EPILOGUE: WHY DID I WRITE THIS BOOK?

Learning computer programming was somewhat problematic for me. During my first year, when I started my studies in the Department of Electrical Engineering at the University of Oulu, Finland, I had to study a course of computer programming. The course introduced the Fortran programming language. I understood practically nothing about it at that time. A couple of years later, we were given a course in the general operating principles of computers, and we studied machine-level (assembly language) programming. This later course had a tremendous impact on me. I started to understand things. I could understand why programming languages are like they are, and why the Fortran statements had to be so strange as they were. Because of these early experiences, my opinion is that a person needs to understand to some extent how a computer works before he or she can learn to write computer programs. For this reason, the basic operating principles of computers are explained in Part I of this book.

So, when I started to understand things related to computers, I decided to concentrate on programming and software development in my studies. Later on, I worked in software companies and found out, among other things, that it is often very difficult to read and understand computer programs written by other software developers. My life as a professional software developer also taught me that making software involves learning new things.

A turning point in my career was the morning of September 21, 1988. That morning I wanted to write a new piece of source code that would enhance the system that my project had been developing for about half a year. When I wrote that program, I decided not to use any abbreviations like i, j, ptr, buff, ch, etc. in the names I used in the program. Instead, I named all variables with long names consisting of up to five natural words. I was a little bit unsure whether programs could be written that way, but it turned out to be possible to write programs with long names without abbreviations, and the program I wrote on that September morning worked very well.

I still cannot say exactly why I tried this "programming without abbreviations". I wanted to try something new in my work. I wanted to write programs that would be easy to read and understand. The reason why I was able to try a new kind of programming approach on that September morning was that we had recently got new compilers which allowed long names in programs. One reason why abbreviations have traditionally been used in source programs is that earlier compilers did not accept long names. In old computer programs the names of variables and functions had to be abbreviated to 6 or 8 characters.

Anyway, after the mentioned September morning, I was very excited about the new programming style. I started to write long names in my programs, and I stopped the use of abbreviations altogether. Later on I found the term "natural naming" to describe this programming approach. The term was in the title of an article by Daniel Keller (ACM SIGPLAN Notices, Vol. 25, No. 5, 1990). When I started to work at the Technical Research Centre of Finland, VTT, I was able to test my programming ideas on scientific forums, and I wrote a doctoral thesis which is related to the use of natural naming in software development. While working at VTT, I gave short naming courses for software developers in private companies. Many people liked my programming ideas, but there were some people who did not want to give up using abbreviations.

The use of natural naming turned out to be the right way for me to write computer programs. My abilities as a programmer grew when I started to spend more time in invent-

ing informative natural names into my programs. I found out that when I spent more time inventing the names, I understood everything well, I made less programming errors, and I produced better software. The programs that I wrote started to look like carefully written documents which can be read by other people. Then, on the other hand, programs containing abbreviations started to look like documents that have been written without any decent writing rules.

I became convinced that people use abbreviations in their programs because they have learned only this one way of programming. I also used abbreviations in my programs before September 1988. So, to change the habits from abbreviating to the use of natural naming, people should have an opportunity to study with material where abbreviations are not used. That is the reason why I wrote this book. The program examples of this book do not contain abbreviations in the names of variables, arrays, objects, functions, etc. Therefore, people do not learn to use them.

To be able to write this book, I had to gain teaching experience, and I became, at least for the time being, a full-time teacher. In my programming courses I have used only program examples that contain natural names.

For teaching purposes and for this book, I have written many programs. This work has served to intensify my conviction that there is something very powerful in the use of elegant natural names in source programs. Through the use of this naming style I understand the syntax of the programming language better, programs become more robust, and I make less programming errors. Also in the case of larger programs, such as **editor1.cpp** which has 3700 source program lines, I have noticed that I make fewer design and programming errors, and even compilation errors are reduced.

I have improved as a computer programmer and software designer because I spend time thinking about appropriate natural names, and by doing this I am able to find a working structure for the programs. I have seen it many times that people write their programs too quickly without thinking enough. Then they have all kinds of errors in their programs, and they start searching the errors with a debugging tool. Errors in programs are errors in thinking, and many of the errors could be prevented through careful design and programming. The use of natural naming is a means to think and to write programs carefully. I'm not totally against the use of debugging tools, the tools for searching for "bugs" in programs, but their usage can be tremendously reduced by simply thinking a little bit more during the writing of programs.

I am not the only person who has discovered the benefits of natural naming in programming and software development. I already mentioned Mr. Daniel Keller who has published similar ideas to me. I have met some people in Finnish companies who use natural names in their programs, and insist that everything in computer programs has to be explained with names. Probably there are many software developers in the world who exploit the idea of natural naming. The names in textbooks of computer programming have become longer during the 90s. So it is a general tendency to use longer names in programming, but I still believe that in no other book have long natural names been used to such an extent as in this book.

The use of natural names is also a tradition in software design methods such as Structured Analysis and Design (SA/SD) and Object-Oriented Analysis and Design (OOA/OOD). In these methods, the use of natural names helps the designers to think and design. According to my experience, programming becomes more like a designing activity when the person who creates programs spends time to invent informative natural names.

I have written some articles (see my Internet homepage) in which I express the opinion that software development is partially a linguistic process during which the software developers also develop a language with which they communicate. Naming of the concepts of the application domain is one activity in such a linguistic process. Therefore, naming is important "work" that software developers have to do. Some software developers do this work more seriously than others. I hope that this book will eventually increase the number of "serious namers".