Computational Artifacts

Raymond Turner

Computational Artifacts

Towards a Philosophy of Computer Science

Springer

Raymond Turner
School of Computer Science
& Electronic Engineering
University of Essex
Colchester, UK

ISBN 978-3-662-55564-4 ISBN 978-3-662-55565-1 (eBook)
https://doi.org/10.1007/978-3-662-55565-1

Library of Congress Control Number: 2018946563

© Springer-Verlag GmbH Germany, part of Springer Nature 2018
This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.
The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.
The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Printed on acid-free paper

This Springer imprint is published by the registered company Springer-Verlag GmbH, DE part of Springer Nature.
The registered company address is: Heidelberger Platz 3, 14197 Berlin, Germany

Preface

This book has been some time coming. There are several reasons for this.

For some of the subject matter I had a clear conception of the central questions and concerns, but there were other aspects for which my ideas were barely worked out. Preliminary work on these other topics took considerable time. Not only had I not given them much thought, but neither had anyone else. More generally, the literature on the philosophy of computer science, at least in my conception of it, was sparse.

Secondly, I had no overriding conceptual framework. This made it difficult even to organize the material that I did have. Eventually this was provided by the notion of a *technical artifact*, and with this the speed of writing increased a little. But there was still a major concern: how to fit all the topics which I now felt should be part of any such book within this framework. Here the literature on the philosophy of technology provided some real help, but there was still considerable work to do on reinterpreting matters in computational terms.

Thirdly, the book must appeal, and be accessible to, philosophers as well as computer scientists. The subject matter of computer science is highly technical in nature, and part of any such book, at least at this stage in the development of the subject, must provide an exposition of the relevant core material. Books on computer science are written for a technically minded audience who have the right mathematical and engineering backgrounds. In contrast, philosophy appeals to those who have a more analytic frame of mind, where conceptual puzzles and arguments form the heart of matters. Somehow this pedagogical aspect had to be smoothly fused with the philosophical analysis. Not an easy task.

Any of these barriers would have delayed matters, but their combination has resulted in a very long delay between conception and production. Hopefully, this considerable period of gestation has resulted in a worthy introduction to the philosophy of computer science. Of course, I am still not satisfied with the result. Many of the chapters do little more than sketch the issues; many are just tasters. However, my hope is that the book will provide an overall conception, and a set of questions

and puzzles, that will encourage more philosophers to engage with the material, and thereby inspire a deeper and more complete philosophical analysis of an area of knowledge that has transformed our contemporary world.

My greatest debt goes to Nicola Angius. Working with him over the past few years has helped me to shape my concept of the philosophy of computer science. Liesbeth de Mol and Giuseppe Primiero have, through HaPoC, provided many forums in which I have delivered keynote presentations that have been instrumental in developing many of the ideas of the present work. Their support and insights have been invaluable.

Discussions with Colin Phillips on Wittgenstein's philosophies of language and mathematics have greatly influenced my approach to the present subject. The notion that programming involves rule following is a theme that crops up throughout the book.

Discussions with the staff at my home university, especially Sam Steel and Chris Fox, have been instructive and encouraging.

Finally, without my friends and family I would not have started, let alone finished.

Colchester, April 2018 *Raymond Turner*

Contents

Part I INTRODUCTION

1 COMPUTER SCIENCE 5
1.1 Machines 5
1.2 Theoretical Computer Science 7
1.3 Programs and Programming 8
1.4 Computational Thinking 10
1.5 The Discipline 11

2 TOWARDS A PHILOSOPHY OF COMPUTER SCIENCE 13
2.1 Semantics 13
2.2 Ontology 15
2.3 Methodology 16
2.4 Epistemology 17
2.5 Conclusion 18

Part II ONTOLOGY

3 COMPUTATIONAL ARTIFACTS 25
3.1 Function and Structure 25
3.2 Design and Manufacture 26
3.3 Theories of Function 27
3.4 Computational Artifacts 28

4 LOGIC MACHINES AS TECHNICAL ARTIFACTS 31
4.1 Function 31
4.2 Structure 33

4.3 Design . . . 36
4.4 Correctness . . . 37
4.5 Implementation . . . 37
4.6 Logic Machines as Technical Artifacts . . . 38
4.7 The Abstract-Concrete Interface . . . 39
4.8 The von Neumann Computer as Artifact . . . 39
4.9 High-Level Functional Notation . . . 40
4.10 Conclusion . . . 41

5 THE ONTOLOGY OF PROGRAMS . . . 43
5.1 The Functional Specification of Programs . . . 44
5.2 Structure . . . 48
5.3 Implementation . . . 49
5.4 The Symbolic and the Physical . . . 50
5.5 Programs as Technical Artifacts . . . 52

6 SOFTWARE SYSTEMS AS TECHNICAL ARTIFACTS . . . 53
6.1 Requirements . . . 53
6.2 Structure and Design . . . 54
6.3 Implementation . . . 55
6.4 Software Systems as Artifacts . . . 56
6.5 Verification, Validation, and Malfunction . . . 57
6.6 Conclusion . . . 57

Part III SEMANTICS

7 THE LANGUAGES OF COMPUTER SCIENCE . . . 63
7.1 Varieties of Languages . . . 63
7.2 Formal Languages . . . 65

8 PROGRAMMING LANGUAGES . . . 67
8.1 Imperative Languages . . . 68
8.2 Functional Languages . . . 71
8.3 Logical Languages . . . 73
8.4 Object Orientation . . . 74
8.5 Concurrency and Nondeterminism . . . 75
8.6 Theories of Representation and Computation . . . 76

