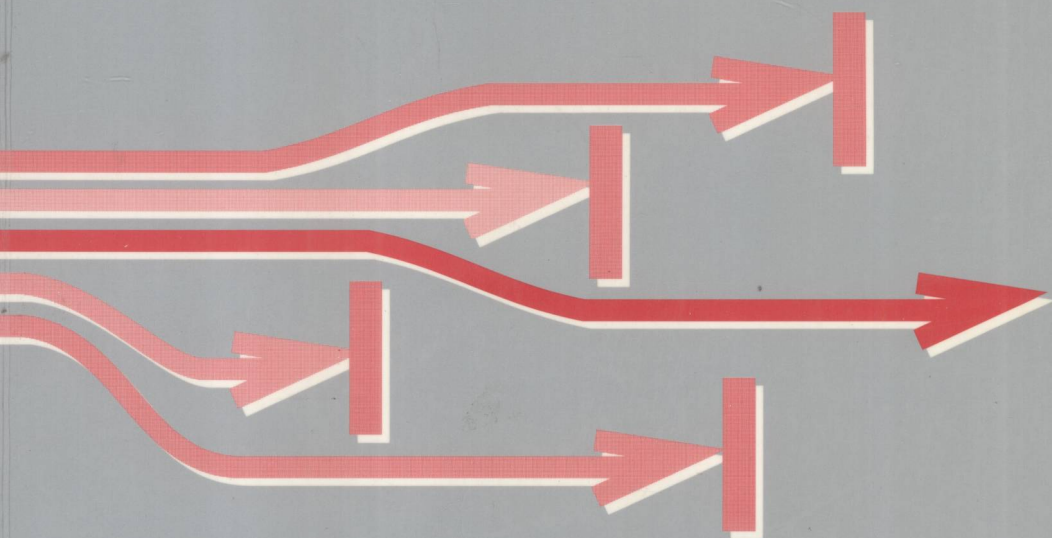


Derek Andrews

A Theory and Practice of Program Development



Springer

FACIT

TP31
A565

9860579

Derek Andrews

A Theory and Practice of Program Development



E9860579



Springer

Derek Andrews, BSc, MSc
Department of Mathematics and Computer Science,
Leicester University, University Road, Leicester LE1 7RH, UK

Series Editor

S.A. Schuman, BSc, DEA, CEng
Department of Mathematical and Computing Sciences
University of Surrey, Guildford, Surrey GU2 5XH, UK

ISBN 3-540-76162-4 Springer-Verlag Berlin Heidelberg New York

British Library Cataloguing in Publication Data
Andrews, Derek

A theory and practice of program development. - (Formal
approaches to computing and information technology)

1.Computer software - Development

I.Title

005.1

ISBN 3540761624

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

Apart from any fair dealing for the purposes of research or private study, or criticism or review, as permitted under the Copyright, Designs and Patents Act 1988, this publication may only be reproduced, stored or transmitted, in any form or by any means, with the prior permission in writing of the publishers, or in the case of reprographic reproduction in accordance with the terms of licences issued by the Copyright Licensing Agency. Enquiries concerning reproduction outside those terms should be sent to the publishers.

© Springer-Verlag London Limited 1997
Printed in Great Britain

The use of registered names, trademarks etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant laws and regulations and therefore free for general use.

The publisher makes no representation, express or implied, with regard to the accuracy of the information contained in this book and cannot accept any legal responsibility or liability for any errors or omissions that may be made.

Typesetting: Camera ready by editors
Printed and bound at the Athenæum Press Ltd., Gateshead, Tyne and Wear
34/3830-543210 Printed on acid-free paper

Formal Approaches to Computing and Information Technology

Springer

London

Berlin

Heidelberg

New York

Barcelona

Budapest

Hong Kong

Milan

Paris

Santa Clara

Singapore

Tokyo

Also in this series:

Proof in VDM: a Practitioner's Guide

J.C. Bicarregui, J.S. Fitzgerald, P.A. Lindsay, R. Moore
and B. Ritchie

ISBN 3-540-19813-X

Systems, Models and Measures

A. Kaposi and M. Myers

ISBN 3-540-19753-2

Notations for Software Design

L.M.G. Feijs, H.B.M. Jonkers and C.A. Middelburg

ISBN 3-540-19902-0

Formal Object-Oriented Development

K. Lano

ISBN 3-540-19978-0

The B Language and Method

K. Lano

ISBN 3-540-76033-4

Formal Methods and Object Technology

S.J. Goldsack and S.J.H. Kent

ISBN 3-540-19977-2

Preface

All software development methods are based on the same basic idea: to provide a way of producing good software from a set of system requirements. A good software development method should provide a notation that can be used during the analysis of the system requirements to produce an abstract model (a specification) that captures the essence of the system and that can be used as the basis of the development. A good development method should try to encourage the specification of what the system does at the early stages of design and provides a series of strategies and transformations (usually described informally) on how to turn the specification into executable code. What is frequently missing from many development methods is a formal (in the mathematical sense) background and justification to both the notation and the transformations. A formal approach can provide an insight into the analysis, specification, design and development of computer systems.

