Texts and Monographs in Computer Science

## Texts and Monographs in Computer Science

Suad Alagić
**Object-Oriented Database Programming**
1989. XV, 320 pages, 84 illus.

Suad Alagić
**Relational Database Technology**
1986. XI, 259 pages, 114 illus.

Suad Alagić and Michael A. Arbib
**The Design of Well-Structured and Correct Programs**
1978. X, 292 pages, 68 illus.

S. Thomas Alexander
**Adaptive Signal Processing: Theory and Applications**
1986. IX, 179 pages, 42 illus.

Michael A. Arbib, A.J. Kfoury, and Robert N. Moll
**A Basis for Theoretical Computer Science**
1981. VIII, 220 pages, 49 illus.

Friedrich L. Bauer and Hans Wössner
**Algorithmic Language and Program Development**
1982. XVI, 497 pages, 109 illus.

Kaare Christian
**A Guide to Modula-2**
1986. XIX, 436 pages, 46 illus.

Edsger W. Dijkstra
**Selected Writings on Computing: A Personal Perspective**
1982. XVII, 362 pages, 13 illus.

Edsger W. Dijkstra and Carel S. Scholten
**Predicate Calculus and Program Semantics**
1990. XII, 220 pages

W.H.J. Feijen, A.J.M. van Gasteren, D. Gries, and J. Misra, Eds.
**Beauty Is Our Business: A Birthday Salute to Edsger W. Dijkstra**
1990. XX, 453 pages, 21 illus.

Melvin Fitting
**First-Order Logic and Automated Theorem Proving**
1990. XIV, 242 pages, 26 illus.

Nissim Francez
**Fairness**
1986. XIII, 295 pages, 147 illus.

*continued after index*

# Programming with Specifications

An Introduction to ANNA, A Language for Specifying Ada Programs

**David Luckham**

Springer-Verlag
New York Berlin Heidelberg London
Paris Tokyo Hong Kong Barcelona

David Luckham
Computer Science Laboartory
Stanford University
Stanford, CA 94305-4055
USA

*Series Editor*

David Gries
Department of Computer Science
Cornell University
Ithaca, NY 14853
USA

Printed on acid-free paper.


Softcover reprint of the hardcover 1st edition 1990



Photocomposed copy prepared by the author using the author's LaTeX file.

9 8 7 6 5 4 3 2 1

ISBN-13:978-1-4613-9687-1 e-ISBN-13:978-1-4613-9685-7
DOI: 10.1007/978-1-4613-9685-7

*These pages are dedicated*
*to those who sat by*
*and watched them grow:*
*Susannah and Brian*

# Preface

**Topics**

- *what this book is about,*
- *its intended audience,*
- *what the reader ought to know,*
- *how the book is organized,*
- *acknowledgements.*

Specifications express information about a program that is not normally part of the program, and often cannot be expressed in a programming language. In the past, the word "specification" has sometimes been used to refer to somewhat vague documentation written in English. But today it indicates a precise statement, written in a machine processable language, about the purpose and behavior of a program. Specifications are written in languages that are just as precise as programming languages, but have additional capabilities that increase their power of expression. The terminology *formal specification* is sometimes used to emphasize the modern meaning. For us, all specifications are formal.

The use of specifications as an integral part of a program opens up a whole new area of programming — *programming with specifications*. This book describes how to use specifications in the process of building programs, debugging them, and interfacing them with other programs. It deals with a new trend in programming — the evolution of specification languages from the current generation of programming languages. And it describes new strategies and styles of programming that utilize specifications. The trend is just beginning, and the reader, having finished this book, will

certainly see that there is much yet to be done and to be discovered about programming with specifications.

This is a book for people who have attained some experience with programming languages and have already written some significant programs. Better yet, the reader should have tried to understand or modify someone else's programs. Such readers will have concluded from their own experience that the current methods of programming have to become more disciplined. Now they are ready to explore programming with specifications.