9 SEMANTIC THEORIES 77
9.1 The Roles of Semantics 77
9.2 Normativity 79
9.3 Compositionality 80
9.4 Rigor 82
9.5 Conclusion 83

10 FORMAL SEMANTICS 85
10.1 Vernacular Semantics 85
10.2 Operational Semantics 86
10.3 Denotational Semantics 90
10.4 Definitional Priority 91
10.5 Game-Theoretic Semantics 93
10.6 Programming Languages as Mathematical Theories 93

11 SEMANTICS AND IMPLEMENTATION 95
11.1 Compilers, Interpreters, and Virtual Machines 95
11.2 Compilation and Semantic Interpretation 98
11.3 Semantics and Interpretation 99
11.4 Programming Languages as Technical Artifacts 103

12 SPECIFICATION LANGUAGES 105
12.1 Typed Predicate Logic 105
12.2 The Logic of TPL 107
12.3 Definitions 108
12.4 Z and Set Theory 109
12.5 VDM and Three-Valued Logic 110
12.6 Types, Not Sets 111
12.7 Expressive Power 112

Part IV METHODOLOGY

13 SOFTWARE SYSTEM METHODOLOGY 117
13.1 The Waterfall Method 118
13.2 The Spiral Method 118
13.3 The Agile Method 118
13.4 Methodology 119

14 SPECIFICATION 121
14.1 Requirements Analysis as Modeling 121
14.2 Definition 122
14.3 Intention 123
14.4 Intentional Stance 124
14.5 Precision and Information 126

15 THE PHILOSOPHY OF DESIGN 129
15.1 Correctness 130
15.2 Simplicity 130
15.3 Modularity 131
15.4 Formal Methods 131

16 SIMPLICITY 133
16.1 Elegance 134
16.2 Parsimony 137
16.3 Justification 138

17 MODULARITY 141
17.1 Encapsulation 142
17.2 Bundling 143
17.3 Information Hiding 144
17.4 Independence 145
17.5 Single Function 146
17.6 Cohesion and Coupling 147
17.7 Modularization 148

18 FORMAL METHODS 151
18.1 Formal Specification 152
18.2 System Properties 153
18.3 Programming 153
18.4 Module Correctness 154
18.5 Transformational Programming 157
18.6 Constructive Programming 157
18.7 Limitations of Formal Methods 158
18.8 Programming as a Mathematical Activity 159

19 THE DESIGN OF PROGRAMMING LANGUAGES 161
19.1 Simplicity 161
19.2 Elegance via Uniformity 162
19.3 Parsimony 164
19.4 The Defense of Simplicity 165
19.5 Expressive Power 165

20 SEMANTICS AND DESIGN 169
20.1 The Principle of Correspondence 170
20.2 Type Completeness 172
20.3 The Principle of Abstraction 173
20.4 Programming Languages as Mathematical Theories 174

21 DATA ABSTRACTION 175
21.1 The Traditional Account of Abstraction 176
21.2 The Way of Abstraction 177
21.3 Russell's Paradox 179
21.4 Data Abstraction 179
21.5 Principles of Abstraction 181
21.6 Implementation as a Basis for Abstraction 182
21.7 A Theory of Computational Abstraction 183

Part V EPISTEMOLOGY

22 COMPUTABILITY 189
22.1 Mathematical Modeling 189
22.2 The Programming Language Approach 190
22.3 Invariance 191
22.4 The Machine Approach 192
22.5 Semantics and Turing Machines 192
22.6 Turing Machines as Technical Artifacts 194
22.7 Machines and Rules 194

23 FEASIBLE COMPUTATIONS 197
23.1 Computational Complexity Theory 197
23.2 The Cobham-Edmonds Thesis 198
23.3 Quantum Computing and Feasibility 199

24 VARIETIES OF CORRECTNESS 201
24.1 Digital Circuits 202
24.2 Programs 203
24.3 Software Systems 204

25 PROGRAM CORRECTNESS 205
25.1 The Mathematical Challenge 205
25.2 The Mechanical Challenge 207
25.3 The Pragmatic Challenge 209
25.4 The Scientific Challenge 210
25.5 Proof and Principles of Program Design 211

26 TYPES AND CORRECTNESS 213
26.1 Type Inference 214
26.2 Semantics 215
26.3 Implementation 216
26.4 Correctness 217

27 THE SIMPLE MAPPING ACCOUNT 219
27.1 The Correctness of Electronic Devices 219
27.2 Simple Mapping Account 220
27.3 The Mathematical Case 222
27.4 Good and Bad Programs 223

28 COMPUTATIONAL EXPLANATION 225
28.1 Technological Explanation 225
28.2 Mathematical Explanation 227
28.3 Structural Explanation 229
28.4 Conclusion 230

29 INTENTION AND CORRECTNESS 231
29.1 The Standard Account 232
29.2 Definition and Intention 232

30 RULE FOLLOWING AND CORRECTNESS 235
30.1 Computing and Programming 235
30.2 Rule Following 236
30.3 The Skeptical Paradox 237
30.4 The Skeptical Solution 238

30.5 Rules and Customs 238
30.6 Rule Following and Computer Science 239

Nomenclature 241

References 243

Index 253