Reinhard KEIL-SLAWIK

DEPARTMENT OF COMPUTER SCIENCE H. NIXDORF INSTITUTE

COGNITIVE IMPERIALISM

TO COMMENT ON SIMON'S ARTICLE IS A DIFFICULT ENDEAVOUR. First, neither he nor I are literary critics; I am a computer scientist, he is a cognitive scientist. Second, Simon touches on many basic issues and concepts in such a brief way that it is impossible to assert what cognitive psychology has to offer to literary criticism—apart from adding a new and imperialistic (as Simon admits) view to the field. Last but not least, his arguments are based on the physical symbol system hypothesis which I do not share. However, discussions of the validity or appropriateness of this hypothesis fill up already thousands of pages—no need to add yet another page.

On the other hand, it is the central issue in Simon´s paper, and thus needs some consideration. I will try to cope with these problems by selecting one issue that may help to shed some light on the underlying assumptions and consequences of this hypothesis, and which, at the same time, is related to a common field of interest for both of us, namely programming and understanding programs. The crucial question is: Can meaning be located in a (program) text, and hence be deduced by interpreting this text?

It is my hope that the way I characterise the problems and draw the conclusions is not completely irrelevant to the field of literary criticism.

The problem of locating *meaning in a text* is a crucial issue for software engineers because the engineering perspective is strongly tied to the idea of writing a complete, consistent, and unambiguous specification of a system such that this document could be handed over to other persons and they would know exactly what to do. This would imply that the meaning can be located in the respective text. If this were true, it would equally well apply to all design artefacts, be it programs (source code), data models, requirements definitions, (mathematical) specifications, or user manuals; and it would solve a lot of problems.

However, more and more empirical studies and laboratory observations of the programming process reveal that the meaning created in the communicative processes among programmers, designers, users, and managers, cannot be captured by documents1. And there are numerous practical examples showing us that we should not confuse the result of a process, the document, with the process as such. We have to distinguish between *programming as discourse*, which is a meaning-creating activity, and *programs as text*. Software development is not only the production of a product, but a long and tedious learning process. Design artefacts emerge as a trade-off between various interests and technical alternatives rather than representing self-contained specifications of technical solutions to well-known problems.

If we regard software as a mathematical object that is interpreted by a machine, its semantics are a static attribute of the program text. Once the instruction sequence is fixed, the behaviour of a program is determined solely by the input. However, the crucial point for developers as well as for users is determining whether a given instruction sequence is appropriate for supporting execution of the task at hand, i.e., finding out which input sequence will produce the desired output in a suitable and comprehensible manner. Since software embodies a variety of claims and assumptions about the context and the nature of the problems to be solved by installation of the system at the workplace, the properties describing the relations between software and the usage context cannot be expressed in terms of formalisms. Too many mutually influential factors have to be taken into account. The nature of the problem as it is perceived by the designers changes with every new insight, and very often incompatible requirements lead to design conflicts that have to be resolved.

The knowledge required for design, then, has to be built up in the course of a tedious and often painful learning process. During this process, the designers learn which aspects fit into their already developed framework, and which ones require redesign, correction, or restructuring of already existing design artefacts and programs. The reasons and motivations behind such changes, and the arguments concerning how these changes are to be achieved while maintaining the overall quality of the design, are not part of a program or its specification, and they cannot be documented in their entirety.

The same holds from a user´s perspective. As designers of interactive systems, we have learned that users do not understand systems by reading bulky user manuals. It is through the combination of reading manuals, using the system, and talking to fellow colleagues that the meaning is created. Thus, to foster communication and learning among the parties involved in the development process, documents have to be complemented by prototypes (executable programs), and the development of both of them requires extensive communication.

To conclude: in systems development meaning is not a relation between an individual human being and a text. It is a combination of acting and communicating in a rather complex social setting. Reading and writing only covers a small, although important, fraction of the overall meaning-creating process. Meaning cannot be located in a text as long as the interpretation of the respective text requires some kind of learning. If no learning is required, we can behave as if the meaning would reside in a text because we adhere to a frame of reference such as defined by conventions, standards, rules, or theories that determine the possible interpretation. Meaning resides in the social processes of developing, using, and, especially, revising such regulations and conventions due to our experience. Our behaviour is, of course, structured to a large extent by regular patterns or the execution of mechanisms because these patterns and mechanisms provide shortcuts for the human mind. We do not have to learn the same thing over and over again. By finding an appropriate symbolic representation, we may even be able to delegate the symbolic transformations that do not require human-conscious interpretation to a machine. Cognitive science may help us to produce explicit descriptions of such mechanic transformations. Thus, to the extent the task of literary criticism would be to identify such regular patterns, cognitive science could be of some use. If, however, the primary problem is not to determine the meaning in the most precise way but to trigger discourses and to initiate changes, or to enable our scholars to find their own way of interpreting the world rather than to seek for the most and only appropriate interpretation, then the fight among different schools in the form of constructive critique may prove to be a prerequisite to accomplish this goal.

Jerome Bruner has made us aware of the *paradigmatic and the narrative mode of thought* and that both of them complement each other—neither one can be replaced by the other (Bruner, 1984). Simon advocates the paradigmatic mode of thought but fails to describe its limitations. I will not deny that cognitive science may provide a productive stimulus for literary criticism, but in its imperialistic form, I am afraid, it will only add yet another doctrine to the "current noisy combat"—and, even worse, in the form presented it might not be the best doctrine one could add.

NOTES

1 For an excellent overview of the various facets involved here as presented by scientists with different scientific backgrounds see Floyd et al. (1992), and Naur (1992).