An approach to software development is to focus on the separation of the abstract and concrete views of a system. A good development method should emphasize:

- abstract vs concrete;
- logical vs physical; and
- ‘what’ vs ‘how’.

The abstract view should be easily understood by the system architect and should concentrate on describing what the system does. The concrete view of the system is a program that describes how the system works at a level of detail can be efficiently executed by a computer. Design can be seen as the bridge between these two views of a system. This bridge should have mathematical foundations.

The book provides a weakest pre-condition semantics for almost all of VDM-SL and uses the semantics to derive a software development method and to give the method a rigorous justification. The strategies and transformations that are used are both justified and explained mathematically.

The language described in the book is based on VDM-SL (perhaps it is closer to Meta-IV on which VDM-SL is based). It has been updated with a more modern syntax: it is a subset of VDM-SL with the following restrictions:

- no loose functions;

- restrictions on patterns (including no loose patterns); and
- parameters to functions and procedures must be read only.

Other changes are to the statement part of the language. The main change is the introduction of Dijkstra's guarded commands for the executable part of the language. This has been one for the following reasons:

1. guarded commands are used as one of the basic constructs of the language;
2. they are elegant; and
3. an excellent idea should be republicized!

Simple input and output statements have been introduced so that the executable language could be implemented and used. Some other minor changes to the syntax have been made to make programs more readable. The most important change is the introduction of a specification command for refinement purposes. The author should apologize for these changes, but the advantages of Dijkstra's guarded commands outweigh the need to conform to the ISO Standard for VDM-SL.

The semantics are used to derive appropriate program and data refinement rules. It is easier for a developer attempting a refinement, or proving properties of a specification or a program to derive new, useful, rules using weakest pre-conditions. In particular it has been found that proofs of properties of language constructs can be more easily discharged with this approach.

The book shows that it is now possible to derive both a specification language and development method based on a few simple concepts, and to give a formal explanation of all of the development steps used in producing code from a specification. The book introduces the idea that the purpose of analysis is to provide a specification that can easily be understood by the human mind before the expensive process of developing a program starts. Specifications are 'programs' that should be biased towards a style that allows efficient execution by an intellectual computer, and that programs are 'specifications' that can be executed efficiently by an electronic computer. Design is the process of transforming a specification into efficient code.

The approach provides the formal semantics of a simple, but powerful, wide-spectrum programming language (VDM-WSL) and gives a formal definition of both algorithmic and data refinement. The following topics are covered in detail using a formal approach:

- formal specification;
- proving properties of a specification;
- properties of programs;
- algorithmic refinement;
- data refinement; and
- the correctness of code.

The theory of algorithm and data refinement also provides insight into the design process and development strategies that can be used to produce efficient code. Issues such as why reusable code is difficult to achieve have also been addressed. A mathematical foundation and background to most, if not all, of the ideas of software specification, design and development are given.

A course developed at Leicester over a ten-year period now uses the approach put forward in this book. The course was initially based on the refinement style of C. B. Jones from [23]; it slowly evolved to the current style. In a third-year course at Leicester that uses the ideas in the book, students have carried out the specification and refinement of some small systems, for example a system to manage a book library and a car-hire system. Other examples that have been developed were an analysis of sorting algorithms and the correct implementation of both AVL and 2-3 trees. With teaching in mind, many of the easier lemmas, theorems, and corollaries have been left for the reader to complete; these are terminated with a hollow box.

The book would be suitable for a third- or fourth-year undergraduate or a post-graduate course in formal methods – with this in mind, a complete set of teaching material is available. The material consists of:

1. a complete set of over-head projector slides together with a set of teaching notes;
2. a student study guide consisting of weekly reading suggestions and exercises;
3. a set of worked examples together with model answers, these would be suitable to add to the study guide or for use as examination questions; and
4. a technical report that describes the kernel language, the full language and an executable subset.

This material is available in machine readable form (in \LaTeX source) on the World-Wide-Web – contact the author at derek@mcs.le.ac.uk.

