Advanced Course on Computer Systems Architecture. (Grenoble, December 1972)

Summary of the talks on "Programming Methodology" by Edsger W.Dijkstra.

The talks will deal with five main subjects:

1) On the Necessity of Correctness Proofs.
2) On the Mathematical Structure of Correctness Proofs.
3) On the Feasibility of Correctness Proofs.
4) On the Impact of Correctness Concerns on System Structure.
5) On the Impact of Correctness Concerns on the Process of Program Composition.

Ad 1. This is a very short section the moral of which can be summed up by the observation that program testing can be a very effective means for demonstrating the presence of bugs, but is a hopelessly inadequate method for showing their absence. As a result, correctness proofs remain as the only alternative by means of which we can hope to reach the required confidence level.

Ad 2. For sequential programs an axiomatic definition of semantics will be proposed that can serve as a mathematical basis for correctness proofs. For dealing with repetition, the method of the invariant relation will emerge as the most practical tool. This experience will be carried over to co-operating sequential processes, with "mutual exclusion" as the logical corner stone. The problems of deadlock and of individual starvation are introduced as the parallel programming analogue of the termination problem.

Ad 3. When "in principle" correctness proofs can be given, we still have the obligation to arrange matters in such a way that they can also be given in practice. This is no trivial obligation, because without special precautions, the necessary amount of manipulation -be it formal or informal, be it mechanized or done by hand- required for the correctness proofs will explode for all but the most simple systems. Abstraction will be presented as a method for arranging our thoughts in such a manner that the practical feasibility of correctness proofs is not unnecessarily impaired. It will become appparent that the purpose of abstraction is not to be vague, but to create a new semantic level on which we can be absolutely precise. It will also become apparent that, if the method is to be practical at all, the system write-up should reflect the abstractions in terms of which we can continue to understand what we are designing.

Ad 4. The abstractions introduced serve to understand the possible behaviour of the system and therefore we must require that they admit a sufficiently truthful implementation. By way of illustration it will be shown how certain hardware features impair the achievable truthfulness to such an extent that we may question whether equipment with such properties admits intelligent use at all. Apart from truthfulness of the implementation we require a form of robustness, excluding the situation in which the data, on which individual basic assertions about system behaviour rely, are scattered all through the system. By way of illustration, hardware features violating this second requirement will also be given.

Ad 5. The one impact of correctness concerns on the process of program composition is that our basic software has to satisfy requirements analogous to the ones that in the previous section we imposed upon hardware. Besides that we shall show some of the formal techniques by means of which the process of program composition can be aided very effectively, formal techniques that allow us to derive from the requirements a program satisfying the correctness proof (salvo errore et omissione) automatically.