THE COGNITIVE DYNAMICS
OF COMPUTER SCIENCE
Press Operating Committee

Chair
Roger U. Fujii,
Vice President
Northrop Grumman Mission Systems

Editor-in-Chief
Donald F. Shafer
Chief Technology Officer
Athens Group, Inc.

Board Members

Mark J. Christensen, Independent Consultant
Herb Krasner, President, Krasner Consulting
Ted Lewis, Professor Computer Science, Naval Postgraduate School
Hal Berghel, Professor and Director School of Computer Science, University of Nevada
Phillip Laplante, Associate Professor Software Engineering, Penn State University
Richard Thayer, Professor Emeritus, California State University, Sacramento
Linda Shafer, Professor Emeritus University of Texas at Austin
James Conrad, Associate Professor UNC- Charlotte
Deborah Plummer, Manager - Authored books

IEEE Computer Society Executive Staff
David Hennage, Executive Director
Angela Burgess, Publisher

IEEE Computer Society Publications
The world-renowned IEEE Computer Society publishes, promotes, and distributes a wide variety of authoritative computer science and engineering texts. These books are available from most retail outlets. Visit the CS Store at http://computer.org/cspress for a list of products.

IEEE Computer Society / Wiley Partnership
The IEEE Computer Society and Wiley partnership allows the CS Press authored book program to produce a number of exciting new titles in areas of computer science, computing and networking with a special focus on software engineering. IEEE Computer Society members continue to receive a 15% discount on these titles when purchased through Wiley or at wiley.com/ieees

To submit questions about the program or send proposals please e-mail dplummer@computer.org or write to Books, IEEE Computer Society, 100662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314. Telephone +1-714-821-8380.

Additional information regarding the Computer Society authored book program can also be accessed from our web site at http://computer.org/cspress
THE COGNITIVE DYNAMICS OF COMPUTER SCIENCE

Cost-Effective Large Scale Software Development

Szabolcs Michael de Gyurky

Computer Artwork by

Mark A. Tarbell
To Immanuel Kant

That wonderful teacher,
that exquisite mind and heart,
and beautiful soul.
CONTENTS

List of Figures xv
Acknowledgments xvii

Chapter 1 Introduction 1
1.1 The Retention of Software Jobs 2
1.2 Depth of Experience 2
1.3 The Scope of This Book 3
1.4 The Nature of Computer Science 4
1.5 The Future of Computer Science 4
1.6 The Essence of Philosophy 5
1.7 Why Autonomy? 6
1.8 An Architecture for Autonomy 8
1.9 Other Notes 9

Chapter 2 Prologue 11
2.1 How This Book Originated 11
2.2 The Importance of Management 12
2.3 The Tie-in with Autonomy 13
2.4 Major Themes of This Book 14
2.5 The Challenge of a New Idea 14
2.6 The Importance of Visualization 15
2.7 The Move Toward Autonomy 16
2.8 Why I Wrote This Book 17
2.9 Merging Theory and Practice 18
2.10 The Pace of Computer Science 19
2.11 The Importance of Cognitive Dynamics 21

Chapter 3 The Philosophical Foundations of Computer Software Design 23
3.1 The Philosophical Origins of Computer Science 23
3.2 Influence of the Cognitive Philosophers 25
3.3 Abstracting the Human Thought System 27
3.4 The Philosophical Foundations of Software Development 28
3.5 The Phenomenon of Reality 29
Chapter 4  The Philosophical Imperatives of Architectural Design  36

4.1  The Manager as Architect  36
4.2  The Manager as Teacher  37
4.3  The Manager as Social Worker  38
4.4  The Manager as Axman  39
4.5  The Philosophical Imperatives of Architectural Design  39
4.6  Availability of the Manager  40
4.7  Project Manager: 10 Key Attributes and Responsibilities  40
4.8  Philosophical Aspects of Engineering  43
4.9  The Importance of Finishing the Job  44
4.10  Visualizing an Architecture  44
4.11  The Role of Intuition in Design  47
4.12  “Sufficient Reality” and Inference in the Design Process  48
4.13  Dialectics in the Achievement of Sufficient Reality  49
4.14  The Relationship of Logic to Software Architectures  50
4.15  The Logic of the Systems Design  53

