class, which had nothing to do with a C++ class. With the new frameworks, you needed only to create a subclass of a class provided by the framework, and that was that. We were happy to be suddenly freed from all the drudgery (or almost all), and also happy that we suddenly found such a neat application of object orientation.

Programming Microsoft Windows is just an example; the enthusiasm with object orientation and inheritance was pervasive. It is strange to learn now that we may have been wrong, but perhaps we were not that wrong. Inheritance may not be essentially bad. Like all technology, it can be put to good and bad uses, and the bad uses of inheritance have been documented in many places (the Gang of Four book being a good start). Here we will give an example of a beautifully crafted software system that has inheritance as its foundation. That system is Smalltalk.

Smalltalk is a pure object-oriented language, and although it never actually made it to the mainstream, it influenced language evolution in many ways. Perhaps the other language that exerted so much influence on later computer languages was Algol-60—another example that was more influential than actually used.

This is not a presentation of the Smalltalk programming language and its environment (these two really go together), but a presentation of basic architectural ideas and where they may lead us in our programming tasks. To borrow a term from the psychology of design, this is a discussion on basic design principles and the *affordances* they give the programmer. Donald Norman, in *The Psychology of Everyday Things* (1988), lucidly (and entertainingly) explains the notion of affordances. Simply put, an object by its appearance allows us, and even invites us, to do certain things. A hanging rope invites us to reach for it and pull it; a horizontal handle bar invites us to push it; and a door-handle invites us to reach for it and turn it. In the same way, the way a programming language looks to the programmer invites her to do certain things with it, and warns her against doing other things. A beautiful crafted language has a beautiful architecture, and that will show through the programs we make with it.

A strong expression of this is the Shapir-Whorf Hypothesis (SWH), which states that language determines thought. The hypothesis has excited linguists and language designers for many years. The preface to the first edition of *The C++ Programming Language* starts with SWH, and the 1980 Turing Award lecture by K. E. Iverson (of APL fame) was devoted to the importance of notation for expressing our thoughts. The SWH is controversial—after all, everybody has had the experience of not being able to find the words for something, so we are able to think of more than we can say—but in computer code the link between languages and programs is clear. We know that computer languages are Turing complete, but we also know that for some things some languages fit better than others.

But apart from influencing program architecture, a language's architecture is interesting in its own right. We will take a glimpse at Smalltalk's own architecture—implementation choices, design concepts, and patterns. We see many of them today in more recent programming