4GLs Make Great Strides, But at What Sacrifice?

BY JOSEPH R. SCHOFIELD JR.
Special to GCN

In the rush to fourth- and fifth-generation languages, have we abandoned existing proven technologies in a desperate attempt to alleviate application development backlogs? Furthermore, have we underestimated the speed at which standards committees can accept and implement changes to standardized languages?

Certainly, 4GLs have brought positive effects in prototyping user interfaces, collecting meta data, or data about data, via dictionaries and repositories, and accelerating the software development life cycle.

But before proponents accept too many accolades, further investigation of the sacrifices resulting from use of so-called 4GLs is appropriate in several areas:

- Standardization. How many 4GL vendors identify the level of ANSI standards with which they comply?
- Portability. Less than a handful of vendors support 4GL code that can be ported as is to more than one hardware environment.
- Database options. How many products support DBMS products not marketed by the same vendor? How many 4GL products give the purchasing organization an opportunity to select a DBMS that best suits its needs?
- Training. What are the chances of finding college graduates with practical experience using the chosen 4GL? How often will transferred programming staff need to be retrained for different 4GLs?
- Efficiency. What do 4GLs cost in excessive machine use? Most 4GLs cause the execution of generalized routines, and most still are executed by interpretation rather than compiled syntax.

Recognizing that all technologies deliver less than the unceasingly upgraded requirements demand, 4GLs may prove most useful in an environment with these characteristics:

- Abundant machine resources.
- Unsophisticated developer background.
- Major changes to applications.
- Less than 25 percent annual turnover of application developers.
- Low-volume transaction processing.
- Frequent ad hoc reporting requirements.
- Stable hardware, system software and network.

4GLs have made the software development world take a giant leap backward, partly because of the languages themselves and partly due to misapplication.

COBOL, defamed as it is, remains the language with the most lines of developed code. COBOL is portable, standardized and efficient compared with 4GL execution. It is the host for all major business DBMS products. Proposed changes to COBOL standards will ensure its functionality for many years.

What language has the second-greatest number of lines of code today? An even older language from an earlier generation: assembler.

Fortunately, there is both room and reason for optimism about the emergence of higher-level languages. But users should watch out for vendors who claim to have incorporated 5GL technology in their products already.

The following questions represent an acid test for a 5GL-based product:

- Is the product based on a natural language interface? Does it respond to voice prompts?
- Is the speech pattern natural or deliberate, keyword-driven or system-driven?
- Does the product differentiate between voice sources for security reasons, storing secondary voice patterns for subsequent security auditing?
- Does the product know when it is being addressed, ignoring peripheral or unrelated conversation and noise from a legitimate source?
- Does the product incorporate artificial intelligence? Can it resolve conflicting data and rationalize its conclusions?
- Does the AI component lead to superior intelligence, reducing the likelihood of less than optimal decision-making in the future?
- Can the product be enhanced to conform to potential standardization?
- Is the interface module a shell removed from the user interface, or are all changes apparent to the user?

4GLs made the software development world take a giant leap backward.

There is even more reason for hope in the near future as computer-aided software engineering technology automates the front- and back-ends of the software life cycle. Analysis and design tools that perform verification and validation tasks can ameliorate the misapplication of 4GL technology.

Too often, structured analysis and design methodologies have been undermined by quick and dirty prototypes that slip into production.

Understanding why these applications are termed "dirty" does not always require a postgraduate degree in computer science.

Not all proponents agree on 5GL characteristics and features. But all undoubtedly agree that a 5GL should advance our tool set for the software life cycle.

That advancement may help return systems development from the detour triggered by 4GLs.

Joseph R. Schofield Jr. is a computer systems consultant at Sandia National Laboratories in Albuquerque, N.M.