Chapter 5  Project and Task Organization  54

5.1  The Role of Organization  55
5.2  The Ability to Organize  57
5.2.1  Traditional Hierarchical Project Organization  57
5.3  The Difficulty of Communication  60
5.4  The Title of “Manager”  61
5.5  The Flat, Nonhierarchical Organization  61
5.6  Projects, Tasks, and Work Units  65
5.7  Large Organizations and Staffing  66
5.8  Staffing Up: The Initial Team  69
5.8.1  The Initial Team  69
5.8.2  Phase One Team Expansion  70
5.8.3  Phase Two Team Expansion  72
5.9  Balancing Hardware and Software  73
5.10  Incremental Deliveries  75
5.11  Functional Organization  76
5.12  Interface Protocols of the Organization  77
### Chapter 6  The Philosophy of Communication  
6.1  “Sanity Is an Achievement!”  81  
6.2  Gauging Understanding  82  
6.3  Internal Team Communication Protocols  82  
6.4  External Team Communication Protocols  84  
6.5  Technical English as the Medium  85  
6.6  Engineers as Technical Writers  87  
6.7  Documentation: Articulation of the Requirements and Design  87  
6.8  The SRD: Software Requirements Document  88  

### Chapter 7  Software Management Standards  
7.1  Three Good Standards  91  
7.1.1  JPL-STD-D-4000  92  
7.1.2  MIL-STD-498  92  
7.1.3  DOD-STD-2167A  92  
7.2  Aspects Addressed by a Standard  93  
7.3  Preparing to Select the Standard  94  
7.4  Standards for Implementation  96  
7.4.1  Waiving the Standard  97  

### Chapter 8  The Estimation of Software Cost  
8.1  Sponsor Costing Issues  99  
8.2  Types of Cost Estimates  101  
8.3  “Lines of Code” Metrics  101  
8.4  The Major Work Areas, Functions, and Tasks that Must be Included in the Estimation of Cost  103  
8.5  The Detailed Cost Estimate  105  
8.6  The SRD as a Contract  106  

### Chapter 9  The Exercise of Project Control  
9.1  The Functions of Project Control and Oversight  109  
9.2  The Requirements Phase  110  
9.3  Contents of the Software Requirements Document  111  
9.4  The Design Phase  113  
9.5  The Implementation Phase  113  
9.6  The Test and Integration Phase  115  
9.7  Personnel Issues  116  
9.8  The Hacker and Other Personalities  118  
9.9  The Buck Stops at the Top  119  
9.10  How People Think, Pay Attention, and Remember  121
Chapter 10 The Development Process Methodology 125
10.1 The “Design Hub” as Implementation Tool 126
10.2 The Architecture Definition Process 127
10.3 The Use of Large-Scale Representations 129
10.4 Design Team Meetings 130
10.5 Rapid Development versus Prototyping 131
10.6 The Traditional Development Methodology 132
10.7 Action Items, Change Requests, and Software Discrepancy Reports 134
10.8 Resolving Problems and Impasses 134

Chapter 11 The Development of System Architectures 136
11.1 Pushing the Architecture 137
11.2 The Point of “Acceptable Reality” 138
11.3 The Importance and Imperative of Visualizing Phenomena 140
11.4 Traditional Architectures 141
11.5 The Inferred Architecture 142
11.6 The Redesign or Upgrading of Existing Systems 144
11.7 The Approach to New Systems 145

Chapter 12 The Impact of Leadership on Software Development 146
12.1 Recognizing Good Leadership 146
12.2 The Concepts of Management and Leadership 148
12.3 Rewarding Failure 149
12.4 The Leader’s Subordinate 151
12.5 Indications of Poor Leadership 152
12.6 Leadership and Ethics 153
12.7 The Attributes of Leadership 153
   12.7.1 Unselfishness 154
   12.7.2 The Welfare of Others 154
   12.7.3 Ambition 154
   12.7.4 Integrity 154
   12.7.5 Loyalty 155
   12.7.6 Knowledge 155
   12.7.7 Tact 156
   12.7.8 Judgment 157
   12.7.9 Initiative 157
   12.7.10 Bearing 158
   12.7.11 Courage 158
   12.7.12 Decisiveness 159
12.7.13 Dependability 159
12.7.14 Dynamic Energy 160
12.7.15 Enthusiasm 161
12.7.16 Empowerment 163
12.8 The Ramifications of Failure 164
12.9 The Absence of Leadership 165
  12.9.1 Absenteeism 166
  12.9.2 Hidden Agendas 166
  12.9.3 Communication Gap 167
  12.9.4 Poorly Defined Goals 167
12.10 The Basis in Leadership for Failure 168
  12.10.1 Personal Struggles 168
  12.10.2 The “Machiavellian Prince” 169
12.11 The impact of Poor or Nonexistent Leadership 169
  12.11.1 Conquering the Organization 170

