application in SAS AF, the end-user was able to obtain his results because the system knew who he was and presented choices to him rather than asking him to fill in blanks.

Certain requirements must be met by 4GLs being considered for use. They include the following:

1. The system must be **usable**. The user should have access to default values that are easily changeable and syntax that is readily understandable and manageable.
2. The user should have as much **control** as he desires. The user should be able to easily control the flow of the programming process. He should be able to interface with other tools and/or control input, output, data management, and analysis to any extent he desires.
3. Using the system should be **logical** and as self-descriptive as possible.
4. The faster the user can get on board and start using the system, the better. There should be a **short learning curve**.
5. The system must have **tolerance** for different syntax. It must have the ability to detect and correct errors easily.

4GL users should spend time in design strategy, not specifying how the flow should occur. Elements include configuring system, integrating the system among its components of input, data management, analysis, and output. An important element is response time in use of the system. Users should not spend time staring at the tube.

Productivity refers to the information value obtained while working in the software environment. **Usability** refers to the ability of the user to navigate the software environment. These two elements are not necessarily at the same level with all software. Productivity may be high while usability is only average.

Graphs such as those displayed in Figure 7 may be created by using either productivity or usability on the Y-axis versus time and effort expended either learning or using the software system on the X-axis. These four graphs are representative of the various types of 4GLs available or desired by users.

These graphs are created with standard S-curves indicating a start-up period for learning or using, during which, there is little to be gained from productivity or usability, after which there is a steep period of gain followed by a leveling off. Further steep gains may be obtained by learning and/or using advanced features of the software.

The system may have a short learning curve and fast usability or any of the other three possibilities. These evaluations are subjective and individually variable.

The first curve is symptomatic of a language that attempts to be all things to all people all the time. In order to accomplish this, the user may be required to become quite familiar with the language before much productivity can be expected. However, once the language has been mastered the value obtained can be quite high. The second curve is representative of languages that are highly modularized. The user needs to learn only a specific amount of the language in order to accomplish his/her goals. With additional investment of time, however, increased value can be obtained. The third curve is an example for a language that offers a moderate value for minimal learning and using time investment. A program that provides only graphics presentation capability might be one example. Another example might be a spreadsheet program that offers minimal data management, reporting, graphics, etc. The fourth curve is the optimal desired language. It is the system that requires minimal learning and using to provide maximal value. In its ultimate form, it does not exist and rather is a stepping stone to fifth generation languages. These curves are presented in order to stimulate discussion about what is and what could and should be for 4GLs in the market place.

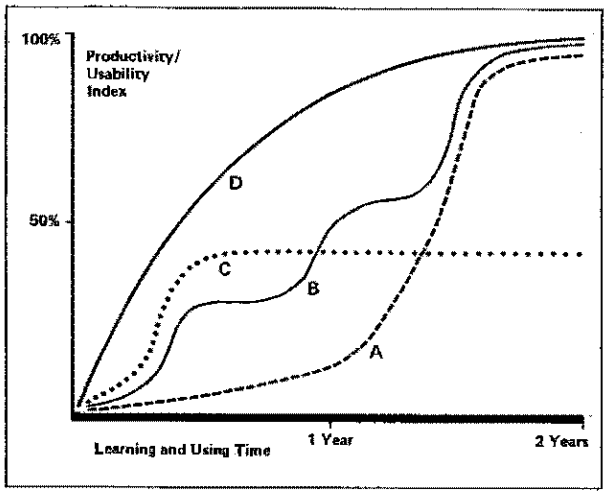

Figure 7. **Cost Benefit Analysis of 4GLs.** This graph compares productivity or usability versus time needed for learning or using the software. The legend follows:

A. Slow learning, high productivity
B. Incremental learning and productivity
C. Fast learning, limited productivity
D. Fast learning, high productivity
(Fantasy -- does not yet exist)

In evaluating software, there must be a limit to the allocation of time to be used for learning and using the language, above which, the software will be rejected.

The following questions ought to be asked when using the graphs and evaluating software.

1. How much of a learning period can be afforded? Large organizations and organizations with a big, continuing workload can afford more extensive learning efforts. Learning efforts are often transferable to new projects.
2. How much of a using period can be afforded? Using efforts are often not transferable to new projects.
3. Is the productivity adequate? How much other software or hardware will be needed to obtain the desired results?
4. Is the usability adequate? Can the desired results be obtained by using the software at all or without going through unnatural contortions?

This subject of expectations for a 4GL is one that needs widespread discussion because the use of software and the accompanying headcount required is the single biggest expenditure in data processing shops.

The 4GL must address the wants of the user **NOT** the perceived needs.

You might ask yourself the following questions:

1. What kinds of unnatural acts do you have to perform in order to get a language to do what you want?
2. Will it do what you want at all? Is the language self-documenting, ie., does it follow a natural human flow of logic?
3. Is the style of use comfortable?

The language may afford you the ability to do a million different tricks, but you need a tool to be productive, not to show off. Bottom line is that the 4GL should do all the up front and leg work for you.

Suppose, in order to build a house, you had to mine ore, smelt it, forge it into metal and make nails. Furthermore, suppose you had to do that in all areas of building. You would never get the house built. You can't do all the work yourself; you need to start at a much higher level.

Suppose, however, that you had all your tools and materials, and you wanted to cut a board for a beam. Your cutting tool, though, was very complicated because it could do all sorts of fancy cuts. You might have to take the time to learn how to use the whole tool before you could cut the board in two. That would be very unproductive, at first.

The above examples signify the need for 4GLs as well as the need for 4GLs that facilitate productivity early in the learning and development cycle.

Conclusions

We need to optimize the process of application programming. Even if we require greater use of computer resources, we will be keeping human resource costs to a minimum. Furthermore, we will be improving software quality because it will be structured by the higher level language in which we will be programming.

The critical need is as stated before: ***User friendly interfaces that require only a conceptual rather than syntactic understanding of the application.*** The system must be human rather than computer oriented. It must uniquely fit the needs of each user. Fifth Generation Languages will provide the interfaces between humans and computers so that the interface is transparent to humans and is truly at the human level. Fourth generation languages should at least make that interface quick and easy. We still need to be specific in our grammar and must interface using computer tools (keyboard, mouse, etc.), but the manner in which the interface is done must be human oriented. That is the responsibility of 4GLs today, to operate as extensions of our thought processes.

Bibliography

1. Baer, Darius (1987). Preparing and Presenting Management Reports using SAS/AF® Software. In SAS® Users Group International Proceedings of the Twelfth Annual Conference, SAS Institute, Inc., Cary, NC.
2. Bishop, Peter (1986). Fifth Generation Computers: Concepts, Implementations and Uses, Ellis Horwood Limited, Chichester, England.
3. Chorafas, Dimitris (1986). Fourth and Fifth Generation Programming Languages, Vol. I: Integrated Software, Database Languages, and Expert Systems, McGraw-Hill, New York.
4. Jackson, Michael (1980). The Design and Use of Conventional Programming Languages. In Human Interaction With Computers, Edited by Smith & Green, Academic Press, New York

Acknowledgement

I would like to acknowledge the assistance of Dale Peterson in the creation of the graphics and the figures.

For more information, contact:

Darius S. Baer, Ph.D.
Department 77K
Support Delivery Systems
ISG Boulder
IBM Corporation
5600 N. 63rd Street
Boulder, CO 80314
(303) 924-2108