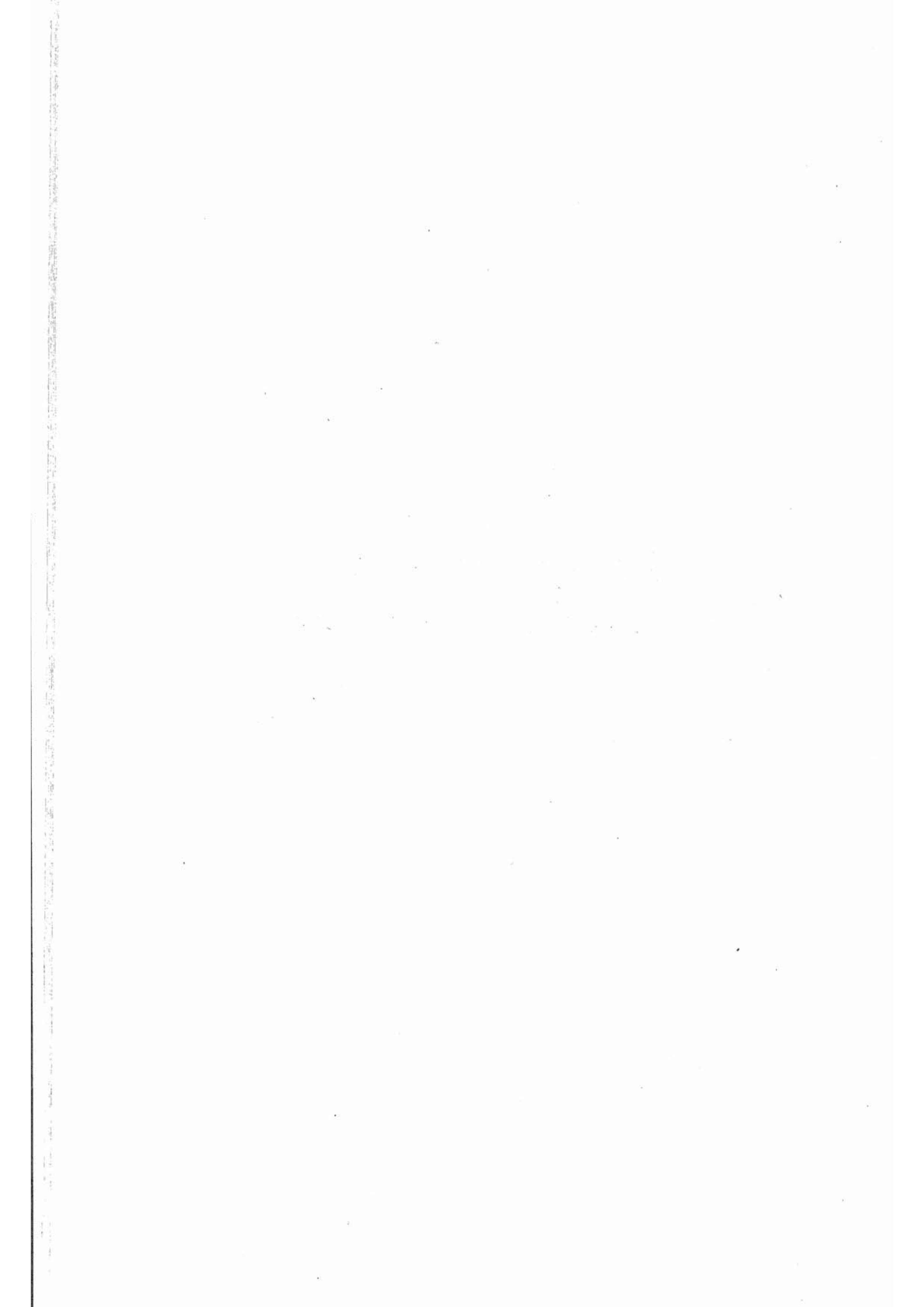
Part 1

Theoretical and Methodological Issues



Programming is a human activity that is a great challenge, involving the design of machine behaviour that can assist, and at times replace, humans in intellectual tasks. Seen in this light, programming is a meta-activity, and the study of programming is useful to the development of basic knowledge in cognitive psychology. For the most part, research works in the psychology of problem solving deal with result-production situations where subjects are asked to attain a goal without being required to express a general procedure to reach this state as is the case of program-production situations, such as sequencing in manufactures, designing directions for use, etc. (Miller, 1981; Hoc, 1988; Chapter 2.3). Computer programming is a valuable and rich paradigm for studying this latter type of situation. In turn, computer science can draw on psychological studies of programming to design more appropriate programming languages and environments as well as more efficient training curricula. For example, in a context of development of top-down programming methodologies, the observation that even expert programmers are not able to conform themselves to this kind of strategy, but show 'opportunistic' strategies which mix top-down and bottom-up components, may lead computer scientists to design more efficient support to programming which does not ignore this fact.

The history of the psychology of programming dates back to the 1960s, probably to a French work which analysed business programmers' behaviour in terms of top-down, data-oriented programming methodology (Rouanet and Gateau, 1968). The broad range of individual differences in the structuring of programs designed to solve the same problem led the authors to the conclusion that a programming methodology was needed for programmers to teach them to use high-level (more abstract) representations of information and control flow to correct for the saliency effects of low-level machine constraints. These results are consistent with more recent studies of planning which indicate that programmers need support to process schematic representations before implementing them in concrete devices (Hoc, 1988).

This kind of research was the exception rather than the rule in the early days of the psychology of programming. Most studies were produced by computer scientists who developed a normative approach to what they considered to be the most powerful programming concepts. The book by Dahl *et al.*(1972) on the principles of structured programming is a good example.