Chapter 13 Management of Software Systems Development 172
  13.1 Self-Respect in the Manager 173
  13.2 The Ethical Workplace 173
  13.3 Narcotics Use in the Workforce 174
  13.4 Spotting Narcotics Addicts 177
  13.5 Courage and Dynamic Energy in Management 178
  13.6 The Traveling Manager 180
  13.7 The Manager as Architect 181
  13.8 The Phenomenon of Decision Making 182
  13.9 The Concept of “Ability” 186
  13.10 Manager: Administrator or Leader? 187
  13.11 Authority, Responsibility, and Accountability 189
  13.12 The Issue of Contempt 189
  13.13 Management: The Fulcrum of Project Execution 191
  13.14 The Ascendance of Mediocrity 191
  13.15 The Pitfalls of Staffing Up 193
  13.16 Salary Issues 195
  13.17 Contracting Out Work 196
  13.18 Evaluating Proposals 197
  13.19 Cost Bidding too Early 198

Chapter 14 Four Case Studies of Low-Cost Systems 200
  14.1 Case Study One: The Joint Theater Level Simulation (JTLS) 202
    14.1.1 The Beginnings of JTLS 204
    14.1.2 Estimating the Cost of War 205
14.1.3 Starting up the Effort 208
14.1.4 Costly Lessons Learned 209
14.2 Case Study Two: The Global Decision Support System (GDSS) 211
   14.2.1 GDSS System Size 211
   14.2.2 The History and Background of GDSS 212
   14.2.3 Expediting the System 213
   14.2.4 The Euler Sphere 214
   14.2.5 Beyond State of the Art 214
   14.2.6 A Replicated, Survivable, Synchronous Database Management System 214
   14.2.7 The Ultra Large Screen Display System 215
   14.2.8 The Local Area Networks 215
   14.2.9 The Wide Area Network 215
   14.2.10 Distributed Client/Server Technology 215
   14.2.11 Message Bus 215
   14.2.12 The GDSS Software Architecture 217
   14.2.13 Accepting the Challenge 220
   14.2.14 Initial Conditions 220
   14.2.15 Rapid Development: A Totally Different Approach 221
   14.2.16 There Can Be Only One! 223
   14.2.17 GDSS End-to-End Architecture 224
   14.2.18 Architecting the Development Effort 224
   14.2.19 Inferential Systems Architecture 225
   14.2.20 The GDSS System Software Layer 226
   14.2.21 Applications Language Selection 227
   14.2.22 Project Documentation 228
   14.2.23 Finding an Ada Expert 228
   14.2.24 Testing and Database Design 229
   14.2.25 Additional Difficulties 231
14.3 Case Study Three: The Topex TCCS System 233
   14.3.1 The Topex TCCS System 233
   14.3.2 System Description 233
   14.3.3 The Initial Conditions 235
   14.3.4 Project Constraints 235
   14.3.5 Implementation Considerations 236
   14.3.6 Development of TOPEX TCCS 237
   14.3.7 Agreeing to Do the Job 238
   14.3.8 Ground Truth 239
   14.3.9 Start of Project Development 239
   14.3.10 Architecting the Environment 241
   14.3.11 Hardware Procurement, Software Procurement 242
   14.3.12 The Relationship with the Contractor 244
14.3.13 Test Plan Scheduling  245
14.3.14 Adherence to a Standard  246

14.4 Case Study Four: The Jason 1 TCCS System (JTCCS)  246
14.4.1 The Jason 1 TCCS System  247
14.4.2 System Description  247
14.4.3 The Initial Conditions  248
14.4.4 Implementation Considerations  249
14.4.5 The JTCCS Architecture  251

Chapter 15 Operations, Operators, and Users: Their Impact on Cost  257
15.1 The Operational Requirement  258
15.2 The Lack of an Operational Requirement  259
15.3 The Operations Scenario  259
15.4 The Cost of Operators and Analysts  260
15.5 The Voyager Project Operations Center  261
15.6 War Gaming  262
15.7 The Value of Simulation  264
15.8 Funds: A Perspective  264