Acknowledgements

Weakest pre-conditions were used by Hehner and others to give the semantics of a (subset) of Pascal and by Dijkstra to give the semantics of his guarded command language [20]. The idea of giving the semantics of VDM-SL with relations and a termination condition was used by Cliff Jones in his Ph.D. thesis [24]. This approach extends naturally when using the weakest pre-conditions to give the semantics of the kernel language used in this book. In a paper by Nelson [30] that gave a slightly different semantics to Dijkstra's guarded command language: undefined (\perp) was added as a possible outcome of a command. In [3] Abrial defined the semantics of B in terms of a set of

basic commands. The formal semantics of VDM-SL have been given using relations ([18]) and based on this work, proof rules for VDM-SL have been given by Peter Gorm Larsen in his Ph.D. thesis [25]. Work at Oxford ([27, 26]) and work by Morris [28, 29] introduced the ideas of a programming calculus that dealt with both program and data refinement. Based on the Oxford work, in a series of examples, Andrews and Ince mixed program and data refinement to develop executable code from a specification [11, 10, 9]. This book pulls these ideas together to give a weakest pre-condition semantics for a close relative of VDM-SL, and uses these semantics to derive a set of refinement rules.

The author would also like to thank Professors Dr. H Klaeren and S. Schuman and an anonymous referee for their comments on an earlier draft and Rosie Kemp and the staff at Springer-Verlag for their patience and support.

Chapter Summaries

Chapter 1 – Writing Correct Programs Why refinement, the same simple example treated two different ways: starting with a specification, writing the code and then showing it to be correct and secondly deriving the correct code from the specification by a series of transformations – software development as a strategy game.

Chapter 2 – A Small Programming Language A small language for writing both specifications and programs. The semantics of the language are given using both relations and the weakest pre-condition approach. The concept of testing is discussed, leading to a meaning of program correctness.

Chapter 3 – Concepts and Properties The basic properties of the kernel language: simple program transformations. Setting and testing variables. The basic properties of a theory of refinement. A normal form and a framework for proving properties.

Chapter 4 – Building New Commands from Old Extending the language, principles and theory. Recursive definitions. (This is seen as a chapter for the advanced reader, the remainder of the book does not depend on it.)

Chapter 5 – Program Refinement Stepwise refinement. The concept of refinement. Replacing specifications: the meaning of program correctness and program refinement. Three different ways of looking at correctness and their equivalence. What is testing. Other refinement models.

Chapter 6 – The Basic Commands Refining the basic statements. Implementation issues; the language and the compiler writer.

Chapter 7 – Declarations and Blocks Introducing local variables, the idea of scope. Blocks and the refinement rules.

Chapter 8 – Command Sequences Introducing semicolon into a program. Concepts and properties. Refinement rules for semicolon.

Chapter 9 – The Alternative Command The if command and its properties. Using the command in refinement. Refinement rules for alternatives.

Chapter 10 – The Iterative Command The do command and its properties. Using the command in refinement. Refinement rules for iteration. The problem of termination and its proof. Loop parameters: initialization, guards, invariants and termination. Establishing the invariant. Refining loop bodies.

Chapter 11 – Functions and Procedures The definition of procedures. Introducing function procedures and proper procedures. Refinement rules for introducing and removing procedures and their calls. Refinement rules for using recursion.

Chapter 12 – Examples of Refinement at Work From a simple specification to executable code, the emphasis is on showing the ideas at work.

Chapter 13 – On Refinement and Loops How to refine a specification to use iteration. Hints on finding the guard, invariant and variant terms.

Chapter 14 – Functions and Procedures in Refinement Refining to procedures – introducing procedure declarations and procedure calls. Using recursion.

Chapter 15 – Refinement and Performance Using refinement to improve the performance of a program – a fast multiply and a fast integer divide algorithm.

Chapter 16 – Searching and Sorting Refining a search specification to executable code (linear and binary search). Refining a sort specification to executable code.

Chapter 17 – Data Refinement Replacing abstract data models with data that can be implemented by an executable programming language. The idea of data refinement. An informal explanation of the theory. A simple data refinement.

Chapter 18 – A Theory of Data Refinement The theory behind the rules and methods introduced in Chapter 17. Other approaches to data refinement. Dealing with assignment. This chapter is optional, but it does complete the theory of refinement and shows how everything fits together.

Chapter 19 – An Alternative Refinement of the security system Alternative ways of solving the specification used in Chapter 17.

Chapter 20 – Stacks and Queues Refining stacks and queues to executable code to illustrate data refinement in action. Different approaches to the same problem.

Chapter 21 – Dynamic Data Structures How to deal with pointers. Proving the correctness of a data representation that uses pointers.