During the 1970s, the 'second generation' of programming studies was dominated by a number of empirical studies, comparing diverse types of languages (e.g. diverse expressions of conditionals), program layouts (e.g. flowchart, indentation, etc.), methods (e.g. top-down, modular, etc.), and so on. The watershed publication initiating this era was by Weinberg (1971) which argues that a psychological viewpoint should be incorporated into any approach to programming. The seminal changes in theoretical frameworks in the study of human problem solving had only just appeared (Newell and Simon, 1972), and Weinberg makes reference to aptitude theory which was better known at that time but was in fact ill suited to the issues raised by programming. Nevertheless, most of the major orientations now currently discussed were present in this book, especially the need to increase our knowledge of the cognitive processes underlying programming and its learning, and to define more accurate indicators of this activity and experimental investigations using tasks and subjects more representative of real programming.

Most experimental studies at this time were conducted by computer scientists, and very few referred to psychological theory or methodology. The purpose of these

studies was rapid assessment of tools on the sole basis of finished products and no attempts were made to gain insights into the activity itself. This yielded a number of problematical and sometimes contradictory experimental results. A good example was the evaluation of flowchart use in comparison with listing, for which improvements (Wright, 1977) as well as lack of effect (Shneiderman *et al.*, 1977) had been shown on overall performances without understanding the reasons for this difference. However more detailed analyses of performance enabled researchers to precisely define the activity components affected by flowchart use (Brooke and Duncan, 1980; Gilmore and Smith, 1984): overall performance is shown to be improved only in situations where these activity components are crucial. Shneiderman (1980) provides a good review of the state of the art at this 'second generation' time.

When an experiment is not initially thought out within the framework of a theory, interpretation is seldom straightforward. As far as psychological processes are concerned, sole reference to a computer science theory can be misleading, especially when there are major disparities between computer science objects and programmer representations. Program complexity measures are particularly informative in this respect. A complexity measure is based on a certain way of structuring programs. In order to be psychologically valid for a certain kind of programmer, the structure must be consistent with the representational structures this programmer uses. For example, Halstead's metrics (1977) evaluate information transmitted by the operators and operands in the program. The structure which is considered here is the surface structure of the program, probably valid for beginners or professionals confronted with a very unknown program. As far as a deeper structure is concerned, this measure is inadequate since it ignores the high-level structures (chunks) used by expert programmers in understanding programs (Curtis *et al.*, 1984).