Chapter 16 The Autonomous Cognitive System  266
16.1 Introduction  266
16.2 The Scale of Autonomy  267
16.2.1 Category IV Autonomous Cognitive System: Superman  267
16.2.2 Category III Autonomous Cognitive System: Perseus  268
16.2.3 Category II Autonomous Cognitive System: Robot  269
16.2.4 Category I Autonomous Cognitive System: Automaton  269
16.3 “I Will, Because I Can”  270
16.4 Toward Cognitive Dynamics  271
16.5 Building an Autonomous System  271
16.6 An Appropriate Model  272
16.7 System-Level Requirements for Autonomy  273
16.8 Architectural Domains for Autonomy  274
16.8.1 Domain I: The Human Thought Architecture Model (Functional Architecture)  274
16.8.2 Domain II: The Human Thought Process Model (Common Software Services)  275
16.9 In Summary  277
CONTENTS

Epilogue 279

The Science of Computer Science 279
The Professional Software Manager 279
Cognitive Philosophy in a Modern Technical Context 280
Cognitive Dynamics is the Unifying Theory 281
The Issue of Software Cost 281
The Paradigm Shift of Cognitive Dynamics 282

Glossary of Acronyms 283

Index 287
LIST OF FIGURES

Figure 1  Computer Science Architectural Interpretation of Kantian Philosophy
Figure 2  The Requirements and Design Process: Dialectically Achieving “Sufficient Reality”
Figure 3  Project Organization: Traditional Hierarchical Approach: Isolated Control
Figure 4  Project Organization: Flat, Nonhierarchical Approach: Optimal Control (Front View)
Figure 5  Project Organization: Flat, Nonhierarchical Approach: Optimal Control (Overhead)
Figure 6  Estimation of Software Cost: Software Work-Estimation Worksheet
Figure 7  Two General Approaches to System Development: Known & Unknown Requirements
Figure 8  The Kantian Thought Process: Decision-Making Schema
Figure 9  Joint Theater Level Simulation: Operating System-dependent Architecture
Figure 10  Global Decision Support System: Wide Area Network Configuration
Figure 11  Global Decision Support System: Integrated Software Functional Design
Figure 12  Global Decision Support System: Official Memo of Recognition
Figure 13  Topex/Poseidon TCCS Software Architecture
Figure 14  The Topex War Room: The Design Hub as an Implementation Environment
Figure 15  JASON-1 TCCS Software Architecture
Figure 16  JASON-1 TCCS Pluggable Architecture
Figure 17  JASON-1 TCCS User Interface Client/Server Architecture
ACKNOWLEDGMENTS

To my dear friend, editor, and critic, Mark Tarbell, whose artwork graces these pages: Thank you.
To my dear wife, Cheryl, for her criticism and encouragement: I could not have done it without your support.

In a systems development environment, ideally one learns as much from one’s subordinates as from one’s superiors. Subordinates and superiors are therefore equally teachers in their disciplines. So I’m fortunate to have had great teachers, from above and below, and it has been a privilege to have had the opportunity to work with truly gifted engineers and scientists here at the Jet Propulsion Laboratory. I thank our great team members, past and present, starting with my mentor and teacher at SHAPE in Belgium, Colonel, USAF, Joseph Bullers, my boss who taught me command and control concepts, and architectures on a global scale.

To Dr. Joseph P. Fearey at JPL, who as my partner, mentor, and teacher taught me the mathematical underpinnings of event-driven simulations expressed in computer software architectures and the art of computer modeling and event-driven simulation: Thank you!

A big thank you also to the championship engineering team members without whom I could never have done the jobs I have done successfully:

Henry Judd       John Guineau       Dan Zink       Sebastian Van Alphen
Rick Wissinger   Joe Galinsky      Mark Tarbell    Rod Iwashina
Willie Huo       Art Rinaldi       Louie Hirsch    Tom Collins
Larry Johnsen    Dan Wenrick       Edward Colbert  Diane Melin
Jerry Hill       Don Royer         Jim Brownfield  Paul Firnett
Sara Hyman       Mark Fujishin     Banny Winn      Charles Yamarone
Jay Braun        Mike DiLoreto     Larry Hawley    Joe Provenzano

Kent Thomson, for his unwavering loyalty to the JTCCS and support.
Carol J. Tarbell, for having contributed more to this book than she knows.

