Applications of Formal Methods: Developing Virtuoso Software

Susan L. Gerhart, MCC Software Technology Program

Formal methods, formal specification, formal verification!

These terms are widely bandied about and have acquired a variety of meanings to mathematicians, engineers, computer scientists, and logicians. The September issues of Computer, IEEE Software, and IEEE Transactions on Software Engineering all contain articles that define and illustrate this area of research and show how it is maturing into practice.

Before getting to the articles, I'd like to offer some personal observations on the transition of formal methods into industrial practice.

International comparison. The international disparity in uptake of formal methods is striking. Make no mistake about it: Only a small percentage of practitioners even in the country with the widest use — Britain — has ever seen a formal method, let alone used one. But European progress in this area is far more rapid than in North America.

Three kinds of US approaches have evolved but with less industrial application than in Europe.

The first one — the tools-heavy, security-driven approach — has produced a vibrant research community interested in the challenging problems of modeling security and building secure systems. But, to date, few lessons have been carried over to other software-engineering applications, where the problems are different and more attention must be paid to costs.

The second, academic approach, which focused on programs and languages, has provided useful training for some practitioners but has faced severe problems in scaling up and in its lack of attention to real-world development efforts.

The third major approach, IBM's Cleanroom, is a complex process model involving statistical testing, verification, functional specification, and reviews. It is usable only with major organizational commitment to
Formal methods’ transition is culturally and economically based, not simply technology-driven.

This tutorial is a careful blend of theory and pragmatism centered around the notion of a formal specification. Even those working in the field for many years will find a useful framework here. This tutorial is specification-centered, unlike other frameworks, which tend to be organized around verification, tools, theories, or specific problem domains (like communication protocols and security). Through this tutorial, you should gain a feeling for formal methods as a research area, as an engineering approach, and as an intersection of several branches of computer science.

So, now that you’ve read, or at least glanced through, the tutorial, you may also be interested in *IEEE Transactions on Software Engineering*, where 11 research papers cover topics in more depth than the tutorial; these topics include specification languages and analyzers, foundations, and techniques.

This issue of *IEEE Software* covers the application of formal methods across a range of challenging software and hardware formalizations and shows what you can gain from doing so.

The first article, Anthony Hall’s “Seven Myths of Formal Methods”, identifies several widely held conceptions about formal methods with which he strongly disagrees, based on his experience. This article highlights the fact that Europeans and North Americans have historically emphasized different approaches to formal methods. Oversimplifying, the Europeans tend to work without tools and to focus on specifications while the North Americans tend to work with tools and to focus on proofs.

Hall’s article addresses the bounds of formal methods, identifies the central role of specifications in the development process, and covers education and training. Some readers will take exception to his “facts,” but bear in mind that these are facts against the background of the house methods used at his company. The application sketched is a large tool set built under the object-oriented paradigm.

J. Michael Spivey’s “Specifying a Real-Time Kernel” illustrates the application of formal methods to a safety-critical system. The objective of this study was to improve the existing documentation of a diagnostic X-ray machine to serve later implementations. The separation of the kernel from applications had an interesting consequence: It helped identify a design flaw in the kernel that could have caused damage by the X-ray application. Other applications might have stumbled over the same flaw, which informal documentation would not have exposed. The article is a good example of the use of the Z (pronounced “Zed”) notation and its methods for modeling systems. Spivey delineates the limitations of this specification, providing a good challenge for other specification techniques to tackle the remaining properties, like real-time performance, for completeness and comparison.

The article by Norman Delisle and David Garlan, “Specification of an Oscilloscope,” follows the theme of using specifications to gain insight into hardware/software systems. You might ask, “Oscilloscopes! Aren’t those just simple, well-understood hardware components?” No. Today’s oscilloscope is a complicated, software-controlled instrument about which specification techniques (in this article, Z) offer the potential of improving user-level models as well as helping the oscilloscope builders find better designs. This nontraditional use of formal methods in a specific domain suggests that other industrial applications are ripe for similar investigations into reusable frameworks.

The American and other national-security communities have been heavily involved with the evolution of formal meth-
From the requirements of a terminal service, for checking specifications and implementations against models of security properties. Richard Kemmerer's "Integrating Formal Specification and Verification Techniques" is both an illustration of how formal methods are applied in a security context and a case study of their use in several phases of development, starting from the requirements of a terminal serving a security officer, to formal requirements and design expressed as state transitions and showing that these agree, through detailed design specifications and proofs of these agreement with higher level specifications, stopping just before code-level verification (due to complications typical of such projects). This article shows many aspects of the North American style of applying formal methods to critical systems.