The 'third generation' of the psychology of programming was initiated in the early 1980s by a wide-ranging debate on the theoretical (e.g. see Hoc, 1983) and methodological (e.g. see Moher and Schneider, 1982) grounds for this kind of study. Pioneering efforts were followed by an increase in the number of psychologists studying programming, especially notational and debugging aspects (Green, 1980). At the same time, some computer scientists in the cognitive science field were developing cognitive approaches to programming that had a direct bearing on Psychology, e.g. Soloway *et al.* (1982), studied programming knowledge representations in terms of hierarchical schemas and plans.

The originality of these recent studies lies in a more indepth investigation of programming as an activity, through tools such as individual protocol analysis and cognitive modelling. These studies have benefited from enhancement by cognitive psychology methodology, which aims to elicit the externalization of covert behaviour. New experimental paradigms have been generated from psychological theories (on text comprehension, human problem solving and planning, etc.; see Chapter 1.4) and hypotheses drawn from observation (especially verbal report techniques; see Chapter 1.5).

The aim of the present volume is to explore this fast-growing trend in the psychology of programming. The value of psychological frameworks is stressed, but greater emphasis is placed on the need for a combination of psychology and computer science approaches. Computer science has developed languages, tools and environments that implicitly represent diverse conceptions of programming and enter into the definition of programming tasks (e.g. task requirements). Programming activity cannot be de-

fined in isolation from this cultural environment. Computer science conceptions need to be assessed before investigating their role in programmer activity. A number of psychological difficulties can be anticipated by an *a priori* analysis of the coherence of these conceptions. Floyd (1984) provides this kind of analysis in her comparative evaluation of programming methods (e.g. decomposition of the programming process into sequential steps which turn out not to be independent and well defined when they are implemented).

The implication is that a psychological investigation must begin with analysis of these rapidly evolving concepts which range from very procedural approaches to functional, logical, object-directed, and more-declarative ones (see Chapter 1.1 by Pair). At the same time the number of styles of interacting with programs is rapidly increasing. Today's programmers have far more variety of possible approaches to choose between (see Chapter 1.2 by Green). Most of the studies referred to in this book deal with procedural programming, but very recent advances that are still unavailable to the general public are also presented. Pennington and Grabowski (Chapter 1.3) describe the richness of programming activity by defining its cognitive components, and their relationships, beyond program design, e.g. problem and program understanding, debugging, etc. An introduction to cognitive psychology is presented by Ormerod (Chapter 1.4) who selects the main concepts that are relevant to the psychology of programming and are used in the diverse chapters of this book. Gilmore (Chapter 1.5) introduces the reader to observational and experimental methodology in cognitive psychology and shows how scientific results are obtained.

The theoretical and methodological issues presented in Part 1 are complemented by Parts 2 and 3 which review the main research findings in two areas: features of programming languages and the learning of programming (Part 2), and expert skills and job aids (Part 3).

These studies have been run in more or less simplified situations, even though some are devoted to real programming tasks with a realistic level of complexity. The final part of this book addresses broader issues, related to everyday programming in companies.

Readers should measure the ecological validity of the data presented here in relation to the type of programmer the study deals with. Programmers, whether they are novices or experts, do not constitute a homogeneous population.

Novices belong to at least two distinct categories and their goals in the learning of programming differ. At school, programming is taught for enrichment, or in a transfer perspective which assumes that other knowledge can be acquired during programming learning (e.g. some understanding of mathematical concepts which can help the pupil in mathematics). A very limited amount of time is devoted to programming, which raises the issue of its true educational value (see Chapter 2.5). Adult and young adult novices can train to become professional programmers. Here training can be lengthy and cover several converging types of knowledge pertaining to coding in a programming language, using well-known algorithms, representing specifications in an efficient way for programming, etc. The relevance of teaching material to the working world is essential for assessing training curricula of this type.

