Tecnología

# Algunas lecciones y perspectivas sobre los lenguajes de programación

J. Ángel Velázquez Iturbide[1], Andrés Marín López[2]

*[1] Escuela Superior de Ciencias Experimentales y Tecnología, Universidad Rey Juan Carlos, Socio de ATI; [2] Escuela Politécnica Superior, Universidad Carlos III de Madrid, Socio de ATI*

***<a.velazquez@escet.urjc.es>***
***<amarin@it.uc3m.es>***

***Resumen:*** *hablamos del pasado y el futuro de los lenguajes de programación. Sobre el primero, comentamos algunas lecciones aprendidas de su historia, sobre todo de sus éxitos y fracasos. Sobre el segundo, señalamos algunas tendencias futuras.*

## 1. Introducción

Han transcurrido casi 45 años desde que apareció el primer lenguaje de alto nivel, **FORTRAN** inventado por Backus en 1957. McCarthy diseñó el lenguaje **LISP** en 1958, sentando las bases de los lenguajes funcionales. Algunos otros lenguajes surgieron en esos años, destacando **COBOL** y **Algol**. En todo este tiempo el mundo de los lenguajes de programación ha mostrado una gran vitalidad y han surgido distintos lenguajes: lenguajes orientados a objetos como **SIMULA 67** o ***Smalltalk***; lenguajes lógicos como ***ABSYS 1*** o ***Prolog***; lenguajes funcionales como ***Standard ML*** o ***Haskell***; lenguajes algebraicos como **OBJ** o ***Clean***; lenguajes de bases de datos relacionales como **SQL**; lenguajes concurrentes como **ALEF** u occam; o lenguajes de especificación como **LOTOS** o **SDL**. El lector interesado puede encontrar más detalle en la bibliografía [1, 2, 4, 6, 7]. Nuestro interés aquí es mostrar algunas conclusiones nuestras sobre lo sucedido en este periodo. Por supuesto, asumimos que nuestra visión es parcial, tanto por nuestro punto de vista como por nuestros conocimientos de lo sucedido. Desde aquí animamos al lector interesado a estudiar lo expuesto de forma más general y sistemática.

## 2. Algunas lecciones

La historia nos ha dado algunas lecciones respecto a la evolución y el uso de los lenguajes informáticos. Comentamos algunas lecciones sobre la dicotomía entre lenguajes generalistas y lenguajes específicos para un dominio de aplicación, su nivel de abstracción, la estandarización de los lenguajes y el precio de los errores.

No hay un lenguaje que sea mejor que los demás en todos los aspectos. Si algo nos enseña la historia de los lenguajes de programación es que nunca hay un lenguaje que nos satisface completamente. Bien se busca un lenguaje más abstracto, bien más portable, bien más adecuado a cierto dominio, etc. De hecho, igual que existen algunos antagonismos en la informática, como la eficiencia en espacio y tiempo en la algoritmia, parece haber un antagonismo entre lenguaje generalista y lenguaje para un dominio. Esta simple dicotomía debería hacernos desistir de emprender ninguna cruzada a favor de ningún lenguaje y limitarlas a cada uso concreto.

Por ejemplo en los años sesenta se inició un debate entre los partidarios de FORTRAN y de lenguajes tipo Algol [8]. Los partidarios de Algol, mayoritariamente académicos procedentes de matemáticas, han denostado FORTRAN, preferido por físicos e ingenieros. El resultado de este enfrentamiento es que, hasta FORTRAN 90, FORTRAN no ha incorporado muchos de los elementos de la programación estructurada. Al revés es aún peor: los lenguajes tipo Algol no contienen algunos de los elementos que justifican el éxito de FORTRAN.

Otra cuestión importante es el nivel de abstracción de los lenguajes. Desde los primeros lenguajes de máquina hasta los lenguajes orientados a objetos pasando por lenguajes ensambladores, FORTRAN y lenguajes procedimentales, la tendencia ha sido hacia un mayor nivel de abstracción [5] en los lenguajes de programación. Sin embargo, esta tendencia no es inmutable. Por un lado, ha habido lenguajes que al extenderse han provocado una bajada del nivel de abstracción, por ejemplo C. Pero quizá la anomalía más evidente se ha dado con los lenguajes declarativos (funcionales, lógicos, con restricciones) que, a pesar de su altísimo nivel de abstracción, no se han impuesto. No es fácil determinar la razón de este fracaso; quizá falta de apoyo comercial, entornos de desarrollo, herramientas o librerías de programación, o quizá una exigencia de un esfuerzo de formalización excesivo para las necesidades o la formación del programador medio.

