PART V: Conclusion: Model Building and the Master Programmer

The limits of my language mean the limits of my world…

— Ludwig Wittgenstein, "Tractatus Logico-Philosophicus"

Theories are like nets: He who casts, captures…

— Ludwig Wittgenstein, "Tractatus Logico-Philosophicus"

The best you can do by Friday is a form of the best you can do…

— Charles Eames, Noted Twentieth Century Designer

We have come to the end of our task! In Part V we will give a brief summary of our views of computer language use, especially in a comparative setting where we have been able to compare and contrast the idioms of three different language paradigms and their use in building structures and strategies for complex problem solving. We begin Chapter 32 with a brief review of these paradigm differences, and then follow with summary comments on paradigm based abstractions and idioms.

But first we briefly review the nature of the programming enterprise and why we are part of it.

Well, first, we might say that programming offers monetary compensation to ourselves and our dependents. But this isn't really why most of us got into our field. We authors got into this profession because computation offered us a critical medium for exploring and understanding our world. And, yes, we mean this in the large sense where computational tools are seen as epistemological artifacts for comprehending our world and ourselves.

We see computation as Galileo might have seen his telescope, as a medium for exploring entities, relationships, and invariance's never before perceived by the human agent. It took Newton and his "laws of motion" almost another century fully to capture Galileo's insights. We visualize computation from exactly this viewpoint, where even as part of our own and our colleagues' small research footprint we have explored complex human phenomena including:

- Human subjects' neural state and connectivity, using human testing, fMRI scanning, coupled with dynamic Bayesian networks and MCMC sampling, none of which would be possible without computation.

- Patterns of expressed genes as components of the human genome. These gene expression patterns are assumed to be at the core of protein creation that enables and supports much of the human animal's metabolic system, including cortical activity and communication.
- Real time diagnostics and prognostics on human and mechanical systems. These complex tasks often require various forms of hidden Markov models along with other stochastic tools and languages.
- Understanding human language and voiced speech also requires computational tools, including various stochastic tools and models. Better language tools will require conditioning such systems with realistic models of human understanding and intention.

Of course this list could go on to include many of the exciting tasks that make up the daily challenges of our readers. What is important is that we see computer programming less in terms of the act of building tools, than as a medium for creating and debugging models of the world – as an epistemological medium.

We feel that there are (at least) two consequences of our thinking of computation as an epistemological medium: First, as programmers we are model builders. We use our data structures and search strategies to capture state, relations, and invariance's in our application domains. We come to understand this domain through progressive approximation. And our domains are rarely static, but change and evolve across time. Thus we often require stochastic engines and probabilistic relationships to capture these complex evolving phenomena.

Second, we explore our world by iterative approximation. When we build a model, we make an approximation of some aspect of reality. The quality of our model building is often seen through the lens of failure. As the philosophers of science continue to remind us, good models are falsifiable. It is through their failure points that we begin to appreciate our own failure to comprehend aspects of the phenomena we wish to understand. When our models are carefully designed and crafted, we can then deconstruct them to address these failure points and attempt to expand our understanding. Our increased understanding is then reflected in the next iteration of our model building. Thus the iterative design methodology, whether used by the individual programmer, or as is more often the case, within the collaborating communities of groups of programmers is a critical methodology in coming to understand our application domains.

We urge the reader to keep these ideas in mind in reading the final chapter and its reprise of the book's main themes of language-paradigm-based abstractions and idioms of the master programmer.