Foreword

Ludwig Wittgenstein once compared a language to a city. In the historic center were gnarly lanes, in the middle were broad avenues and gardens with diverse architecture, and on the edges were geometrically planned suburbs. He was, of course, speaking of what we now call "natural" languages, but the analogy holds to our computer languages as well. We have low-level languages that fit the historic centers. And the boxy modeling techniques we use are the Stalinist apartment blocks in the suburbs.

It's the broad avenues and diverse architecture in between that have evaded the conventions of most of our computer languages. If you look at the workings of the city, there are street layouts, traffic patterns, zoning maps, architectural codes, and landscape maps; in the buildings are structural, plumbing, electrical, telecom, ventilation, and security plans; and in the factories, you'll find more specialized process, fluid, machine, and automation schemas. These are a few of the domain-specific languages of the real world. Each has a rigid set of semantics and an established body of practice. Each has been created because the prior alternatives were insufficient for the tasks at hand.

Of course, people now use computers to draw all of these things. In every case, some enterprising vendors (or occasionally users) have created packages that implement the specific modeling tasks for the domain. The applications have been limited to the domain, and the cost of maintaining the infrastructure has been considerable.

At the same time, in the world of computer systems, the most frequently used design tool is the whiteboard. And there is some kind of

(usually manual and highly tacit) process in which the whiteboard sketches eventually get translated into code. Ideally, this would be a smooth progressive rendering of design, moving from broad concepts to precise code.

Unfortunately, today it's not so smooth. Whether developers use generic modeling languages like UML (in the minority case), or go from dry-erase marker to 3GL, there's always an abrupt shift from the human-readable world of the domain to the computer-executable world of the software. The goal of the Microsoft DSL Tools is to bridge that gap.

What if we could make it as easy to sketch a design in the language of the problem domain as it is to draw on a whiteboard, and then progressively annotate the sketch until it were sufficiently rich to become an executable model? That technology isn't here yet, but the DSL Tools are a huge leap forward.

The DSL Tools democratize the creation of domain-specific languages that can capture high-level design in an idiom familiar to domain experts and transform the designs into running software. This is a big step toward mass customization—the idea of capturing the domain patterns of a family of related software solutions and assembling the specific results from well-defined components. Almost every successful industry has learned to do this, but software has lagged.

When we achieve mass customization, the economics of software will change from the craft era to an age of software supply chains with component marketplaces and well-defined rules for reuse. The DSL Tools will be remembered as a pivotal step in that transformation.

There are no better individuals to write this book than Steve Cook, Gareth Jones, Stuart Kent, and Alan Cameron Wills. They are the creators of the DSL Tools. They have decades of experience in the use and design of prior generations of modeling tools. This depth of knowledge informs a passion and an expertise that can't be matched in the industry. Their work is a great contribution.

Sam Guckenheimer
Author, *Software Engineering with Microsoft Visual Studio Team System*
Redmond, WA
March 2007