The Pragmatics of Model-Driven Development
by Bran Selic, pp. 19–25. The potential benefits of using models are significantly greater in software than in other engineering disciplines because of the potential for a seamless link between models and the systems they represent. Unfortunately, models have rarely produced anticipated benefits. The key lies in resolving pragmatic issues related to the artifacts and culture of the previous generation of software technologies.

What Models Mean
by Ed Seidewitz, pp. 26–32. Models must be more than informal pictures. By carefully considering a model’s relationship to the thing being modeled and to other models derivable from it, we can understand how to use models to reason about the systems we build and how to use metamodels to specify languages for expressing models.

UML without Pictures
by Conrad Bock, pp. 33–35. This article reviews the concepts of repository-centered development with UML, explaining how notation, semantics, and model compilation relate. UML can be used in many formats, including presented as text, parsed into a standardized repository, and compiled to multiple programming languages. The example given uses the UML 2 repository.

Model-Driven Development: A Metamodeling Foundation
by Colin Atkinson and Thomas Kühne, pp. 36–41. Metamodeling is an essential foundation for MDD, but there’s little consensus on the precise form it should take and role it should play. The authors analyze the underlying motivation for MDD and then derive a concrete set of requirements that a supporting infrastructure should satisfy. They discuss why the traditional “language definition” interpretation of metamodeling isn’t a sufficient foundation and explain how it can be extended to unlock MDD’s full potential.

Model Transformation: The Heart and Soul of Model-Driven Software Development
by Shane Sendall and Wojtek Kozaczynski, pp. 42–45. The model-driven approach can increase development productivity and quality by describing important aspects of a solution with human-friendly abstractions and by generating common application fragments with templates. This article examines different approaches to model transformations and recommends desirable language characteristics for describing them.

Model Metamorphosis
by Torben Weis, Andreas Ulbrich, and Kurt Geihs, pp. 46–51. MDD will become practical only if transformations from platform-independent to platform-specific models can be largely automated. Kafka, an extensible rule-based transformation language, and the Kase modeling tool enable automated, customized transformations that support round-trip engineering. With Kafka and Kase, developers can construct transformations without detailed knowledge of metamodels.

A Metamodeling Approach to Pattern-Based Model Refactoring
by Robert France, Sudipto Ghosh, Eunjee Song, and Dae-Kyoo Kim, pp. 52–58. Design patterns capture development solutions to design problems in forms that make the designs more modular, modifiable, reusable, and understandable. This metamodeling approach to pattern-based refactoring of design models incorporates the precise specification of design patterns and transformation rules.

Model-Driven Integration Using Existing Models
by Peter Denno, Michelle Potts Steves, Don Libes, and Edward J. Barkmeyer, pp. 59–63. Models can describe business transactions in terms of business entities and help automate some systems integration tasks. The joint action model describes a new business transaction motivating systems integration.

Separation of Concerns in Model-Driven Development
by Vinay Kulkarni and Sreedhar Reddy, pp. 64–69. MDD has improved productivity, quality, and platform independence but hasn’t been that successful at supporting reuse and system evolution. To do so, a system must be specified as a composition of multiple views corresponding to stakeholders and their concerns. The proposed Template abstraction addresses this problem in an integrated way, dealing with separation of concerns at both the model and the code level.

Trade-offs between Productivity and Quality in Selecting Software Development Practices
by Alan MacCormack, Chris F. Kemerer, Michael Cusumano, and Bill Crandall, pp. 78–85. The authors report detailed data and analyses on productivity and quality from 29 Hewlett-Packard projects. While some software development models’ characteristics affect performance negatively when considered alone, their impact disappears when considered in combination with other attributes. So, processes should be thought of as coherent systems of activities rather than as a series of individual practices that can be implemented piecemeal.

Software Process Improvement: Blueprints versus Recipes
by Ivan Aaen, pp. 86–93. Viewing software processes as blueprints emphasizes that design is separate from use, and thus that software process designers and users are independent. In the approach presented here, software processes are viewed as recipes; developers individually and collectively design their own software processes through facilitation, reflection, and improvisation.