Chapter 22 – Binary Trees A formal derivation of the standard algorithms for working with ordered binary trees.

Chapter 23 – Epilogue What next.

This book was produced using the software package ‘Textures’ (Blue Sky Research) and the \LaTeX_ϵ and NPL VDM-SL macro packages on an Apple Macintosh computer.

Contents

1. Writing Correct Programs	1
1.1 Satisfying Specifications	1
1.2 An Alternative Approach	4
1.3 Summary	6
2. A Small Programming Language	7
2.1 Satisfying Specifications	7
2.2 Specifications and Programs	11
2.3 The Semantics of Commands	12
2.3.1 Some Primitive Commands	15
2.3.2 A Basic Command	16
2.3.3 New Commands from Old – Operators	17
2.4 Non-determinism and Partial Commands	21
2.5 The Concepts of Predicate Transformers	24
2.6 Substitution	26
2.6.1 Rules for Substitution	28
2.7 The Formal Semantics of the Kernel Language	28
2.7.1 The Bounded Commands	28
2.7.2 The Unbounded Commands	31
2.8 The Weakest Liberal Pre-conditions, Termination, and Relations	32
2.9 Executable Programs	36
2.10 Summary	36
3. Concepts and Properties	37
3.1 More Commands	37
3.2 The Domain of a Command	40
3.3 Some Properties of Commands	45
3.3.1 Sequence	45
3.3.2 Bounded Non-deterministic Choice	46
3.3.3 Guarded Commands	47
3.3.4 Unbounded Non-deterministic Choice	49

3.3.5	Assertions	50
3.4	A Normal Form	50
3.5	Some Further Properties	54
3.5.1	Setting and Testing Variables	56
3.6	The Primitive Commands Revisited	61
3.7	Initial Values	63
3.8	A Compiler for the Small Language	65
3.9	Summary	65
4.	Building New Commands from Old	66
4.1	Defining Commands	66
4.2	An Approach to Recursion	68
4.3	Sequences of Predicates and their Limit	70
4.3.1	A Skeptical Result	72
4.3.2	A Credulous Result	74
4.4	Limits of Predicate Transformers	75
4.4.1	The Two Approaches	77
4.5	Limits of Commands	78
4.6	Tidying the Loose Ends	81
4.7	Epilogue	82
5.	Program Refinement	84
5.1	Stepwise Refinement	84
5.2	Replacing Specifications	85
5.3	Conventions	96
5.4	Refinement Classes	96
5.5	Alternative Views of Refinement	99
5.6	Other Refinement Relations	101
5.6.1	Weak Refinement	101
5.6.2	Partial Refinement	102
5.6.3	Full Refinement	102
5.6.4	Strong Refinement	102
5.6.5	Refinement by Simulation	103
5.7	Summary	104
6.	The Basic Commands	105
6.1	The Constant Commands	105
6.2	Assertions	109
6.3	Assignment	112
6.3.1	Implementation Issues	115
6.4	Summary	116

7. Declarations and Blocks	117
7.1 The Declaration Command	117
7.2 Some Interactions Between Commands	118
7.3 The Definitional Commands	121
7.3.1 Declarations	122
7.3.2 The Let Command	122
7.3.3 The Def Command	123
7.3.4 The Def and Let Commands Compared	123
7.4 Refining Definitional Commands	125
7.5 Defining Constant Values	127
7.5.1 Refining Functions and Constants	128
7.6 Logical Constants	129
7.7 Summary	134
8. Command Sequences	135
8.1 Solve a Part and then the Whole	135
8.1.1 Choosing the First Step	137
8.1.2 Choosing the Second Step	140
8.1.3 Choosing Both Steps	142
8.2 Summary	144
9. The Alternative Command	145
9.1 Divide and Conquer	145
9.2 The Partition	147
9.3 Reassembly	150
9.4 Alternatives – The If Command	152
9.5 Refining Specifications	157
9.6 Summary	158
10. The Iterative Command	159
10.1 Another Approach	164
10.2 The Generalized Loop and Refinement	166
10.3 The General Variant Theorem	168
10.4 An Application	170
10.5 Loops	172
10.6 Iteration – The Do Command	176
10.7 Practical Do Commands	179
10.8 The Refinement of Loop Bodies	180
10.8.1 Decreasing Variants	180
10.8.2 Increasing Variants	182
10.9 Summary	183

