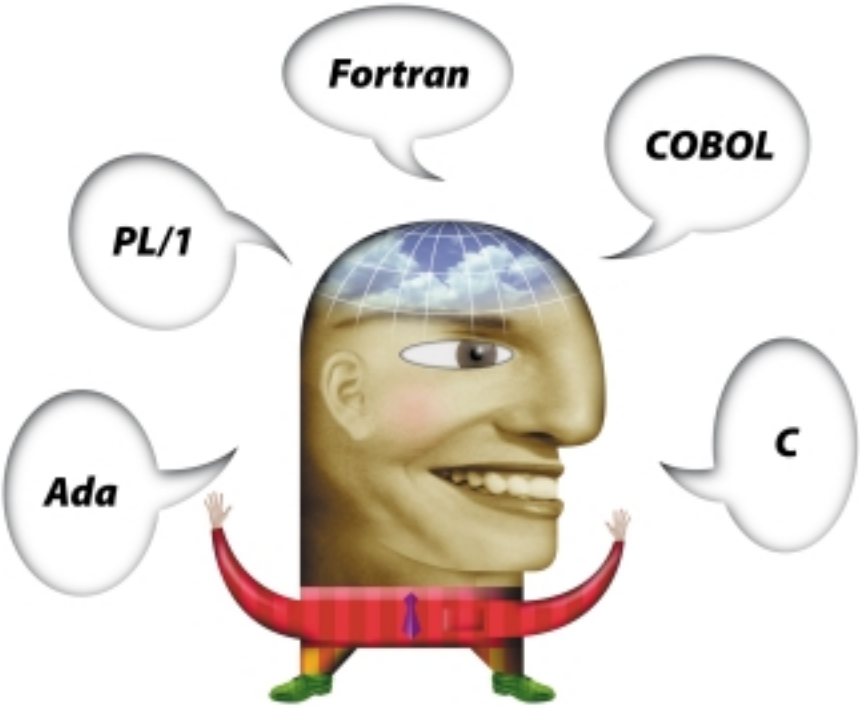

Now that the year 2000 crisis has subsided, **Kevlin Henney** finds how legacy languages will fare in the future, and charts their past development

Mind Your Language

DEBATES on programming languages are always sure to inspire more heat than light. A previous article looked at a number of languages quantitatively as opposed to qualitatively, concluding that once a language has made it into the mainstream, it tends to be quite difficult to kill-off quickly or completely, and that the industry is generally multilingual.[1]

As an industry it would be fair to describe software development as being strongly fashion-led. As with fashion, there is the perception that whatever is at the shows and in the magazines is what everyone is doing. As with fashion this is not the case: people still wear their old clothes and get on with their lives. The call to "wake up and smell the JavaBeans"[2] because Java is the answer, everyone is doing it, and C++ and all other languages are in free fall is about as far from reality as the view that Java is an Internet language for a minority of programmers. From the other corner of the debate, the fact that a language may be better (whatever that means) than another language has never stood in the way of it being unsuccessful in the market place. The elegant Eiffel is a classic example of this; the market is a harsh antidote for idealism. The truth is always a subtler blend of opposite views, and tends to have less regard for fashion, logic or conservatism.

Silent majority

It is easy to fall into a polarised view based on one's work environment. There are many COBOL shops where the idea of GUIs and curly bracket languages seem distant, faddish, and irrelevant; there are vibrant ecommerce companies steeped in XML, Java and Perl to the degree that there is almost denial there has ever been another purpose for software; there are Microsoft-only sites where the idea that other viable operating systems exist, and are measurably more reliable, is at best hearsay; there are places in academia where there is still a strong belief that formal methods are viable contenders for the mainstream; and so on. The message is that software development is more multicultural, in this respect, than many individuals or consultancy groups would like to admit—it is difficult to take a position of extreme moderation and be heard.

For many programming languages the concept of the silent majority applies. There are many languages that are definitely not fashionable but are still quite widely used, away from the glare of publicity. They are used in the context of continued development of existing systems—euphemistically called maintenance by many, an unfortunate and inappropriate term that software borrowed from the physical engineering disciplines—and, perhaps surprisingly, for the development of new systems. With the year 2000 boundary condition out of the way, it is worth taking stock of some of the better known.

