Discurso de investidura como doctor "honoris causa" del Excmo. Sir Charles Antony Richard Hoare

10 de mayo de 2013

Acepto con inmenso placer la alta distinción que ustedes me otorgan con este Doctorado Honoris Causa por la Universidad Complutense de Madrid. Me gustaría expresar mi agradecimiento de todo corazón a todos los que hoy están aquí presentes como testigos de esta espléndida ceremonia.

También agradezco al orador que me ha precedido, el Profesor Ricardo Peña, su elogioso relato de los éxitos que he alcanzado en mi carrera de cincuenta años como Científico de la Computación. Su Laudatio comenzaba con mi primer y más afortunado logro, la invención del algoritmo de ordenación Quicksort. En 1960 yo era un graduado visitante en la Universidad Estatal de Moscú. Me atrajo la idea de inventar un algoritmo de ordenación. Mi primer resultado fue un algoritmo conocido actualmente como el método de la burbuja. Pero enseguida lo deseché porque era demasiado lento, ya que necesitaba un tiempo proporcional al cuadrado del tamaño de la lista a ordenar. La siguiente idea se me ocurrió inmediatamente después de rechazar la primera, y fue ya el algoritmo Quicksort. Creo que tuve muchísima suerte. ¿Qué mejor comienzo puede haber para una carrera académica, teniendo en cuenta especialmente que nunca me doctoré?

Raramente tuve tanta suerte después. Prácticamente todas mis ideas afortunadas posteriores relativas a la investigación estuvieron precedidas por una larga sucesión de ideas desafortunadas, a las que tuve que prestar considerable dedicación antes de rechazarlas. A menudo, la versión publicada de

mis artículos estuvo precedida por tres versiones preliminares, y en alguna ocasión hasta por siete versiones. Cada una era una reformulación completa de la anterior. Una constante de mi investigación ha sido que todas mis buenas ideas han sido el resultado de un laborioso proceso de investigar y rechazar mis más numerosas malas ideas. Por tanto, en mi respuesta a la Laudatio que acaban de escuchar, me gustaría equilibrar un poco las cosas. Me gustaría relatarles la historia de algunas de las equivocaciones que he cometido en mi larga carrera. Pero cada relato tendrá una enseñanza positiva: que el error fue a menudo el estímulo directo para un acierto posterior.

Mi primera y más conocida equivocación la cometí a mediados de los sesenta cuando trabajaba para la División de Cómputo Automático de la empresa Elliot Computers, el cual fue mi primer trabajo. Yo había sido ascendido al cargo de Ingeniero Jefe de un proyecto que había de implementar un sistema operativo para los modelos de la familia de computadores vendidos por la Compañía. Tras más de treinta personas-año invertidas en su implementación, el proyecto no estuvo disponible en la fecha prometida, y no una sola vez, sino varias. ¡Cuando estuvo disponible, no pudo ser comercializado en absoluto! Por culpa del sistema de paginación de memoria, resultó ser ridículamente lento. Había desperdiciado dos años de esfuerzo laboral de cada uno de nosotros. Afortunadamente, mi Compañía me perdonó, e incluso confiaron de nuevo en mí, encargándome la tarea de enmendar el fracaso. He relatado en detalle esta historia quince años después, en 1980, en mi discurso de aceptación del Premio Turing. Este discurso se ha publicado y reeditado numerosas veces bajo el título "Los viejos trajes del Emperador".

Personalmente, aprendí muchísimo de este fracaso. Supuso el impulso inicial para mí continuado interés posterior por el fenómeno de la concurrencia en la programación: ¿Cómo se puede organizar la ejecución simultánea de partes distintas de un mismo programa, tanto en un solo computador como en un sistema distribuido? Mi investigación en concurrencia me condujo al desarrollo de la teoría de Procesos Secuenciales Comunicantes, y a su aplicación al diseño de la arquitectura de los transputers por parte de la Compañía británica de microcomputadores INMOS. Desde entonces, y hasta el día de hoy, he continuado desarrollando y generalizando la teoría sobre la concurrencia.