In writing the book, I have had in mind primarily two groups of people: professional software engineers and college undergraduates and graduates taking courses in computer science or software engineering. But in this age of the home computer, people who have the sort of experience I have just alluded to can come from almost any age group and many different backgrounds.

What precisely should the reader know already? There are two prerequisites, which I will describe by telling you just a little about the book.

The book deals with the use of specifications to develop Ada programs. Specifications are written in a formal language called Anna. Anna is a specification language. It is no harder to learn than programming languages such as Pascal or Modula2 or Ada. In fact, Anna is an extension of the Ada language. It allows annotations to be included as part of an Ada program. Annotations can be processed by tools "like" compilers, and by very different kinds of tools as well. Anna stands for "ANNotated Ada."

Ideally, the reader should already know Ada. This is the first prerequisite, but it is not absolutely necessary. A reader who knows Modula2 or more advanced dialects of Pascal, or C++, can use this book as a way of simultaneously learning Ada as well as Anna. This is possible because the concepts and methods of programming with specifications are independent of any particular programming language. They apply equally well to any language containing constructs similar to Ada. Anna could just as easily have been based on Modula2, for example.

Why choose to study programming with specifications in the context of Ada — or any particular programming language? My answer is this. In order to develop new ways of programming that are really practical, it is absolutely essential to deal with the real problems that are faced by the real programmer in the use of a real programming language. And since Ada is certainly the most ambitious Algol-like language of the time, it is a logical choice upon which to base the development of new programming methods. Of course, some of the details involved in specifying programs written in any of today's programming languages are quite messy. Indeed, these messy details have an annoying way of complicating methods that are really very simple. But, if we are successful in developing new methods of programming, the ultimate consequence will be the evolution of more advanced languages that make those methods easy to apply. By exposing the messy details, they will eventually disappear!

The second prerequisite is a little background in the theory of computer science — not a lot, just a little. This involves three things that are normally part of an undergraduate curriculum: (1) basic data structures (lists, trees, sets, stacks, and queues); (2) formal logic, usually called Propositional and Predicate Logic (you should know about Boolean operators, what a quantifier is, and what a formal proof looks like); and (3) an undergraduate course in abstract algebra (a knowledge of axioms for linear ordering, groups, a little of that sort of thing).

Anna and methods of programming with specifications are presented informally. They are described in much the same way as most books describe programming languages or algorithms and data structures. The idea is not to demand a lot of background from the reader. So the book is really an experiment to see if the use of specifications in programming can be taught just like the use of advanced programming constructs and structures are taught now. As readers progress in the methods of writing and using specifications, they may become interested in exploring the foundational theories. These can be found in other books on the mathematical semantics of specifications and programs, and on axiomatic proof systems.

This book does two things: it explains Anna and it describes possible ways of using it. The book alternates between explaining Anna constructs and giving examples describing methods of programming with specifications. I have used four devices to help this alternation: (1) **commentaries** on examples, (2) **guidelines** on constructing specifications, (3) **recipes** for describing methods of applying specifications, and (4) the star (*).

Examples nearly always include a **commentary** that encapsulates various details of methodology. Our examples aren't perfect either; their imperfections are used to illustrate the compromises and choices one may face in the practical world of imperfect languages and too little time. Commentaries also include a lot of the details that are specific to Ada. Those interested in a general overview of Anna can skip commentaries, but I don't advise it.

**Guidelines** are common-sense rules of thumb about how to construct specifications, and what kinds of information to express in them. They are prominently displayed at various points in the discussion of applications of specifications.

**Recipes** really are cookery. Sometimes I want to describe an algorithm for applying specifications that really is too complicated for humans to do in general. I give a rather vague outline called a recipe. Recipes give the reader a taste of the method as it applies to simple examples. Good cooks should be able to reconstruct a complete algorithm, with variations to taste. Future environments will contain tools that automate such algorithms. So eventually, users will need to know only *what* a recipe produces, and not *how* to cook it.