Fortran

Fortran started life to prove a point: a compiled language could produce machine code with efficiency close to that of handcrafted assembler. John Backus led his team at IBM in this endeavour between 1954 and 1957. He freely admits that they made the syntax up as they went along, preferring to focus on a close match between the language and the features of the underlying IBM 704 on which it was originally developed. This certainly explains something of the language's syntax and its frequent lack of consistency.

Although it has long since outgrown its original acronym (FORmual TRANslation because, at the time, its syntax had a vague resemblance to mathematical notation) its efficiency and ability to be first to market for the numerate disciplines ensured that it would win out of the more elegant ALGOL 58 and ALGOL 60.

Fortran has evolved radically through a number of standards and extensions. Modern Fortran is radically different to the FORTRAN of the early 1960s (and not simply because Fortran 90 sanctioned the lower case spelling of the name). Programming using exclusively modern Fortran features leads to

differences comparable to the difference between Java and classic C.

It was once said in the scientific programming community that whatever language they were using in the year 2000 it would be called Fortran. It would appear that this prophecy has been fulfilled. Fortran is still taught on science and engineering courses around the world, and is used extensively within these communities. Although Fortran has enjoyed popularity in commercial programming domains, it has been largely displaced leading to the mistaken perception by many developers in the industry that it has completely disappeared. The greatest threats to Fortran's position in its traditional market have come from C, C++ and tools such as Mathematica.

COBOL

When people talk of legacy systems, COBOL is without a doubt the language that most often springs to mind. Along with Fortran it is the language that one can most often expect to find in the history of programmers over the age of 30. It is not simply a matter of history, though. Y2K raised many people's awareness of how much COBOL was out there. Although distinctly unfashionable, COBOL code globally constitutes the most code in any single language. Estimates vary, but in terms of lines of code we are talking international telephone numbers.

What we now know as COBOL started life as an interim report (the final one was never completed) resulting from a user push for a business-oriented language under the auspices of the US Department of Defense and the newly formed Conference on Data Systems Languages (CODASYL) committee. Because of the perceived urgency, the interim report was published as a stopgap measure in 1960 until the 'real' COBOL could be completed. The final report never happened.

Because of the vacuum for suitable business languages, COBOL rapidly took hold of this niche in the 1960s. Once there, it proved difficult to displace regardless of critique of its elegance, design, and other attributes—"sticks and stones...". Only with significant changes in development platform and software architecture has COBOL's dominance been challenged—PCs, client server, GUIs, object-orientation and the Web have all threatened it. However, in these circumstances COBOL has, through proprietary extensions, been extended to cope. Although these have not led to anywhere near the same market presence as the COBOL mainframe association, they are still significant. Through such adaptive extensions and a splintering of mainframe dialects, there are many COBOLs out there, and not simply the ones defined through ANSI standards.

What is COBOL's future as the Y2K teams wind down? Although some companies have chosen to redevelop their systems to a new architecture and to a different business model, there is less incentive to throw it out of the many environments where errant code has been fixed. The language is still taught on many computing courses, and has refused to be displaced by more transient 4GLs, which have themselves caused upgrade problems. While the demand for COBOL may drop a little, it is unlikely to do so drastically.

PL/1

PL/1 (Programming Language 1) was originally touted as the last programming language. As is often the case with such ambitions, it certainly wasn't, finding itself in the company of thousands of other languages during the 1970s inflationary period of language growth.

It was conceived in the mid-1960s as a complete general purpose programming language, equally suited to scientific, business, and systems programming. As such it used the ALGOL, FORTRAN, and COBOL of its times as starting points. A further criticism often levelled at it was that because it was many languages rather than one, it all too easily allowed developers to program in their original language's idioms. It has been said that one can program FORTRAN in any language, and PL/1 certainly enabled this to happen in a more complete sense than most.

