DOI:10.1145/2693199

Vinton G. Cerf

A Long Way to Have Come and Still to Go

I AM a big fan of Robert Heinlein's stories and novels. His early and mid-career works generally appeal to me more than his later novels, which brings me to *Starman Jones*,^a written in 1953, and almost certainly intended for nerdy teenage guys. I actually had not read this novel until recently. I had read *Tunnel in the Sky*, which attracted a similar audience. Young kid, in his teens, thrust into situations that would challenge a capable and experienced adult, coming out a success. Not unlike the early Horatio Alger novels, Heinlein emphasized hard work, honesty, a strong moral compass, winning out against unfair and discriminatory odds. Makes me think of Harry Potter, too! Who doesn't like it when the underdog wins out?

But this is *not* the reason I wanted to draw attention to this novel. In 1953, computing was still very much in its infancy, but Heinlein has his protagonist using books of tabulated calculations and values and manually converting numbers into binary form to *program* the computers onboard a starship. Information is relayed *orally* and keyed in, in binary! All this under time-pressure to make a jump to light speed at exactly the right time.

In 1953, IBM released the model 701. From Wikipedia:^b

The system used vacuum tube logic circuitry and electrostatic storage, consisting of 72 Williams tubes with a capacity of 1024 bits each, giving a total memory of 2048 words of 36 bits each. Each of the 72 Williams tubes was 3

inches in diameter. Memory could be expanded to a maximum of 4096 words of 36 bits by the addition of a second set of 72 Williams tubes or by replacing the entire memory with magnetic core memory. The Williams tube memory and later core memory each had a memory cycle time of 12 microseconds. The Williams tube memory required periodic refreshing, mandating the insertion of refresh cycles into the 701's timing. An addition operation required five 12-microsecond cycles, two of which were refresh cycles, while a multiplication or division operation required 38 cycles (456 microseconds).

Heinlein's rendering of programming in the far future was even more primitive than programming the IBM 701, which was largely done in assembly language, as Marc Rochkind documents in his draft history.^c We have come a long way from 1953 and, ironically, oral interaction with computers has become increasingly common, not to enter binary codes, but to speak to the computers to get them to accept and execute tasks. For example, pretty much anywhere you use a keyboard, Google's applications now allow you to speak. We are still far from serious dialogue, but one can begin to imagine this possibility.

The history of programming is one of increasing abstraction and context. High-level frameworks and built-in libraries fashioned in layers together with parallel operation and remote procedure calls mediated by networking protocols have changed the programming landscape over the past 60 years. Anyone who has programmed spreadsheets or created presentations can appreciate the power of high-level constructs. Programming in languages

like Python, JavaScript, Ruby on Rails, or HTML5 seems to me rather different from FORTRAN, ALGOL, and C++. I look forward to hearing from readers with experience in some or all of these languages and applications to learn how they view the progress of programming over the decades.

I have written before about the problems we still face with regard to programming errors and I will not repeat my diatribe here, but one does wonder to what extent the mistakes we make are in part a consequence of the level of language in which we express our intentions. The lower the level of expression, perhaps the more generality we can achieve, but at the potential risk of making serious mistakes. One is reminded of Edsger Dijkstra's famous letter to *Communications* "Go To Statement Considered Harmful."^d This leads me to wonder whether it is possible to write significant programs by way of high-level oral (or written) interaction with a programming system. For this to work, I suppose it has to be possible to *discuss* with the programming system the objectives of the program, to enter into fairly specific descriptions of algorithms, and to interact with a growing body of program text that represents the programming system's understanding of the programmer's intent. One might have to be able to ask questions about the evolving software and its anticipated behavior. We have a long way to go to reach such an objective.

In a kind of Turing test variant, one might pit a really good programmer against an automatic system, with the party carrying out the negotiated programming effort trying to distinguish between the automatic and the manual programmers. ■

^d *Commun. ACM* 11, 3 (Nov. 1968), 147–148

Vinton G. Cerf is vice president and Chief Internet Evangelist at Google. He served as ACM president from 2012–2014.

Copyright held by author.

^a Robert A. Heinlein, *Starman Jones*, ISBN 0-345-32811-6, Del Rey published by Ballantine Publishing Group, 1953.

^b http://en.wikipedia.org/wiki/IBM_701

^c <http://basepath.com/new/701/Programming-the-IBM-701.pdf>