Experts are not more homogeneous as a group, and can be defined by the regularity of activity, and the level and nature of expertise.

Activity can be regular or casual. Regular programmers are those who spend their time designing, coding, debugging, documenting and modifying programs. Casual

programmers are people whose main activity is not programming. They work in other fields and they program for professional purposes. In terms of training as well as job aids, the needs of these two kinds of programmers are quite different. Few studies have been devoted to casual programmers who are certainly in the majority and who require intelligent support systems, such as advice giving systems (e.g. see Giboin, 1988).

Expertise can be general or specific. General programmers need broad expertise in languages, tools, methods, team work, etc. They may be called upon to deal with almost any kind of programming environment or problem domain during their careers. They have to learn to use multipurpose tools and produce high-quality software. More-specialized programmers are expert in a particular application domain, such as management, statistics, programmable controller programming (see, for example, the work done by Visser (1987) referred to in Chapter 3.3), etc. For this type of programmer, specific-purpose languages and tools can be developed and taught.

References

Brooke, J.B., Duncan, K.D. (1980). Experimental studies of flowchart use at different stages of program debugging. *Ergonomics,* **23**, 1057-1091.

Curtis, B., Forman, I., Brooks, R., Soloway, E., Ehrlich, K. (1984). Psychological perspectives for software science. *Information Processing and Management,* **20**, 81-96.

Dahl, O.J., Dijkstra, E.W., Hoare, C.A.R. (1972). *Structured Programming.* London: Academic Press.

Floyd, C. (1984). *A Comparative Evaluation of System Development Methods.* Berlin: Technische Universität Technical Report.

Giboin, A. (1988). The process of intention communication in advisory interaction. *3rd IFAC Conference on Man-Machine Systems, Oulou (Finland).*

Gilmore, D.J., Smith, H.T. (1984). An investigation of the utility of flowcharts during computer program debugging. *International Journal of Man-Machine Studies,* **20**, 357-372.

Green, T.R.G. (1980). Programming as a cognitive activity. *In* H.T. Smith and T.R.G. Green (Eds), *Human Interaction with Computers.* London: Academic Press.

Halstead, M.E. (1977). *Elements of Software Science.* New York: Elsevier.

Hoc, J.M. (1983). Psychological study of programming activity: a review. *Technology and Science of Informatics,* **1**, 309-317.

Hoc, J.M. (1988). *Cognitive Psychology of Planning.* London: Academic Press.

Miller, L.A. (1981). Natural language programming: styles, strategies, and contrasts. *Perspectives in Computing,* **1**, 22-33

Moher, T. and Schneider, G.M. (1982). Methodology and experimental research in software engineering. *International Journal of Man-Machine Studies,* **16**, 65-87.

Newell, A. and Simon, H.A. (1972). *Human Problem Solving.* Englewood Cliffs, NJ: Prentice Hall.

Rouanet, J. and Gateau, Y. (1967). *Le Travail du Programmeur de Gestion: Essai de Description.* Paris: AFPA-CERP.

Shneiderman, B. (1980). *Software Psychology: Human Factors in Computer and Information Systems.* Cambridge, MA: Winthrop.

Shneiderman, B., Mayer, B.R., McKay, D. Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. *Communications of the ACM,* **20**, 373-381.

Soloway, E., Ehrlich, K., Bonar, J. and Greenspan, J. (1982). What do novices know about programming? *In* A. Badre and B. Shneiderman (Eds), *Directions in Human-Computer Interaction.* Norwood, NJ: Ablex.

Visser, W. (1987). Strategies in programming programmable controllers: a field study on a professional programmer. *In* G. Olson, S. Sheppard, and E. Soloway (Eds), *Empirical Studies of Programmers: Second Workshop.* Norwood, NJ: Ablex.

Weinberg, G.M. (1971). *The Psychology of Computer Programming.* New York: van Nostrand.

Wright, P. (1977). Presenting technical information: a survey of research findings. *Instructional Science,* **6**, 93-134.