PL/1 was developed by IBM. Originally named NPL (New Programming Language) it was further developed at the company's Hursley labs in the UK, where it was felt that the name might cause confusion with the National Physical Laboratory.

SIMULA killer

The only other language at the time with similar ambitions was Algol 68, which became lost in complexity and committees. Usable implementations of Algol 68 only began appearing in the mid-1970s, long after the original definition. It was competing with PL/1, whose implementations were improving after a shaky start in the 1960s. The backing of IBM and an ANSI/ECMA standard in 1976 saw to it that in spite of its significant complexity, PL/1 stayed afloat while Algol 68 persisted only in memory and language design influence. PL/1's complexity led to the formation of subsets, including PL/C and the official 1981 ANSI standard PL/1 subset.

PL/1 was always poised for the big time, to the point that it was used as the second language in the classic *Elements of Programming Style* (the other language in that being FORTRAN 66).[3] Its touted position as the only programming language one would ever need was responsible, in Norway, for preventing major public funding of SIMULA, the first object-oriented language. However, with the possible exception of ALGOL, PL/1 did not displace those languages that it was intended to supersede. It has been used typically for large systems, and never fully broke away from living on mainframes and minis. IBM sites were the most likely to use it, and so its original status as a vendor-created language was also never fully shaken off. Its use has certainly declined in favour of either newer or other old and established languages, such as FORTRAN and COBOL.

C

C is the progenitor of the curly bracket family, which includes C++, Java, scripting languages such as Perl, and interface definition languages such as OMG IDL. It is descended from ALGOL 60, via the BCPL rather than the Pascal lineage. Based on his experience of the Multics project, Ken Thompson at AT&T experimented with a number of languages, including BCPL, for developing the kernel of what was to become Unix. Dennis Ritchie developed C in 1972 as a high-level systems programming language for this purpose, rather than for the general purpose role in which it has since found favour.

The fate of C and Unix were intertwined in the early 1980s, but the uptake of C as a general systems programming language spread it far beyond the Unix environment, displacing assembler and other high-level languages as the standard for systems programming on many systems. The 1990s saw C become accepted as general purpose programming language, and it became a typical sight on developer CVs, either as the language of choice or as a common second language. C is a small language, with only a handful of keywords and concepts. However, some of these concepts and syntax rules are tricky for the newcomer (and even the experienced developer, as witnessed by the International Obfuscated C Coding Contest). This has often made it a difficult language to move to for developers experienced in languages based on a fundamentally different mindset, such as COBOL.

The original de facto standard, the "White Book", came out in 1978.[4] This is commonly known as K&R C or Classic C. ANSI

became involved in standardising the language, leading to the C89 standard, and ISO become involved soon after. A new standard, C99, is about to hit the streets. The emphasis on portability has served to create a very narrow dialect range for the language. Compared to other languages, initial standardisation had little to do in the way of accommodating radical differences between compilers, and focused more on precise exposition. Recent standardisation has focused on extension, mostly in the areas of internationalisation and numeric programming. The former is a natural consequence of the expansion of software markets and the last follows from a sense of competition with Fortran.

Classy C

The most successful divergence from regular C is what was originally known as C with Classes, and later renamed C++. This began in the early 1980s and features in the language influenced the standardisation of C. C++ is largely compatible with C, which helped it gain a foothold in the areas traditionally dominated by C; where C has led, C++ has inevitably followed. For instance, C has made inroads into the embedded systems community, which traditionally shunned higher-level languages, and C++ has followed, leading to subset specifications such as Embedded C++.[5] Although C++, now an ISO standard language, extended C in the directions of safety, object-orientation, and generic programming, it still retains its C heritage. It represents a challenging move to OO for many non-C programmers.

However, C has not been completely displaced by C++ or its more differentiated offspring, Java. Partly because of the mass of existing code already written in C, and partly because it holds a place as the lowest common denominator most powerful language available. The Open Source movement of Linux, Perl, Apache, and others, and the original free software movement, exemplified by GNU, is largely C-based. QA Training's experience of C is that the demand for the language is still high, although the profile of course delegates has evolved during the last decade. Many Visual Basic developers wish to get to grips with C to better understand and interact with the Windows API, and C++ developers wish to find out more about the legacy inherited from C.

