Object-Oriented Software Construction, 2nd Ed.
Bertrand Meyer
Prentice-Hall, 1997
pp. v - vii

Preface

Born in the ice-blue waters of the festooned Norwegian coast; amplified (by an aberration of world currents, for which marine geographers have yet to find a suitable explanation) along the much grayer range of the Californian Pacific; viewed by some as a typhoon, by some as a tsunami, and by some as a storm in a teacup — a tidal wave is hitting the shores of the computing world.

"Object-oriented" is the latest in term, complementing and in many cases replacing "structured" as the high-tech version of "good". As is inevitable in such a case, the term is used by different people with different meanings; just as inevitable is the well-known three-step sequence of reactions that meets the introduction of a new methodological principle: (1) "it's trivial"; (2) "it cannot work"; (3) "that's how I did it all along anyway". (The order may vary.)

Let us have this clear right away, lest the reader think the author takes a half-hearted approach to his topic: I do not see the object-oriented method as a mere fad; I think it is not trivial (although I shall strive to make it as limpid as I can); I know it works; and I believe it is not only different from but even, to a certain extent, incompatible with the techniques that most people still use today — including some of the principles taught in many software engineering textbooks. I further believe that object technology holds the potential for fundamental changes in the software industry, and that it is here to stay. Finally, I hope that as the reader progresses through these pages, he will share some of my excitement about this promising avenue to software analysis, design and implementation.

"Avenue to software analysis, design and implementation". To present the object-oriented method, this books resolutely takes the viewpoint of software engineering — of the methods, tools and techniques for developing quality software in production environments. This is not the only possible perspective, as there has also been interest in applying object-oriented principles to such areas as exploratory programming and artificial intelligence. Although the presentation does not exclude these applications, they are not its main emphasis. Our principal goal in this discussion is to study how practicing software developers, in industrial as well as academic environments, can use object technology to improve (in some cases dramatically) the quality of the software they produce.

Structure, reliability, epistemology and classification

Object technology is at its core the combination of four ideas: a structuring method, a reliability discipline, an epistemological principle and a classification technique.

The structuring method applies to software decomposition and reuse. Software systems perform certain actions on objects of certain types; to obtain flexible and reusable systems, it is better to base their structure on the object types than on the actions. The resulting concept is a remarkably powerful and versatile mechanism called the class, which in object-oriented software construction serves as the basis for both the modular structure and the type system.

The reliability discipline is a radical approach to the problem of building software that does what it is supposed to do. The idea is to treat any system as a collection of components which collaborate the way successful businesses do: by adhering to contracts defining explicitly the obligations and benefits incumbent on each party.

The epistemological principle addresses the question of how we should describe the classes. In object technology, the objects described by a class are only defined by what we can do with them: operations (also known as features) and formal properties of these operations (the contracts). This idea is formally expressed by the theory of abstract data types , covered in detail in a chapter of this book. It has far-reaching implications, some going beyond software, and explains why we must not stop at the naïve concept of "object" borrowed from the ordinary meaning of that word. The tradition of information systems modeling usually assumes an "external reality" that predates any program using it; for the object-oriented developer, such a notion is meaningless, as the reality does not exist independently of what you want to do with it. (More precisely whether it exists or not is an irrelevant question, as we only know what we can use, and what we know of something is defined entirely by how we can use it.)

Abstract data types are discussed in chapter 6, which also addresses some of the related epistemological issues.

The classification technique follows from the observation that systematic intellectual work in general and scientific reasoning in particular require devising taxonomies for the domains being studied. Software is no exception, and the object-oriented method relies heavily on a classification discipline known as inheritance .

Simple but powerful

The four concepts of class, contract, abstract data type and inheritance immediately raise a number of questions. How do we find and describe classes? How should our programs manipulate classes and the corresponding objects (the instances of these classes)? What are the possible relations between classes? How can we capitalize on the commonalities that may exist between various classes? How do these ideas relate to such key software engineering concerns as extendibility, ease of use and efficiency?

Answers to these questions rely on a small but powerful array of techniques for producing reusable, extendible and reliable software: polymorphism and dynamic binding; a new view of types and type checking; genericity, constrained and

unconstrained; information hiding; assertions; safe exception handling; automatic garbage collection. Efficient implementation techniques have been developed which permit applying these ideas successfully to both small and large projects under the tight constraints of commercial software development. Object-oriented techniques have also had a considerable impact on user interfaces and development environments, making it possible to produce much better interactive systems than was possible before. All these important ideas will be studied in detail, so as to equip the reader with tools that are immediately applicable to a wide range of problems.

A modest soul is shocked by objects of such kind

And all the nasty thoughts that they bring to one's mind.

Molière, Tartuffe, Act III.