La siguiente equivocación fue aún más cara. Yo le llamo "mi error de 1000 millones de dólares" Al comienzo de los años sesenta me tropecé con el lenguaje de programación Simula, brillantemente diseñado e implementado en Oslo por

Kristen Nygaard y Ole-Johan Dahl. Más adelante, ambos recibieron el Premio Turing por su trabajo. Simula incorporaba una versión primitiva de lo que actualmente se conoce como programación orientada a objetos. Mi contribución al lenguaje fue el descubrimiento de que todo puntero o referencia a un objeto podían y debían ser tipados, según el tipo del valor al cual apuntaban. Como resultado, una simple comprobación de tipos, que podía ser llevada a cabo completamente antes de la ejecución, podía garantizar la consistencia de los tipos de cada uso de una referencia, y por tanto garantizar la seguridad y la corrección estructural de cualquier programa orientado a objetos. Esta solución se adoptó por primera vez en la definición del lenguaje Algol W, diseñado e implementado por Niklaus Wirth. Este lenguaje no obtuvo la aprobación del Comité Internacional de ALGOL, que lo había encargado originalmente. Sin embargo, la idea de tipar y comprobar el tipo de las referencias se adoptó en la versión de Simula de 1967, ampliamente admirada, y en lenguajes más recientes como C++ y Java.

La idea de tipar adecuadamente las referencias fue realmente mi idea más temprana acerca de cómo tener objetos dinámicos en un lenguaje de programación de alto nivel. Desgraciadamente, lo estropeé todo al decidir mantener en mi diseño una característica por todos conocida: el llamado puntero nulo. El puntero nulo es muy útil para representar la ausencia de información, por ejemplo información acerca de la mujer de un hombre que no está casado. Pero también es altamente peligroso. Es el causante de innumerables errores de programación, que conducen en ejecución a resultados impredecibles, y que hacen dichos errores difíciles de detectar y corregir.

Muchos de esos errores todavía permanecen latentes en el software suministrado a los clientes, y son ellos quienes han de sufrir sus impredecibles e incontrolables consecuencias. Algunos de los errores pueden exponer a los computadores a ataques maliciosos o fraudulentos, del tipo de los que hoy suponen a la economía mundial muchos miles de millones de dólares al año. Esta estimación, no solo incluye el perjuicio directo, sino también el coste de adoptar las decisiones necesarias para disminuir el riesgo. Considerando el tiempo de cincuenta años transcurrido desde que cometí el error, estoy seguro de que el coste total excede los mil millones de dólares. He escuchado estimaciones que hablan incluso de mil millones por año.

Al final de los años 1980, trabajé con Robin Milner en Edimburgo y Jan Bergstra en Holanda en un proyecto denominado CONCUR, financiado por la

Comunidad Europea mediante una Acción de Investigación Básica. El objetivo declarado era unificar las tres diferentes teorías sobre la concurrencia promovidas por los tres investigadores principales del proyecto. Mi contribución era CSP, la de Milner, CCS, su Cálculo de Sistemas Comunicantes, y la de Bergstra, ACP, su Álgebra para la Programación Concurrente.

Me temo que debo añadir el proyecto CONCUR a la lista de mis fracasos. Los tres investigadores principales hicimos notables progresos en el desarrollo y aplicación de nuestras propias teorías, pero nunca CONCURrimos (es decir, nos pusimos de acuerdo) en una sola teoría sobre la concurrencia. Este error es un asunto serio porque ningún ingeniero razonable, o jefe empresarial responsable, van a adoptar una teoría que es todavía objeto de disputa entre los principales expertos del área.

Una vez más, el error inspiró la dirección que había de seguir mi investigación posterior. Dirigí mi atención a técnicas matemáticas y lógicas que habían de conducirme a una unificación con mayor éxito de un abanico aún más amplio de teorías de la programación, que incluían los familiares programas secuenciales, así como los programas que se ejecutan concurrentemente. Ya se ha explicado en mi Laudatio la importancia de la unificación en el progreso de la ciencia; pero esta importancia todavía no ha sido reconocida por los científicos informáticos.

Como resultado, y junto con mi colega y amigo de muchos años He Jifeng, nos embarcamos en un programa de investigación, que finalmente se extendió más de diez años, mucho más de lo que nunca hubiéramos imaginado. Ello nos condujo a la publicación de un libro en 1998, titulado "Teorías Unificadoras de la Programación". Por desgracia, el libro no consiguió atraer la atención de los investigadores de nuestra área y la editorial tampoco puso mucho empeño en venderlo. Actualmente está disponible libremente en la Web.