To all of you dear colleagues who prepared the requirements and the design, wrote the software, tested it, and maintained the budget and the administrative support flow, a sincere thank you! I especially thank you for never hesitating to tell me when I was wrong, and putting me back on the correct course. Open criticism and the stating of sincere opinions is the heart of Immanuel Kant’s “Pure Ethics.”
As you can see, we are not that big of a team. Out of all those people who came, stayed a while, and moved on, it was we who brought the cost of a quality system down in one case down to under $10 per line of code. I’m not saying that we didn’t have lots of problems and big-time stresses, but it is winning that counts, and we won all of the games we played. “Played” perhaps is not the right word, because it was terribly hard at times.

However, a project should be a team to be effective, and a team is special. It is far more than a project, because on a team the members work unselfishly toward a common goal, and on a project people often work toward their own personal advancement, trying to outshine instead of support each other.

I will always believe that anyone can make software work, or build a system if enough time and money are thrown at it. The quality of the product, built on budget and on schedule, proves the worth of the manager and the team.

Finally, there are people in my two careers who taught me, mentored me, and led me by good example. Without these wonderful teachers I could not have written this book.

So I’m eternally grateful to:

**Colonel Joseph W. Bullers USAF**, for teaching me military command and control.

**Colonel George Dornberger USAF**, for teaching me the meaning and beauty of loyalty and integrity, and for being a true officer and gentleman wrapped in the person of a great fighter pilot.

**Lieutenant General William Ginn USAF**, for teaching me leadership in its finest form.

**Lieutenant Colonel Edward R. Beaty U.S. Army**, for setting the example and teaching me how to be a combat leader.

**Dr. Thomas Thornton** for being a good, steadfast boss, a great manager, and for his confidence in my engineering and management skills.

**Mr. John R. Tupman** for giving me an offer at JPL I couldn’t refuse.
THE COGNITIVE DYNAMICS
OF COMPUTER SCIENCE
Computer software development has become very expensive in these United States, and there are a number of issues that high cost has brought with it. I suppose the most important of these is that the programming jobs are going overseas, along with most of our other manufacturing jobs. We can claim that we still do high-technology design, and the critical jobs of testing and integration. But these jobs, even if we did them well (which we don’t always do), will also go overseas, along with the making of our beloved Levi jeans.

There is also a silent acceptance within the computer science community that the quality of American software is poor, to put it mildly. This is well known, even to our major publishing houses. They note that we write fewer and fewer “readable” books that our young and not so young practitioners can use. We forget that the academic preparation of young Americans is the foundation of their professional lives. It is hoped that a long and serious apprenticeship lies ahead of them, until they, too, become masters of their professional fields. A large portion of our publications and books is written just to publish, and not to teach the young professional. When I pick up foreign textbooks, I’m astonished at the pains their authors took to make them readable and truly useful to the reader. Foreign books are not written solely for one’s academic peers; they are written to teach the reader.

We Americans are the great innovators and inventors, of course, but such jobs generally are restricted to a limited number of research houses, facilities, and universities. The earning of our daily bread as software professionals is becoming more and more difficult as the jobs are going away.

Yet, we are able to produce software of the highest quality, and at a lower cost, here in this country than anywhere else in the world. At present, based on my experience, the Federal Government pays between $350 and $800 per commented line of code for ground systems (the command and control software for aerospace and military applications). And for on-board flight systems—the software that is used in missiles, satellites, aircraft, spacecraft, instruments, and smart-guided weapons—the costs are much higher, ranging from $450 to $1,200 per line of commented code. Such systems are the very reasons for our global military superiority.

In contrast to these staggering costs, my colleagues and I here at the Jet Propulsion Laboratory have been able to reduce the cost of ground systems to about $10 per
commented line of code on our most recent systems. During the past 20 years, my teams and I have produced high-quality software at a cost ranging from $10 to $50 per line of commented code—depending on the mathematical complexity of the application, the availability of the technology in the marketplace, and the schedule to which we had to work.

