Contents

Preface

Why This Book?

During the first four month of 1994 I was presented with a wonderful opportunity. My old University in Wroclaw, Poland, invited me to give two courses for the students of Computer Physics. The choice of topics was left entirely to my discretion. I knew exactly what I wanted to teach...

My work at Microsoft gave me the unique experience of working on large software projects and applying and developing state of the art design and programming methodologies. Of course, there are plenty of books on the market that talk about design, programming paradigms, languages, etc. Unfortunately most of them are either written in a dry academic style and are quite obsolete, or they are hastily put together to catch the latest vogue. There is a glut of books teaching programming in C, C++ and, more recently, in Java. They teach the language, all right, but rarely do they teach programming.

We have to realize that we are witnessing an unprecedented explosion of new hardware and software technologies. For the last twenty years the power of computers grew exponentially, almost doubling every year. Our software experience should follow this exponential curve as well. Where does this leave books that were written ten or twenty years ago? And who has time to write new books? The academics? The home programmers? The conference crowd? What about people who are active full time, designing and implementing state of the art software? They have no time!

In fact I could only dream about writing this book while working full time at Microsoft. I had problems finding time to share experiences with other teams working on the same project. We were all too busy writing software. And then I managed to get a four-month leave of absence. This is how this book started.

Teaching courses to a live, demanding audience is the best way of systematizing and testing ideas and making fast progress writing a book. The goal I put forward for the courses was to prepare the students for jobs in the industry. In particular, I asked myself the question: If I wanted to hire a new programmer, what would I like him to know to become a productive member of my team as quickly as possible?

For sure, I would like such a person to know

- C++ and object oriented programming.
- Top-down design and top-down implementation techniques.
- Effective programming with templates and C++ exceptions.
- Team work.

He (and whenever I use the pronoun *he*, I mean it as an abbreviation for *he or she*) should be able to write reliable and maintainable code, easy to understand by other members of the team. The person should know advanced

programming techniques such as synchronization in a multithreaded environment, effective use of virtual memory, debugging techniques, etc.

Unfortunately, most college graduates are never taught this kind of "industrial strength" programming. Some universities are known to produce first class computer hackers (and seem to be proud of it!). What's worse, a lot of experienced programmers have large holes in that area of their education. They don't know C++, they use C-style programming in C++, they skip the design stage, they implement bottom-up, they hate C++ exceptions, and they don't work with the team. The bottom line is this: they waste a lot of their own time and they waste a lot of others' time. They produce buggy code that's difficult to maintain.

So who are you, the reader of this book? You might be a beginner who wants to learn C++. You might be a student who wants to supplement his or college education. You might be a new programmer who is trying to make a transition from the academic to the industrial environment. Or you might be a seasoned programmer in search of new ideas. This book should satisfy you no matter what category you find yourself in.