In “The Humble Programmer” Dijkstra offers an overview of the programming universe as he saw it in 1972.

As the power of available machines grew by a factor of more than a thousand, society’s ambition to apply these machines grew in proportion, and it was the poor programmer who found his job in this exploded field of tension between ends and means. The increased power of the hardware, together with the perhaps even more dramatic increase in its reliability, made solutions feasible that the programmer had not dared to dream about a few years before. And now, a few years later, he had to dream about them and, even worse, he had to transform such dreams into reality.

From its title to the repetition of phrases like “the poor programmer” to its references to “the intrinsic limitations of the human mind,” the essay presents a chastened vision of the human capacity to create software: the programmer, Dijkstra says, must be “fully aware of the strictly limited size of his own skull.”

Despite this deliberate lowering of the eyes, Dijkstra does offer a not-so-humble prediction: that, “well before the seventies have run to completion, we shall be able to design and implement the kind of systems that are now straining our programming ability, at the expense of only a few percent in man-years of what they cost us now, and that besides that, these systems will be virtually free of bugs.” The first prediction essentially came true, as the programming field managed to continue to produce increasingly ambitious systems. The second, however, is still nowhere in sight.

Dijkstra presents arguments for why his prediction is technically feasible, built on the proposition that we should “confine ourselves to intellectually manageable programs.” This sounds sensible, but it will always run up against the nature of software innovation, which always pushes us to try new stuff at the edge of what’s possible, because that’s what’s *worth* doing.

There’s something pleasing about the down-to-earth way Dijkstra expresses the strangely-bound-together optimism and pessimism of his world-view, in which the trajectory of software is always progressive — but that very progress is what keeps the process from getting any easier.

Programming will remain very difficult, because once we have freed ourselves from the circumstantial cumbersomeness, we will find ourselves free to tackle the problems that are now well beyond our programming capacity.

I wonder if other readers will have the reaction I had to Dijsktra’s argument that bugs are simply not to be tolerated: “If you want more effective programmers, you will discover that they should not waste their time debugging, they should not introduce the bugs to start with.” If he means, it’s easier to fix errors at the time you make them rather than much later, that makes some sense; but it sounds more like “you simply shouldn’t make any mistakes in the first place.” That would be nice. But it doesn’t sound like the real world programmers everywhere have experienced, all the way back to that seminal moment in the ’40s when Maurice Wilkes realized he would spend a “large part of his life” correcting his own mistakes.

Those who want really reliable software will discover that they must find a means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging - they should not introduce the bugs to start with.

We should confine ourselves to intellectually manageable programs. We must not forget that it is not our business to make programs; it is our business to design classes of computations that will display a desired behaviour.

It is a usual technique to make a program and then to test it. But program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence.

The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.

The question: "Can you code this in less symbols?" or, "Guess what it does?" - as if this were of any conceptual relevance!

As long as machines were the largest item in the budget, the programming profession could get away with its clumsy techniques.

Reference:

1. TuringAwardLecture: "The Humble Programmer," by EwDijkstra - 1972.