The 1980 ACM Turing Award Lecture
The Emperor's Old Clothes

The 1980 ACM Turing Award was presented to Prof. Charles Antony Richard Hoare for his fundamental contributions to the definition and design of programming languages.
In this talk, C. A. R. Hoare has shared his experiences, especially the failures which taught him effective lessons in his journey. He hopes that the future generations also learn from them. He narrates his journey at Elliott Brothers (London) Ltd, where he started as a programmer in 1960. He challenged himself with his first task of developing a fast method of internal sorting, and developed QuickSort but could not express his work effectively. Later on, he was given a much more important task of designing a new advanced high-level programming language for the company's next computer. By great good fortune, he came across the language ALGOL 60, which interested him. So, he attended a course on ALGOL 60 which was offered in Brighton, England, with Peter Naur, Edsger W. Dijkstra, and Peter Landin as tutors. The concept of recursion in this language helped him to express his previous invention "QuickSort" elegantly. He, along with his team, decided to design a modest subset of ALGOL 60, instead of creating a new language altogether. During the design, he adopted four basic principles. The first being *Security*, every syntactically incorrect program must be rejected by the compiler and vice-versa. The Second principle *brevity of the object code produced by the compiler and compactness of run time working data.* There was a clear reason for this: The size of main storage on any computer is limited and its extension involves delay and expense. The third principle being the *entry and exit conventions for procedures and functions to be as compact and efficient as for tightly coded machine code subroutines*. The final principle is that *the compiler should use only a single pass*. In the process, they were able to discover new methods of relaxing the restrictions without compromising on any of the above principles and implemented a successful version of the compiler.

As a result of this work, he was invited to serve in the new working group 2.1 of IFIP, charged with the development and maintenance of ALGOL. Each member of the team was invited to suggest some improvement which he felt were the most important. His suggestion which was to relax the rule of compulsory declaration of variables was rejected. The reason being this relaxation would destroy the strong redundancy of ALGOL. A proof to this was seen when the Mariner space rocket to Venus got lost due to the same reason. The second language design proposal he made was the notation for "case" expression.

Back again at Elliott's, his team started the Mark II Software project which had larger configurations than the previous one. Though he toiled hard, they could only achieve a speed of 2 characters per second and failed to reach the customers expectations. With this failure, the company assigned to him the task of mending the situation. So, they started analysing the possible mistakes which could have led to the failure, but in vain. Only when Andrew, a general manager of their parent company, pointed out that his ignorance to every detail of the project was the main issue, they were able to traceback. Then they started listing out grievances of customers and programmers and found out that over-ambition was their main-failure. There was also failure in prediction, in estimation of program size and speed, of effort required, in planning the coordination and interaction of programs, in providing an early warning that things were going wrong. He learnt from his failures and devised a strategy to classify their customer base into groups and address their issues independently. He would only agree to those features which he understood and felt reasonable enough. It worked and within two years they had moderately satisfied customers.

Later he devoted himself to research on designing programming languages. He proposed a formal method to define a programming language using axioms, and published his first paper on the same. Later on he reviews many projects on designing new programming languages. In the process, he concludes that there are two ways of constructing a software design: *One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies*. The first method being far more difficult than the second. Another conclusion he makes is that when any new language design project is nearing completion, there is always a mad rush to get new features added before standardization. The rush is mad indeed, because it leads into a trap from which there is no escape. *A feature which is omitted can always be added later, when its design and its implications are well understood. A feature which is included before it is fully understood can never be removed later.*

He also expressed his views that the applications today are complex because we are ambitious to use our computers in ever more sophisticated ways. Programming is complex because of the large number of *conflicting objectives* for each of our programming projects. If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution.

At the end, he narrates a story, "The Emperor's Old Clothes", which indirectly relates to his journey in the computing world. The story is about an Emperor who is obsessed with clothes, like programmers are generally obsessed with features. One day he gets tricked into going naked, and so he orders that each of his many new suits should be draped on top of the old. When a new tailor visited the Emperor, he advised him to rather remove layers of his clothes

and strive for simplicity and elegance. One fine day, the Emperor realizes how foolish he had been and moves to a happier life in another story. Just as in Hoare's journey, the over-ambitious requirements forced him to add more and more complex specifications, ultimately leading to failures. Only after which, he understood the power of simplicity was he able to succeed.