Foreword

In *Federalist 51*, James Madison wrote: "If men were angels, no government would be necessary." If he lived today, Madison might have written: "If software developers were angels, debugging would be unnecessary." Most of us, however, make mistakes, and many of us even make errors while designing and writing software. Our mistakes need to be found and fixed, an activity called debugging that originated with the first computer programs. Today every computer program written is also debugged, but debugging is not a widely studied or taught skill. Few books, beyond this one, present a systematic approach to finding and fixing programming errors.

Be honest: Does debugging seem as important, difficult, or worthy of study as writing a program in the first place? Or, is it just one of those things that you need to do to finish a project? Software developers though spend huge amounts of time debugging—estimates range up to half or more of their workdays. Finding and fixing bugs faster and more effectively directly increases productivity and can improve program quality by eliminating more defects with available resources. Preventing mistakes in the first place would be even better, but no one has yet found the technique to prevent errors, so effective debugging will remain essential.

Improved programming languages and tools can supplant, but not eliminate, debugging, by statically identifying errors and by dynamically detecting invariant violations. For example, the type system in modern languages such as Java and C# prevents many simple mistakes that slip by C programmers. Moreover, these languages' runtime bounds checks stop a program when it strays out of bounds, which may be billions of instructions before the error manifests itself. Unfortunately, there are countless ways in which a program can go wrong, almost all of which languages and tools cannot detect or prevent. For example, in recent years there has been considerable work in verifying sequences of operations in a program. Tools can ensure that a file is opened before a program reads it, but they cannot check that the correct file is accessed or that the program properly interprets its contents. If either mistake occurs, someone still must debug the program to understand the error and determine how to fix it.

In addition, debugging can be an enjoyable activity that shares the thrill of the hunt and chase found in a good detective novel or video game. On the other hand, a protracted, unsuccessful search for a bug in your code quickly loses its charm, particularly when your boss is asking repeatedly about your (lack of) progress. Learning to debug well is essential to enjoying software development.

This book can teach you how to debug more effectively. It is a complete and pragmatic overview of debugging, written by a talented researcher who has developed many clever ways to isolate bugs. It explains best practices for finding and fixing errors in programs, ranging from systematically tracking error reports, reproducing failures, observing symptoms, isolating the cause, and correcting defects.

Along with basic techniques and commonly used tools, the book also explores the author's innovative techniques for isolating minimal input to reproduce an error and for tracking cause and effect through a program.

Studying this book will make you a better programmer. You will be able to find and fix errors in your code (and your colleague's code) faster and more effectively, a valuable skill that will enable you to finish projects earlier and produce programs with fewer defects. Also, if you read between the lines you will learn how to write code that is more easily tested and debugged, which further increases your ability to find and correct defects. And thinking hard about what can go wrong with your program can help you avoid mistakes in the first place, so you have less to debug.

James Larus
Microsoft Research