11. Functions and Procedures	184
11.1 Proper Procedures and their Calls	184
11.2 Function Procedures and their Calls	190
11.3 Function Calls in Expressions	193
11.4 An Alternative Approach to Parameters and Arguments	195
11.5 Postscript	195
11.6 Summary	196
12. An Example of Refinement at Work	197
12.1 The Problem – Integer Multiplication	197
12.1.1 Getting Started	197
12.1.2 The Refinement Strategy Continued	198
12.1.3 The Next Step	199
12.2 Logical Constants Revisited	203
12.3 Summary	204
13. On Refinement and Loops	205
13.1 The Factorial Problem	205
13.1.1 The First Solution	205
13.1.2 A Second Solution	207
13.2 Finding Guards and Invariants	210
13.3 Identifying the Loop Type Incorrectly	215
14. Functions and Procedures in Refinement	216
14.1 Factorial	216
14.2 Multiply	218
14.3 Summary	221
15. Refinement and Performance	222
15.1 A Second Approach to Multiplication	222
15.2 Fast Division	224
15.3 Summary	229
16. Searching and Sorting	230
16.1 A Diversion – the Array Data Type	230
16.2 Linear Search	231
16.3 Binary Search	234
16.4 A Simple Sort Algorithm	239
16.4.1 The First Attempt	240
16.4.2 The Other Approach	243
16.5 Summary	246

17. Data refinement	247
17.1 The Refinement Strategy	247
17.1.1 The Problem	248
17.2 The Refinement to Executable Code	251
17.3 The Next Step	255
17.4 The Refinement of the Operations	260
17.5 The First Refinement Step	266
17.6 The Next Refinement Step	268
17.6.1 The <i>check</i> Operation	277
17.6.2 Putting it all Together	277
17.6.3 Further Development	278
17.6.4 The Final Step	278
17.7 A Summary of the Approach	278
17.8 Summary	281
18. A Theory of Data Refinement	282
18.1 An Approach to Data Refinement	282
18.1.1 The Data Refinement of Declarations	296
18.1.2 Refinement and Specifications	296
18.2 Data Refinement in Practice	299
18.3 Another View of Data Refinement	300
18.4 Functional Refinement	303
18.5 An Alternative Data Refinement of Assignments	304
18.6 Summary	307
19. An Alternative Refinement of the Security System	308
19.1 A Data Refinement	308
19.2 Another Approach to the Refinement	313
19.3 Summary	317
20. Stacks and Queues	318
20.1 A Finite Stack	318
20.1.1 The Refinement	319
20.1.2 Reorganizing the Operations	321
20.2 A stack of Boolean Values	327
20.2.1 Some Lemmas about the Representation	329
20.2.2 The <i>empty-stack</i> Operation	330
20.2.3 The <i>push</i> Operation	330
20.2.4 The <i>pop</i> Operation	330
20.2.5 The <i>read</i> Operation	331
20.2.6 The <i>is-empty</i> Operation	331
20.2.7 The <i>is-full</i> Operation	332
20.2.8 The Code	333
20.2.9 Some Lessons	333
20.3 A Finite Queue	333

20.3.1	A Refinement of the Queue	336
20.3.2	Some Theorems	338
20.3.3	The Operations Transformed	339
20.3.4	An Extension to the System	343
20.4	An Efficient Queue	345
20.4.1	Some Properties	347
20.4.2	Lessons	351
21.	Dynamic Data Structures	352
21.1	Simulating a Linked List	352
21.1.1	Some Theorems	353
21.1.2	The Operations Transformed	354
21.2	Explicit Pointers	355
21.3	The Stack Using Explicit Pointers	358
21.3.1	Standard Stack Specification to Pointers	360
21.4	Summary	368
22.	Binary Trees	370
22.1	The Specification	370
22.2	The Refinement	370
22.3	The Refinement of the <i>in</i> Operation	371
22.4	The Refinement of the <i>insert</i> Operation	374
22.5	The Refinement of the <i>delete</i> Operation	377
22.6	An In-order Traversal	382
22.7	Summary	387
23.	Epilogue	388
23.1	An Approach to Loose Patterns and Functions	389
A.	Program Refinement Rules	393
A.1	Replace Specification	393
A.2	Assume Pre-condition in Post-condition	393
A.3	Introduce Assignment	393
A.4	Introduce Command-semicolon	394
A.5	Introduce Semicolon-command	394
A.6	Introduce Leading Assignment	394
A.7	Introduce Following Assignment	395
A.8	Introduce Alternatives	395
A.9	Introduce Iteration	395
A.10	Introduce Proper Procedure Body	396
A.11	Introduce Proper Procedure Call	396
A.12	Introduce Function Procedure Body	397
A.13	Introduce Function Procedure Call	397
A.14	Add Variable	397
A.15	Realize Quantifier	398