Discurso de investidura como doctor "honoris causa" del
Excmo. Sir Charles Antony Richard Hoare
10 de mayo de 2013

It is with immense pleasure that I accept the high distinction which you confer on me by the Award of an Honorary Doctorate of the Universidad Complutense of Madrid. I would like to express my heartfelt gratitude to the University, and to all of you who join me here today as witnesses of this magnificent ceremony.

I also thank the previous speaker, Prof. Ricardo Peña for his flattering story of the successful accomplishments of my fifty-year career as a Computer Scientist. His laudation started with my first and most lucky achievement, namely the invention of the sorting algorithm Quicksort. In 1960 I was a visiting graduate scholar at Moscow State University. I became interested in sorting by computer. My first idea for sorting was an algorithm now known as bubblesort. But I rapidly rejected this idea because it was too slow, taking time proportional to the square of the size of the list to be sorted. My next idea came immediately after rejecting the first one. It was quicksort. I think I was extremely lucky. What better start could there be for an academic career, especially since I never obtained a doctoral degree?

I was rarely so lucky again. Nearly all my subsequently successful research ideas were preceded by a long series of bad ideas, to which I gave long and laborious consideration and then rejected. Often the published version of a paper was preceded by three or occasionally even seven earlier versions. Each version was a complete rewrite of the previous version. It has been a characteristic of my scientific research that all the good ideas are just a by-product of the laborious process of investigating and rejecting my more

numerous bad ideas. So in my response to the laudation that you have just heard, I would like to create a balanced picture. I would like to tell you the stories of some of the mistakes I have made in my long career. But each story will have a moral: that failure was often the direct stimulus for a subsequent success,

My first and best known failure was in the mid nineteen-sixties, when I was working in my first job with Elliott Automation Computing Division. I had been promoted to the position of the Chief Engineer of a project to implement an operating system for the current range of computers marketed by the Company. After more than thirty man-years of implementation effort, the project failed to meet its promised delivery date (not just once but several times). In the end, it failed to be deliverable at all! As a result of memory thrashing, it was just ridiculously slow. I had wasted two years of the working life of everyone engaged in the project. Fortunately my Company forgave me, and even entrusted me with the task of recovering from the failure. I gave a full account of it some fifteen years later, in my acceptance speech for the Turing Award in 1980. This speech has been published and republished many times under the title of 'The Emperor's Old Clothes'.

Personally, I learnt a lot from the failure. It was the impetus for my life-long interest in the phenomenon of concurrency in computer programming: how cans one plan for simultaneous execution of different parts of the same program, either on the same computer or on a distributed system? My research into concurrency led to the development of the theory of Communicating Sequential Processes, and its application by the British Microcomputer manufacturer INMOS in the design of the architecture of their transputer. I have continued since then to develop and generalise the theory of concurrency, right up to the present day.

The next of my failures was even more expensive. I call it my billion-dollar mistake. In the early 1960's, I encountered the programming language Simula, brilliantly designed and implemented in Oslo by Kristen Nygaard and Ole-Johan Dahl. They later jointly won the Turing Award for their work. Simula included one of the very first versions of what is now known as object-oriented programming.

My contribution to the language was my discovery that every pointer or reference to an object can and should be typed according to the type of the value which it points to. As a result, simple type-checking, conducted wholly before a program starts execution, could guarantee the type-consistency of

every use of the reference, and so guarantee the security and structural soundness of any object-oriented program. This solution was first adopted in the design of the ALGOL W programming language, which was designed and implemented by Niklaus Wirth. The language itself failed to get approval from the international ALGOL committee which originally commissioned it. However, the idea of type-checked references was also adopted in the widely admired 1967 version of Simula, and the more recent languages C++ and Java.

The idea of properly typed references was actually my very first idea about making object orientation available in a high-level programming language. Unfortunately I then went on to spoil everything by retention in my design of a well-known feature, known as the null pointer. The null-pointer is very useful to represent the absence of information, for example, information about the wife of a man who is not married. But it is also notoriously dangerous. It is the cause of innumerable programming errors, leading to unpredictable consequences, which make them difficult to detect and correct.

Many of these errors remain in software that is delivered to customers, and it is they who have to suffer the unpredictable and uncontrollable consequences. Some of the errors can expose a computer to malicious or fraudulent attack, of the kind that still costs the world economy many billions of dollars per year. This estimate includes not only the direct damage, but the cost of all the precautions necessary to reduce the risk. Over the period of fifty years since the mistake was made, I am sure that the total cost has exceeded a billion dollars. I have even heard estimates of a billion dollars per year.

