Epilogue, In Full Frankness Exposing the Language

Enthusiastically setting out to solve some of the most pressing problems of software engineering, this book has developed an ambitious method for developing quality systems. Since no method is possible without a supporting notation, we have had to devise, as we learned the various components of object-oriented software construction, what in the end turned out to be a complete lifecycle language for software analysis, specification, design, implementation, maintenance and documentation.

Instead of reading on page one, however, the name of the language that you would be using, you have been invited to participate with the author in developing the notation, chapter after chapter, notion after notion, construct after construct. And until now that language has remained nameless. Why? The reasons were sketched in the preface but may deserve some final elaboration.

First, I hope that even though you were warned that the notation already exists and is extensively documented in tens of published textbooks, hundreds of articles and thousands of Usenet messages, you earnestly accepted the pedagogical convention that you participated in its design as you were reading this book. Although this has made life a little harder for the author — imagine: having to *justify* every single construct, instead of bringing it to the people down from the top of Mount Sinai — the effort will have been worthwhile if it has succeeded in giving you a better understanding not only of what things are but of why they must be that way. Second, the convention has enabled us to concentrate on the method, not on notational details, making this book useful not just to people who will indeed have access to the language through one of the supporting commercial environments, but also to those readers who are required to use less complete O-O languages such as Smalltalk, C++, Ada 95, Java or Object Pascal, or even a non-O-O one such as C, Fortran, Pascal or Ada, to which one can apply the emulation techniques discussed in earlier chapters.

Fiction or not, the mystery is not very hard to penetrate. Even if you have not looked at the back cover of this book or read other works by the same author (including the complete language description [M 1992]), just a cursory glance at some of the bibliographical references will have revealed all there is to reveal. And books such as this one are meant to be not only read but re-read, so the surprise, if any, will not last long.

Even so, keeping the language name away from the discussion (except for a few hints that the alert reader may have noted) has enabled us to concentrate on the method. This is a little paradoxical, since one of the language's principal claims is that, alone among O-O languages, it is *also* a method, avoiding the gap between concept and expression, between analysis and design, between design and implementation, which plagues common O-O approaches and threatens to defeat some of the principal advantages of object technology. Not even the brightest-eyed Java or Smalltalk enthusiast will allege that his language of choice is a general-purpose tool for design, let alone analysis; and users of popular analysis notations such as OMT know that they must move to something else when it comes to producing the actual software. The ambition of the method-notation developed in this book is higher: to fulfill one of the main premises and promises of object technology, *seamlessness*, by serving as a faithful assistant that will accompany you throughout the software construction process.

Literary conventions have an end, so the time has now come, at the close of our extended tour of the beauties of object-oriented software construction, after thanking the reader for patiently going along, through all these pages, with the pedagogical pretense of an anonymous language, to lift the very thin veil that covered the name of our notation: welcome to the world of Eiffel.