En perspectiva, puede decirse que, a pesar de los augurios lanzados en múltiples ocasiones sobre el final cercano del paradigma imperativo, basado en el modelo de computadora de von Neumann, éste mantiene un predominio aplastante. Hasta hace poco este predominio se ha basado en lenguajes procedimentales y actualmente es de lenguajes orientados a objetos. Lo mismo puede decirse respecto al predominio de los lenguajes secuenciales frente a lenguajes paralelos, concurrentes o distribuidos.

A lo largo de estas décadas, la estandarización de los lenguajes se ha mostrado imprescindible y más aún en un mundo con diferentes plataformas interconectadas. Por supuesto, los problemas para estandarizar son muchos y no hay ninguna receta clara para tratarlos: momento de la

estandarización, conformidad del compilador al estándar, obsolescencia de versiones anteriores, etc. En el mundo de competencia feroz que es la informática no puede olvidarse la dificultad de que las empresas acaten la disciplina de un estándar e incluso respeten el tiempo necesario para elaborar la norma.

Los desarrollos de sistemas son cada vez más complejos y costosos, involucrando cuestiones de distribución y concurrencia. Antes de pasar al desarrollo e implantación de un sistema, es necesario demostrar la corrección del diseño y sus propiedades de viveza y seguridad, para evitar pagar el precio de los errores tardíos. Para ilustrar este tipo de sucesos es recomendable hacer una visita al foro de riesgos en computación de la **ACM** (*ACM Risks-Forum Digest*, *http://catless.ncl.ac.uk/Risks*), donde se recuerdan sucesos como un desbordamiento del contador de programa que le costó a un banco de Nueva York pérdidas valoradas en 32.000 millones de dólares, historias de cortes de suministro de operadoras telefónicas como el de Pac Bell de 1991 que afectó a más de seis millones de teléfonos, o fiascos como los errores de diseño de las operaciones en punto flotante del *Pentium*. Tal vez el precio de errores como los mencionados contribuya en el futuro a introducir lenguajes y herramientas formales, a pesar de las dificultades inherentes a su alto nivel de abstracción y desarrollo académico, razaones comentadas en párrafos anteriores.

## 3. El éxito

Es imposible predecir cuándo tendrá éxito un lenguaje de programación, pero podemos apuntar algunas tendencias del pasado. En primer lugar, el éxito de una innovación no siempre llega en el momento de su divulgación. La oportunidad del momento es una pieza clave del éxito de un lenguaje, independientemente de los méritos que aporte. Por ejemplo, la orientación a objetos no triunfó a comienzos de los 80 con ***Smalltalk***, sino en los 80 con **C++** y ***Java***.

Un lenguaje triunfa si satisface las necesidades de una comunidad, aunque no guste a la comunidad académica. Ya hemos comentado antes la divergencia entre seguidores de FORTRAN y de Algol. Los seguidores de FORTRAN apreciaban su soporte para compilación separada de procedimientos, sus facilidades para cálculo numérico, la posibilidad de entrelazar código en ensamblador y la eficiencia del código generado. Por su parte, los académicos preferían la sencillez conceptual y la elegancia de Algol. Otro ejemplo lenguaje de éxito pero denostado por la comunidad académica es COBOL. Sin embargo, se sigue usando muchísimo para aplicaciones de gestión porque satisface mejor sus necesidades que otros lenguajes. Esto se debe a su capacidad para manejar datos heterogéneos y jerárquicos (con estructura de registro), realizar aritmética de punto fijo, generar informes y manejar grandes volúmenes de datos [3].

El aspecto visual y la utilización de modelos claros también suponen otro factor de éxito en la implantación de un lenguaje. Los lenguajes formales actuales, a pesar de todas sus ventajas y el impulso recibido desde la academia, siguen sin imponerse. Por el contrario, vemos cómo el lenguaje de modelado unificado (**UML**), a pesar de que carece de una semántica formal, se está imponiendo rápidamente en las grandes compañías de software. Además de un adecuado nivel de abstracción y el carácter visual del lenguaje, podemos encontrar otras razones de su éxito en la existencia de un creciente número de herramientas de soporte al diseño.