1.1 THE RETENTION OF SOFTWARE JOBS

As a computer software professional, I am greatly concerned about the retention of software jobs within our borders. My colleagues are the very best in their fields, both in applications and systems software, and are pure technical innovators as well. They are highly paid and worth every penny they earn. I pay my programmers from $45 to $85 per hour, according to their years and levels of experience and skill qualifications (most of my computer scientists are contractors and consultants, as opposed to employees). Yet, we complete our tasks within the schedules and on the budgets we have committed ourselves to, producing a high-quality product at a cost far lower than anywhere else I know of in industry or in government.

Clearly, as a team, we are neither better educated nor more experienced than the rest of our colleagues throughout the country. So why is there such a difference in cost? Were it not for the fact that we are losing our livelihoods, this would be a moot point. But there are many “software guys” out there, including my colleagues, who are trying in vain to find jobs. They are worried and are asking me questions. We have professions for which we have invested large amounts of time and money, and we need incomes to pay our mortgages and raise our families.

So how does a project build software at $10 per line of code, while others build at $800 or more for the identical product? That is the essential question that this book is going to answer, together with:

- What does it take to build quality software at a reasonable cost?
- What are the major factors that influence the cost, quality, and development schedule in software design and implementation?

1.2 DEPTH OF EXPERIENCE

This book is a summary of my 32 years of experience in designing and building computer software: 7 years in the U.S. Army building software, and 25 years at the Jet Propulsion Laboratory. This period of accumulated experience also includes the information I gathered as a software management consultant to some of the finest and best U.S. corporations—indeed, the giants in our industry. The problems of quality, cost, and schedule have proven to be the same, end to end, wherever I was invited to help out. The phenomenon of schedule vs. cost vs. quality in computer science holds true for Germany, France, Holland, Denmark, and the rest of the European countries as well. I have no experience with Japan, China, or India.
directly, but there is no reason to believe that their computer scientists and software professionals are any different from ours.

I do not intend this to be “just another book on software,” as some of my colleagues in industry have remarked dryly, but a worthwhile book to read. The intention is to pass on my accumulated experience in computer science, as well as the relevant experience gained in the combat arms of the U.S. Army, which in my case was the Airborne Infantry. Now, there will be those who will say, “What can you learn about computer science and software in the infantry?” Nothing about computer software, it is true, but everything about management, leadership, understanding people, discipline, planning, organization of work and people, as well as seemingly trivial subjects like how to staff through and coordinate action items efficiently. All of these are essential to our work in computer science and to the development of computer software. This is especially true for large systems—those above 250,000 lines of code with high factors of technical complexity combined with a short schedule. The absence of these skills will increase the cost of the product tenfold or more.

1.3 THE SCOPE OF THIS BOOK

The scope of Chapter 16, “The Autonomous Cognitive System,” is much too large for the subject to be covered in one chapter and must be dealt with in great detail on its own. It is an essential part of the overall scope of this book, however, because it interprets the work of the great cognitive philosophers from the software architect’s point of view. I began to approach the works of Kant, Hegel, and Schopenhauer as a way of looking at computer science many years ago; I never had the time to write my interpretation down until now. When designing and building large software-intensive systems year in and year out, back to back, one does not have the luxury to spend time writing, except for the design-associated books specific to a particular project.

The main theme of this book is high-quality, low-cost software, built on schedule; this theme will be re-emphasized throughout. It is a teaching point, and teaching points are just that: They are there to keep the mind focused, specifically on low-cost, high-quality products, on schedule. A minor theme is the inhibitors, those issues that prevent us from building elegantly, fast, and cheaply. These issues are also deeply rooted in the Kantian human thought system (the architecture) and the human thought process (the information flow), the understanding of which will enable a person to at least grasp the “why,” even if one can’t do anything about it. Understanding and learning from mistakes—one’s own and the mistakes of others—is the great teacher and leads to ever better products.

It became very clear to me over the years that the successful building of computer software at a reasonable cost was dependent on several important factors:

- Leadership
- Management
- Communication
• Organization
• Understanding software development standards, architectures, and methodologies

A project lacking in any of these factors will see an increase in the cost of its software products. This book will discuss each of the factors affecting cost, from a practitioner’s point of view.

1.4 THE NATURE OF COMPUTER SCIENCE

Additionally, two very important issues need to be discussed that also bear directly on computer science and the cost of computer software.

