on deepening our knowledge in these areas before making direct assaults on the synthesis of particular categories of programs. After all, the Wright Brothers built a wind tunnel and studied properties of airfoils extensively before they conquered flight.

THE IMPACT OF AUTOMATIC PROGRAMMING RESEARCH—Michael Hammer

Contemporary automatic programming research encompasses a variety of approaches to the goal of transferring some of the programming function from human to machine and thereby alleviating the software problem. A very high level language system translates a program expressed in terms of nonprocedural, problem-oriented constructs into conventional algorithmic code; in so doing, it must perform global program transformations in order to achieve an efficient object program. A knowledge-based system uses the semantics of a particular problem domain to interpret a user's functional specifications for a program in that domain and to construct a system with the specified functionality. A program synthesis system utilizes formal theorem-proving techniques to transform non-functional program specifications (typically expressed in terms of a set of input-output pairs) into an executable program.

Although extensive research has been conducted in each of these areas, and significant advances have been made in addressing the technical problems that each presents, it is unlikely that any of this work will culminate in systems that will meet the ambitious goals that have been set for them. The very high level languages that have been developed so far are adequate for expressing the main body of a program in a nonprocedural way; however, they do not address issues related to deviations from the normal flow of the computation, such as special case recognition, exception handling, and coping with erroneous data or system failure. Furthermore, introducing a new language into an operational environment has in the past proven to be a major undertaking, one that raises serious managerial and technical problems. The principal problem facing the more advanced, artificial intelligence systems is one of scale. It is doubtful that the techniques that suffice for laboratory prototypes can be extended to cope with realistically rich problem domains and large, complex programs.

The difficulty underlying all of these problems is that the fundamental premise of automatic programming is simply false. This premise is that, after thirty years' experience in writing programs, we now know enough about the programming process to automate it. On the contrary, it appears that our understanding of the nature of programming is weak. The ad hoc attacks on the software problem that are currently feasible are not likely to produce major results.

Nonetheless, automatic programming research will have important consequences, both direct and indirect, for the future of software construction. In the near term, we should see limited systems that automate one part of the programming process or that are adequate for the synthesis of small programs; in either case, the context will be a simple problem domain that has a well-defined and understood semantic structure. Examples of such systems include an automatic data base design system and a facility for generating simulation studies. Another possibility is a system that uses knowledge of an application domain to assist a user in deriving a complete and consistent set of program specifications. In another vein, very high level language research should lead to the design of formal specification languages for effective and precise inter-personal communication.

However, the most significant benefit of automatic programming research should be a deeper understanding of programming by human programmers. It has often been the case that the attempt to automate an activity has forced its practitioners to understand and systematize their actions; automatic programming research should have the same effect on programming. In the attempt to computerize the software production process, it is uncovering important general programming principles and techniques. The structures and guidelines that derive from this work will enable and encourage programmers to be more disciplined and organized; in the long run, they can contribute to turning programming from mysticism into science. In other words, automatic programming research should benefit software engineering practice.
PART III—SYSTEMS