In the *The End of History and the Last Programming Language*, Richard Gabriel forwards a theory for what makes a successful programming language.[6] C fulfils all of the criteria; in one form or another it seems to be here to stay as a living language.

Ada

Ada was a language born of committees, the US Department of Defense, and Pascal. It was a product of the 1970s and the enduring belief that a final programming language could still be created. Following a recognition of the rising cost of embedded software development, the DOD commissioned a Higher Order Language Working Group that was to identify requirements for DOD languages and use them to evaluate the current set of languages, which was large, given the pre-Cambrian language explosion of the 1970s. The result of the work was that no existing language was entirely suitable, and work began on developing a new one based on ALGOL 60, Algol 68, Pascal and PL/1. Ada—after Augusta Ada Byron, Lord Byron's daughter and Charles Babbage's assistant, and allegedly the world's first programmer—became a standard in 1983, and a new standard was approved in 1995.

Coming from the Pascal tradition, the syntax of Ada is simple and readable. However, Ada's design tried to please all of the people all of the time. The power of the features crammed into its syntax often cause it to be characterised as a complex language. It is very easy to use a simple subset and be unaware of the more advanced features, but in most programming contexts simple features alone are not enough.

Unfamilar model

Because Ada was created as a general purpose language without having the benefit of evolution to temper its features, some of the concepts introduced in the first round, Ada 83, were awkward to use or insufficient. Ada 95 saw a host of new concepts and features added to the language to meet these perceived shortcomings, some of which represented different language models. For instance, although object-based in its original incarnation, it lacked many of the facilities, such as inheritance or any form of dynamic procedure dispatch, which would qualify it as object-oriented. Ada 95 introduced a more complete object model, significantly making Ada the first internationally standardised language to include OO features. However, the model introduced did not build on the existing task model for concurrency, the newer protected type features for monitor-like types, or the common form for objects used in other statically typed languages. Although powerful, this makes Ada 95's object model an unfamiliar one to the majority of OO developers.

Ada's basis in defence work has ensured that it still has a home there, but it is by no means the exclusive choice for all such software. The overhead associated with many of its features and its runtime support system has held it back from smaller embedded systems use. Its reputation for safety has led to its continued use in larger embedded and safety-critical systems. Ironically, it was Ada code that was used on Ariane 5's ill-fated maiden flight.[7] Although this expensive firework ultimately occurred because of a failure to carry out appropriate tests, there is a further irony that had the system been written in C, the disaster would probably never have happened! The other significant market for Ada has been in education, where it replaced Pascal in a number of universities. However, the time for even that market appears to be passing with many universities favouring Java as a first language.

Conclusion

Although not an exhaustive survey of old languages, this article has not been about finding out whether or not a language is the right fit for you, but how languages evolve and fill niches, and how many programmers navigate their own path between the Scylla and Charybdis of fashion and pragmatism. Certainly, when niches disappear we can expect languages to follow, but although there are always parts of industry undergoing seismic shifts, it seems that there are also parts of it that aren't. ■

Kevlin Henney is a Principal Technologist for QA Training. He can be contacted at **khenney@qatraining.com**

References

1. Henney, K. "Speaking in Tongues", *Application Development Advisor*, 2(1), Sept./Oct. 1998.
2. Guttman, M. Letters response to "Java is an OO Language?", *Application Development Advisor*, 3(1), Sept./Oct. 1999.
3. Kernighan and Plauger. *The Elements of Programming Style*, 2nd edition, McGraw-Hill, 1979.
4. Kernighan and Ritchie. *The C Programming Language*, Prentice Hall, 1978.
5. The Embedded C++ home page, www.caravan.net/ec2plus
6. Gabriel, Richard P. *Patterns in Software: Tales from the Software Community*, Oxford University Press, 1996.
7. The Full Report for Ariane 501, www.esrin.esa.it/htdocs/tidc/Press/Press96/ariane5rep.html