Mandayam Srinivas and Mark Bickford's "Formal Verification of a Pipelined Microprocessor Architecture" describes the application of modern functional languages and supporting verification technology to a scaled-down but realistic microprocessor. The model is of an infinite stream of machine instructions consuming an infinite stream of interrupt signals and is specified at two levels: instruction and hardware design. A correctness criterion is stated for an appropriate sense of equivalent behavior of these levels, and this criterion is then proved using a mechanically supported induction argument. Efforts like this are leading to an understanding of the generic parts of applying formal methods to microprocessor design — indeed, possibly to a stable proof pattern that can be followed in industrial practice as an alternative to simulation.

This issue's formal-methods section concludes with a short article on the use of formal methods in standards. The British Computer Society has been particularly active in using various formal notations, including those for graphics and document systems. This article explains why formal methods should be part of the standards process and describes some standards on which they have been used.

**Biases and context.** There are several biases to note in the articles here, most related to the range of articles submitted:
- No articles were submitted from the communication-protocol field, where tools and notations are very widely used.
- Z was the predominant notation in the high-quality articles submitted, reflecting the extensive efforts to apply Z to numerous examples in addition to its most well-known application (to IBM's CICS transaction processing system). The other major European formal method, VDM, has been applied to several significant examples in language definition and a range of systems components. Several notations popular in the US appear in the papers in *IEEE Transactions on Software Engineering*. However, no articles represent Harlan Mills's Clearroom or David Parnas's brand of formal methods, both of which use formal methods in the context of a process involving significant usage review and testing, as well as proof. (Mills and Parnas will both have articles about their approaches in the November issue.)
- Some CASE and petri-net articles were submitted, but they did not address examples of comparable significance in terms of the specification and verification issues of the selected articles. Papers on these areas are often published in *IEEE Transactions on Software Engineering* and other journals.
- Most of the examples were slanted toward hardware and fixed-function systems, rather than the softer upstream areas where trade-offs and ambiguities reside. This may indicate convergence toward the highest payoff areas of formal methods, since less formal methods apply better to the more upstream problems. On the other hand, it may be intrinsically difficult to portray any significant use of formal methods in the always-confusing process of requirements analysis for systems at the user level. And it may be that hardware specification is simply more mature than software specification. There are also ways to check hardware circuit designs that use different mathematical formalisms and proof techniques; this is not true for software.
- No example uses very complex mathematical structures — no algebras, logical theories, or deep theoretical properties. To some extent, these lie hidden within the notation definitions and are brushed over in the formalizations. Perhaps the applications have not been forced to face mathematics beyond the grasp of a college sophomore, but this might also be a testimony to the power of the notations chosen and the maturation of the field in sorting out its presentation techniques.

The authors of this issue's articles are, of course, advanced masters of the techniques involved, but, in offering these articles, they are also teaching their experience to others who can apply them to their own problems. The domain knowledge and its acquisition are also difficult to portray in these articles — the properties of oscilloscope waves and kernels, for example, are proper and deep domains in and of themselves. The method and its application target are two distinct subjects, requiring a good grasp of each for a successful application.

**Formal methods are not a panacea but a step that must be taken to put software engineering on solid footing.**

Even within the field, formal methods are not viewed as any panacea but as a step that must be taken to put software engineering on the same solid footing as any other engineering field. While the specific techniques to choose and use will certainly vary, the key ideas are now in place:
- notations based on discrete mathematics to be used for system description, design, and analysis;
- formalization as a process applicable to whole classes of systems, applying effectively to even behavior of fielded products; and,
- eventually, technology support based on symbolic mathematics and what we now call programming environments.

Having demonstrated so widely the re-
markable descriptive powers of simple mathematical notation and the opportunities for formalization, the next challenge is to integrate these formal methods with the variety of informal techniques (like design records, conceptual modeling, and graphical representations) also required to achieve the goal of a formally based engineering discipline.

Acknowledgments
I thank the reviewers of the articles submitted to this special issue and the other authors who submitted articles. It's been a pleasure to work with Nancy Leveson, guest editor for the IEEE Transactions on Software Engineering complementary issue.

And, now that I'm an "experienced" guest editor, I offer my apologies to all the other editors for whom I have, unfortunately occasionally, been a deadbeat, evasive, tardy reviewer. Now I know what inefficiency that adds to the reviewing process. My motto now is "If I can't for sure see a time during the next month or if I have any reservations about my qualifications or interest in the article, I'll send it back to the editor." There's always somebody else almost as qualified to review the article, and the editor will be far better off finding an on-time second choice than a late or never first choice.

References

Susan L. Gerhart is leading a project at MCC to transition formal methods into North American industry and to develop technology to integrate formal methods with informal material and advanced validation techniques. Previously, she was a faculty member at the Wang Institute of Graduate Studies and at Duke University and was a technical staff member at the University of Southern California's Information Science Institute.

Gerhart has a BA in mathematics from Ohio Wesleyan University and a PhD in computer science from Carnegie Mellon University. She is a member of the IEEE, Society of Women Engineers, and IEEE Software editorial board.

Address questions about this article to the author at MCC Software Technology Program, 3500 W. Balcones Center Dr., Austin, TX 78759; Internet gerhart@mcc.com.