Some parts of the book are hard to read. They contain complicated formulas, or go into messy details. These sections and chapters are **starred**

(*). Starred sections can be passed over on a first reading.

It is important to read this book in conjunction with other books and research papers as well. I have included reading lists at the end of some chapters. The lists are short — to encourage the reader. They provide an overview of some of the prior work upon which programming with specifications is based, and also how it fits in with other current work and perceived problems in the software area.

The structure of the book is as follows.

Chapter 0 describes how modern programming languages are gradually evolving into specification languages and why this is happening. It gives a short general description of applications of formal specifications to the programming process.

Chapters 1, 2, and 7 deal with annotations of the Pascal-like subset of Ada — called *simple annotations*. A description of simple annotations is in Chapter 1. The basic concepts needed to define the meaning of formal annotations and the correctness of programs are introduced and discussed as the need for them becomes obvious. This way, a lengthy preliminary chapter on formal semantics and correctness is avoided. Applications of simple annotations are described in Chapter 2. More advanced annotations for composite data structures, together with examples of applications are in Chapter 7.

Chapter 3 describes annotations for programs with exceptions and a method of specifying exceptional behavior in Ada programs.

Chapters 4 and 5 deal with specification of packages. These chapters are the heart of programming with specifications. Chapter 4 explains the annotation constructs for package specifications. Chapter 5 describes methods of specifying packages and analyzing the consequences of package specifications. The methods and examples given here are only a small introduction to the science of building formal specifications for software packages. This is an area where more powerful methods and languages need to be developed. Many topics, such as incompleteness of specifications, are only mentioned in passing. These two chapters could easily have taken the entire book.

Chapter 6 describes annotation of Ada generic units and how such annotations are relevant to building reusable software.

Chapters 8, 9, and 10 are devoted to annotation and construction of package bodies. The crucial problem is construction of a package body that is consistent with a given package specification. There are three parts to this problem, each part being assigned a separate chapter. Chapter 8 explains package body annotations. Chapter 9 illustrates ways to analyze whether the body, as it is being implemented, is (or will be) consistent with the original specification. This chapter is starred since it involves rather lengthy annotations. Chapter 10 describes new methods of utilizing a package specification as a guide in implementing a body. These methods are examples of *rigorous* software development methods. They integrate techniques such as runtime checking of specifications into the process of

building packages so that implementation errors are caught as early as possible and certain kinds of errors never happen.

As mentioned earlier, this book is an introduction to programming with specifications. This area is just emerging. The use of specifications is a logical development from recent trends in programming languages, and it is being explored as an approach to dealing with increasing problems in software production. There is much still to be done. I hope the reader will be encouraged to improve upon what is in these pages, and if so, I shall consider the book a success.

## Acknowledgements

The research leading to the development of Anna and its support tools has been sponsored by the Defense Advanced Research Projects Agency. I am indebted to DARPA for the opportunity to do this work in the first place.

Many patient people have read various versions of this book during its evolution, and their comments have been helpful and influential. Some of them were good enough to review more than one version! Particularly, I am indebted to past and present members of the Program Analysis and Verification Group at Stanford who have reviewed the book and developed the Anna toolset: Doug Bryan, John Kenny, Neel Madhav, Walter Mann, Geoff Mendal, Randy Neff, Wolfgang Polak, David Rosenblum, Sriram Sankar, Will Tracz, and Friedrich von Henke. In addition, it gives me great pleasure to thank David Gries for a detailed review beyond the duty of any editor, and Jennifer Anderson, David Guaspari, and Norman Ramsey for detailed comments. And there are many others to whom my thanks are also due, especially the students in my *"Topics in Ada Programming"* courses at Stanford. The book is much better as a result of everyone's efforts to help me, but of course I'm responsible for whatever is wrong with it.

Certainly, the book would not exist without Rosemary Brock, who of all patient people has been the most patient, dealing with many versions in Scribe and TEX and LATEX over several years. So, Rosemary, thank you too.

D. C. L.
Palo Alto
1989

# Contents