## 4. El fracaso

El intento de imponer algunos lenguajes de propósito general se ha saldado con fracaso. Aunque estos lenguajes frecuentemente han estado bien diseñados, han fracasado por su complejidad. El caso más clamoroso fue el de PL/I por IBM en los setenta. Debido al dominio comercial de IBM, se usó bastante en esa década, pero en seguida decayó su uso. También puede considerarse un fracaso la imposición de Ada por el Departamento de Defensa de EE.UU. En este caso, el lenguaje tuvo un diseño especialmente cuidado, pero su gran complejidad lo limitó casi exclusivamente a aplicaciones militares o de control. Un último ejemplo, esta vez en el ámbito de los lenguajes de descripción de hardware, es **VHDL**, también impulsado por el Departamento de Defensa de los Estados Unidos. A pesar de exigirse una descripción en VHDL como requisito para la presentación de cualquier proyecto, no ha desplazado al resto de los lenguajes, especialmente a Verilog en temas de síntesis.

Algunas aportaciones de alto nivel técnico o matemático exigen un esfuerzo tal a los programadores que estos las rechazan. Este esfuerzo puede darse tanto en los propios lenguajes como en metodologías o en herramientas. Muchas herramientas que restringen el proceso de desarrollo (como los editores dirigidos por sintaxis o las herramientas **CASE**) terminan siendo abandonadas, independientemente de su calidad, salvo por algunos seguidores entusiastas.

## 5. Algunas perspectivas futuras

Resulta difícil hacer previsiones a largo plazo, y más aún en el mundo tan cambiante de la informática. De todas formas, no queremos dejar esta breve reflexión sin señalar algunas tendencias actuales. De vez en cuando se produce un cambio de paradigma en el modelo de programación más usado (imperativo, eso sí). A comienzos de los setenta se impuso la programación estructurada, en los noventa ha sido la programación orientada a objetos (por ejemplo C++, *Java*) y los lenguajes de guión (por ejemplo **Tcl/Tk**, **Perl**). Parece que en esta década seguirá el auge y afianzamiento de los lenguajes orientados a objetos y de guión.

Con la explosión de la *web*, el auge le ha llegado a los lenguajes orientados a la *web*, tanto de programación (p.ej. *Java*) como documentales (de marcado, como **HTML** o **XML**). Java continuará gozando de buena salud durante bastante tiempo y volverá a pasar por nuevas fases de diseño para dotar al lenguaje y a la máquina virtual de nuevas medidas de seguridad especialmente en la parte de soporte a hilos para el desarrollo de código móvil. XML y todos los estándares relacionados (**DOM**, **XSL**, etc.) van a seguir

imponiéndose como base de todo tipo de nuevas aplicaciones en Internet.

Respecto a las nuevas tendencias de software de componentes y servidores de aplicaciones, asistiremos al desarrollo de nuevos lenguajes para la integración de componentes y lenguajes, y posiblemente también para la estandarización de lenguajes de catálogos y descripción de componentes. Por otra parte, la comercialización y distribución de componentes forzosamente pasa por el cauce de Internet y deberá ajustarse a los estándares que rigen en ella. ¿Está XML llamado a desempeñar esta función? El tiempo nos ayudará a desvelar esta incógnita, pero nosotros así lo suponemos.

Respecto a las técnicas formales, únicamente aquellas que ofrecen modelos sencillos y visuales parecen tener éxito en la industria. Si a esto unimos la necesidad de verificación y validación de los sistemas, nuestra opinión es que asistiremos a la aparición de nuevos lenguajes y herramientas visuales de comprobación de modelos simbólica, deductivos y transformacionales, y que estos se integrarán en entornos y lenguajes aceptados, como UML o incluso en *Java*. También es probable que haya un renacer de los lenguajes declarativos, apoyados en mejores herramientas, incluido soporte para programación en Internet.

## 6. Referencias

**[1] D. Appleby y J. J. VandeKopple**, *Lenguajes de programación: paradigma y práctica*, McGraw-Hill Interamericana, 1998.
**[2] T. J. Bergin** (coord.), «History of Programming Languages», *ACM Press*, 1996.
**[3] R. L. Glass**, «Cobol - A contradiction and an enigma», *Communications of the ACM*, 40(9):11-13, septiembre 1997.
**[4] T.** W**. Pratt y M. V. Zelkowitz**, *Lenguajes de programación: diseño e implementación*, Prentice-Hall Hispanoamericana, 3ª ed., 1998.
**[5] M. Shaw**, «The impact of abstraction concerns on modern programming languages», en *Studies in Ada Style*, P. Hibbard, A. Hisgen y otros, Springer-Verlag, 2ª ed., 1983
**[6] R. Sethi**, *Lenguajes de programación: conceptos y constructores*, Addison-*w*esley Iberoamericana, 1992.
**[7] R.** W**exelblat** (coord.), *History of Programming Languages*, Academic Press, 1981.
**[8] M. V. Wilkes**, «A revisionist account of early language development», *Computer*, 31(4):22-25, abril 1998.