The first of these is raised by the following questions: Where do computer science and computer software belong in engineering? Should software be the dominant part in an engineering effort, such as an aircraft or spacecraft project, or should it serve a supportive role? This issue of computer science and its placement is causing a rub throughout the industry. It is an issue created by the mindset and outlook on the part of individuals. It is a divisive issue, and the resultant conflicts cost the customer money.

The second critical issue is raised by the questions: What is computer science really? Why did we rename it “information technology?” What is the complete definition of computer software? What is the role of philosophy in computer science? Is it an important role? Why?

All one needs to do is look at the important recent publications in our field. We find that there seem to be no answers to our questions. Our profession used to be called automatic data processing. That evolved into computer science, then into information technology, and finally it has branched off into artificial intelligence and neural networks. What this really means is that while we are pushing the state of the art, we have no agreement as to a unifying science, like biology and chemistry have. Engineers deal with tangible facts and theorems, yet we deal in abstractions. Our profession is a combination of art, science, and engineering. So what is it? Why is it so important that we find a unifying principle?

1.5 THE FUTURE OF COMPUTER SCIENCE

We in computer science are headed full-bore toward total autonomy. This means we need to build a true robot, one that can be sent into space. This would be an autonomous system that, in an intelligent fashion and with an intelligence similar to our human intelligence, could explore our solar system and go to places where we humans cannot survive but from which we can benefit. As a starter, for the building

---

of such a system, we need an architecture. We need a model that is rational, functional, and logical, and one that can be built. This poses a serious problem because it involves a paradigm shift in the way we think about our profession and how we work in it. Paradigm shifts are dangerous events, as we all know. Galileo was nearly burned for his ideas, and Bruno was. There is an inborn anger in some of us human beings when we are confronted with a phenomenon we don’t understand. We either learn, or we are left behind.

I feel that here in the United States we have a few great issues at stake. One is retaining our lead in computer science, and in software programming jobs. Another is pushing the state of the art “through the looking glass.” This must be accomplished, regardless of the opposition.

1.6 THE ESSENCE OF PHILOSOPHY

Why is the work of the great cognitive philosophers so important for the development of high-quality software at a reasonable cost? It is because they were the ones who researched how we humans perceive, think, decide, and act, and why we do what we do. A thorough understanding of philosophy is therefore necessary to do what we do efficiently and thoroughly.

The essence of philosophy comprises many factors:

- How we think
- How we organize our thoughts and our work
- How we contemplate
- How we pay attention and listen
- How we decide
- How we form value judgments
- How we communicate with others

Yet, above all, the essence of philosophy concerns itself with how we treat each other. To me, therefore, it is the greatest of all the sciences. Small wonder that many of the great philosophers were mathematicians and physicists. This discipline, philosophy, upon which all the sciences rest, is given scant attention in the curriculum of our universities and by those of us who earn our living in engineering and the other sciences.

The greatest teachers (and I have had the good fortune of having had quite a few) always used stories from their personal experiences to illuminate teaching points. As students, we were more attentive because the learning process was so much more interesting when it was related to historical events. The teaching of geometry and mathematics becomes exciting when related to Harpalus (a great Greek engineer and philosopher) and how he bridged the Hellespont for Xerxes and the Persian Army. Teaching and instruction become more memorable than watching and listening to someone writing equations on the blackboard. How did he build that 1.6-mile-long pontoon bridge between Abydus and
Sestus\textsuperscript{2} Why did he build it there? What was the outcome for Greece and Persia\textsuperscript{3}? As a matter of interest, Harpalus can be justly referred to as someone we call today “a defense contractor,” as were Archimedes and Histiaeus of Miletus\textsuperscript{4} in their day. The Jet Propulsion Laboratory started as a U.S. Army Ordnance Laboratory, during the Second World War. Thus, before NASA was established, JPL, too, started out as a defense contractor.

There is one more very important aspect to using personal experiences in illuminating (or “adding substance” to) an object in a philosophical sense. As you will see in the architecture of the human thought system and the human thought process (Figure 1), the role of “experience” is one of the dominant roles in Kantian philosophy. How we apply our acquired knowledge, learned in the classroom, and solve problems depend largely on the level of our personal experiences (Figure 8). We acquire personal experiences by living life, and learning personally, or by having someone tell us of their personal experiences in life. Thus, the personal stories I recount in this book are to add substance to an “object” of a design, idea, or concept\textsuperscript{5}, not simply to fill space.