In the later 1980's, I worked with Robin Milner in Edinburgh and Jan Bergstra of the Netherlands on a project called CONCUR, funded by the European Community as a Basic Research Action. Its stated goal was to unify three different theories of concurrency, as propounded by the three principal investigators. I contributed CSP, Milner contributed CCS, his Calculus of Communicating Systems, and Bergstra contributed his ACP, an Algebra for Concurrent Programming.

I'm afraid I must add the CONCUR project to my list of failures. The three principal investigators made great progress in the advancement and application of their own theories, but we could never bring ourselves to concur (that is, to agree) on a single unifying theory of concurrency. This is a serious matter, because no sensible engineer or industrial manager is going to adopt a theory that is still disputed by the leading experts in the field.

Again, the failure inspired the direction of my subsequent research. I turned my attention to mathematical and logical techniques that would lead to a more successful unification of a still wider range of theories of programming, including familiar sequential programs as well as programs that called for concurrent execution. The significance of unification in the progress of Science has already been explained in my laudation; but it has not yet been widely recognised by Computer Scientists.

So together with my long-term research colleague and friend He Jifeng, we embarked on a programme of research which in the end lasted more than ten years, much longer than we ever expected. It led to publication in 1998 of a book entitled 'Unifying Theories of Programming'. Unfortunately, the book failed to attract the attention of Computer Scientists, and the publisher never exerted much effort to sell it. It is now (freely) available, but only on the web.

This was the last failure of my academic career, since shortly afterwards I reached retirement age at Oxford University. Fortunately, I was invited by Roger Needham to join the staff of a new Microsoft Research Laboratory, recently established in Cambridge, England.

Since I joined Microsoft, I have been exceptionally fortunate in witnessing at first hand an extraordinary spread of the use of program assertions and other formal methods in the daily practice of software development. This was encouraged by the use of Software Engineering tools like PREFIX, which was able to detect many of the kinds of error that are known to occur in computer programs, including the null reference error that I described earlier. More modern tools can automatically supply missing assertions, and use them to generate test cases that will reveal errors in newly written code. There are now hundreds of specialist verification engineers and scientists employed throughout Microsoft. They have developed and evolved a wide range of verification and analysis and testing tools, and helped in installing them in routine use.

In my last five years with Microsoft research, my interests have returned back to unifying theories, the subject of the final failure of my academic career. But now I am taking a completely different approach. I have formulated a couple of dozen algebraic Laws of Programming. They are very similar to the laws of arithmetic taught to generations of school-children. They could be taught at the beginning of any undergraduate curriculum in Computer Science. They could be illustrated by exercises of their use in optimising computer programs by correctness-preserving transformations. I think they would make an excellent introduction to formal methods for all computer scientists.

Strangely, my first publication on the Laws of Programming was in 1987, before the start of the CONCUR project, and before the start of the research leading to publication of the book on Unifying theories. This article already contained all the laws relevant for sequential programming languages. But somehow, throughout the next twenty years, I never recognised that this article already showed the simplest way of unifying theories of programming. It is only in the last five years that I have seen how to introduce laws for concurrency into a programming language. As a result, the laws apply equally to sequential and to concurrent programs.

Using these laws, it is possible to prove the correctness not only of Hoare logic, but also its extension to separation logic by Peter O'Hearn. The same laws can also prove the correctness of the operational semantics which Robin Milner used to define his process algebra CCS. As a result I believe that the Laws of Programming at last fulfil the objectives of the CONCUR project, which failed some twenty years ago.

I am very happy if this should turn out to be the last achievement of my working career. It is an achievement which has the property of simplicity and elegance, which is often a distinctive characteristic of the most convincing theories in science. The algebraic approach to computer programming will be for many people quite surprising. But for me, the main surprise has been how long it took me to gain the relevant insight, to realise that I already knew how to solve the problem of unification. The moral of this story is quite the opposite of those that have gone before. I have spent the main part of this address telling you stories about how much I have learnt from my failures. But in this last story about unification of theories, I have told you how much I failed to learn from one of my successes.

In conclusion, may I refer again to the laudation. I was most flattered to hear how many of my discoveries in Computer Science have been incorporated in the undergraduate curriculum of the Universidad Complutense. I hope that your undergraduates find the subject interesting at first and useful later. But please do not think that what we now know and teach in Computing Science is the end of the story. There are many new topics opening up in our subject, and many new insights and understanding to be revealed, and new applications to be opened up. I shall be very happy if your new researches are built on my earlier successes; but I should be equally happy if they were built on my failures.