Este fue el último fracaso de mi carrera académica, ya que poco después alcancé mi edad de jubilación en la Universidad de Oxford. Afortunadamente, Roger Needham me invitó a incorporarme a la plantilla de un nuevo laboratorio de investigación de Microsoft, inaugurado hacía poco tiempo en Cambridge, Inglaterra.

Desde mi incorporación a Microsoft, he sido excepcionalmente afortunado como testigo de primera fila del uso generalizado de asertos, y de otros métodos formales, en la práctica diaria del desarrollo del software. Este

uso ha sido potenciado por el empleo de herramientas de Ingeniería del Software tales como PREFIX, que es capaz de detectar muchas clases de errores que se sabe ocurren en los programas, incluido el uso erróneo de referencias nulas que he descrito más arriba. Herramientas más recientes pueden proporcionar de modo automático asertos ausentes en el texto, y utilizarlos para generar casos de prueba que pueden revelar errores en el código recién escrito. Actualmente hay cientos de ingenieros y científicos expertos en verificación en el conjunto de Microsoft. Ellos han desarrollado y hecho evolucionar un amplio abanico de herramientas de análisis, verificación y pruebas, y han colaborado en su implantación para su uso rutinario por los programadores.

En mis últimos cinco años en Microsoft, mi atención ha vuelto a las teorías unificadoras, el tema del último fracaso de mi carrera académica. Pero ahora estoy adoptando un enfoque completamente diferente. He formulado un par de docenas de Leyes Algebraicas de la Programación. Estas leyes son muy parecidas a las leyes de la aritmética que se enseñan a los niños en la escuela. Podrían enseñarse en los primeros cursos de cualquier grado en Informática. Se podría ilustrar su uso realizando ejercicios de optimización de programas mediante transformaciones que preservan la corrección. Creo que servirían como una excelente introducción a los métodos formales para todos los informáticos.

Resulta curioso que mi primera publicación sobre las Leyes de la Programación sea de 1987, antes de que comenzara el proyecto CONCUR, y antes del comienzo de la investigación que condujo a la publicación del libro sobre Teorías Unificadoras. Ese artículo contenía ya todas las leyes relevantes para los lenguajes de programación secuenciales. Pero, de algún modo, a lo largo de los veinte años siguientes, nunca me percaté de que este artículo ya mostraba el modo más sencillo de unificar las teorías de la programación. Ha sido solo en estos últimos cinco años cuando he encontrado el modo de introducir leyes para la concurrencia en un lenguaje de programación. Como resultado, las leyes se aplican indistintamente a los programas secuenciales y concurrentes.

Usando estas leyes, se puede demostrar la corrección no solo de la lógica de Hoare, sino también la de su extensión a la lógica de separación de Peter O'Hearn. Las mismas leyes pueden también probar la corrección de la semántica operacional que Robin Milner utilizó para definir su álgebra de procesos CCS.

Resumiendo, creo que las Leyes de la Programación finalmente han alcanzado los objetivos de aquel proyecto CONCUR que fracasó hace veinte años.

Estaría muy satisfecho si este resultara ser el último logro de mi carrera científica. Es un resultado que posee las propiedades de simplicidad y elegancia que suelen caracterizar a las teorías científicas más convincentes. Para muchas personas, un enfoque algebraico de la programación será seguramente sorprendente. Pero, para mí, la principal sorpresa ha sido comprobar cuánto tiempo me ha llevado alcanzar la claridad de ideas suficiente para darme cuenta de que ya sabía resolver el problema de la unificación. La enseñanza que se desprende de este relato es más bien la contraria de las enseñanzas de los anteriores. He empleado gran parte de este discurso en relatar lo mucho que he aprendido de mis errores. Pero en este último relato sobre las teorías unificadoras les he mostrado lo mucho que he errado en aprender de uno de mis éxitos.

Para terminar, me referiré de nuevo a la Laudatio. Me ha complacido en extremo escuchar que muchos de mis descubrimientos en Informática los han incorporado ustedes a los planes de estudio de la Universidad Complutense. Deseo que sus graduados los encuentren interesantes en un primer momento y útiles más adelante. Pero por favor no entiendan que lo que hoy sabemos y enseñamos en Informática es el final de la historia. Se están abriendo muchas nuevas áreas en nuestra disciplina, hay muchas nuevas certezas por revelar, y vendrán nuevas aplicaciones. Me haría muy feliz que sus nuevas investigaciones estuvieran basadas en mis éxitos anteriores, pero me haría igualmente feliz que estuvieran basadas en mis fracasos.