Science and history taught together make learning complex subjects far easier. This is a method that is also used in the Army to reinforce a teaching point and to provide the student with a reference to the application of a skill, especially in subjects like engineering and tactics. It so happens that Combat Engineering was among the many skills I acquired through schooling in the U.S. Army. I specialized in bridge design and construction, pontoon bridges and timber trestles being my favorites.

I use stories from personal experience also to illuminate issues of management, leadership, and the architectural design process. I have a habit of doing this in my seminars and classes in order to reinforce the teaching point I am making. After all, the role of experience is one of the dominant themes of the \textit{Three Critiques} of Immanuel Kant.

\subsection*{1.7 WHY AUTONOMY?}

Why do we need a totally autonomous system, one that can act and decide on its own, without human intervention? The first thing that comes to my mind is the


\textsuperscript{4}Histiaetus and Thales actually built a 26-mile pontoon bridge over the Bosphorus for Darius I “The Great.” It collapsed with tragic consequences to those on it. For details, read \textit{The Peloponnesian Wars}.

\textsuperscript{5}Since Chapter 16 of this book culminates in my first thesis for an autonomous system, the reader must understand the correct technical definition of the terms \textit{idea} and \textit{concept} as these dynamic processes relate to the human thought process. The definitions of how these are formed are found in \textit{Die Welt als Wille Und Vorstellung}. Arthur Schopenhauer. Drittes Buch. Seite 316. Deutsche Taschenbuch Verlag GmbH & Co. KG. Muenchen, Germany, 1998. (\textit{The World as Will and Imagination}. Arthur Schopenhauer. Book Three. Page 316. German Pocket Books, Publishers. Munich, Germany, 1998.)
Figure 1. An initial impression of the functional organization of the human thought system.
exploration of deep space. All of space is hostile to human life, in fact, to all of biological life. Mars, Venus, and Jupiter, for instance, are all hostile environments, yet we must explore them because of the categorical imperative: “I will, because I can.” It would be infinitely easier to go to Mars if we had a crew of autonomous systems go ahead of the human crew. The autonomous systems would construct a fully life-supporting facility on Mars, test it, and provide the test data to the crew of the manned spacecraft prior to launch. There are also hazardous places on Earth, such as radioactive contamination sites like Chernobyl, the Hanford Reservation, and Oak Ridge that need cleaning up. Autonomous systems could perform the more dangerous jobs. Finally, there are the handling and disposition of biological and chemical agents, and the handling of unknown viruses and bacteria. All these tasks require autonomous systems that can learn, analyze, and decide on a logical course of action, with the human being still the suggesting superior, but not the controller.

One of the questions a colleague and friend asked me was, “What will we do if we build it, and it turns out that it doesn’t like us?” Good question! I never thought that possibility through, and certainly I have made many mistakes in the past, not contemplating unforeseen possibilities. For example, as a young U.S. Army Special Forces demolitionist in 1962, I used to train with atomic demolitions ammunition. I had to disassemble and reassemble all of the uranium components with only a lead apron, lead gloves, and a pair of goggles to protect me. What all that did to me physically, I have no idea; I was only 24 at the time, and nobody told me about long-term effects.

So yes, we can build an autonomous system. How we control it is a question we have to resolve a priori.

1.8 AN ARCHITECTURE FOR AUTONOMY

The final chapter of this book is the chapter on autonomous systems. From the architect’s point of view it describes the system architecture at a level that we refer to in computer science as a Level I architecture. Every system, by force of imperative, must have the architect’s vision of how the system will look from an initially subjective point of view.

The greatest problem posed to today’s professionals who are contemplating building a truly autonomous system is the architectural design. The reason is that current thinking in computer science is cast into the concrete of hierarchical-sequential logic. In the context that AI and neural networks are attempting to solve the problems posed by autonomy, this approach is unfortunately a “cul-de-sac,” or dead end. Human beings who fall into the “normal” category have a completely dynamic, nonhierarchical, nonsequential, and nonlinear thought and reasoning process. Our “thought system” (our functional architecture), our “external sensory input,” and our “thought process” (data handling) are so dynamic that they cannot be expressed by using simple traditional methods. The traditional approaches to illustrating the architectures of functional relationships are hierarchical and linear; these use the ubiquitous block diagrams and boxes.