A Unified Model of Dependability: Capturing Dependability in Context
by Victor Basili, Paolo Donzelli, and Sima Asgari, pp. 19–25. Today, individuals and organizations increasingly depend on services delivered by sophisticated software-intensive systems. Dependability, a key systems property, must be guaranteed regardless of continuous, rapid, and unpredictable technological and context changes. However, in different circumstances, dependability can imply different system attributes. In this perspective, the authors introduce the Unified Model of Dependability, a modeling language that lets you reason about dependability and turn it into clearly defined and implementable system properties.

Automated Management of Multicustomer Code Bases
by Nicolás Serrano, Ismael Ciordia, Sonia Calzada, and Fernando Alonso, pp. 26–31. For applications such as management information systems, customizing the code base to meet each new customer’s needs can sometimes be unavoidable. Although good design can minimize the need for such code-level modifications, it can’t eliminate it. Moreover, if you don’t carefully and accurately manage customizations when they do occur, the result can be skyrocketing development costs, poor customer support, and an inability to respond quickly to new needs. The authors developed a highly automated, model-driven approach for maintaining multicustomer code bases.

Preserving Distributed Systems’ Critical Properties: A Model-Driven Approach
by Arvind S. Krishna, Cemal Yilmaz, Atif M. Memon, Adam A. Porter, Douglas C. Schmidt, Aniruddha Gokhale, and Balachandran Natarajan, pp. 32–40. As information users move increasingly toward distributed systems to supply basic information infrastructure, the ability of such systems to guarantee basic properties over time increases in importance. This article describes a model-based process for efficiently coordinating the use of distributed-computing resources to help identify viable configurations of systems and software for meeting stringent quality-of-service and PSA requirements. This process, called Skoll DCQA, provides an efficient approach to navigating the very large space defined by all possible system and software settings in a modern system.

Separating Adaptable Persistence Attributes through Computational Reflection
by Francisco Ortín, Benjamin Lopez, and J. Baltasar García Pérez-Schofield, pp. 41–49. A prototype persistence system based on the nitro reflective system demonstrates that computational reflection is a suitable technique for dynamically and transparently adapting any application’s persistence attributes, following the separation-of-concerns principle. Because computational reflection offers the runtime customization of language semantics, it lets developers combine programming language and database management systems into a single computational model.

Identifying Requirements Conflicts and Cooperation: How Quality Attributes and Automated Traceability Can Help
by Alexander Egyed and Paul Grünbacher, pp. 50–58. Identifying conflicts and cooperation among existing and new requirements during software development and maintenance is an important challenge. Quality attributes can help, and automated traceability techniques can efficiently eliminate falsely identified conflicts and cooperation. The authors demonstrate their approach in the context of a simple video-on-demand system.

Open Source Reuse in Commercial Firms
by T.R. Madanmohan and Rahul De’, pp. 62–69. The decision to use open source components raises a spectrum of issues, from requirements negotiation to product selection and integration. To explore these issues from an empirical perspective, the authors conducted a study based on structured interviews with the project leads of 12 firms. Their goal was to understand open source component use in the trenches—that is, how it played out in the development process for software developers and managers day to day.

Piloting XP on Four Mission-Critical Projects
by Jerry Drobka, David Nofz, and Rekha Raghu, pp. 70–75. Software development teams face a continuous battle to improve productivity while maintaining or improving quality. Often, a project’s mandated ship date requires that software development begin with only a portion of the requirements defined. Recently, several Motorola software development teams piloted Extreme Programming to mitigate the risk of changing requirements and improve productivity.

COTS Component Acquisition in an Emerging Market
by Pauliina Ulkuniemi and Veikko Seppänen, pp. 76–82. Component-based software engineering has generated much interest both among researchers and in industry. Using software components offers many benefits but also poses several problems, particularly for business practices. The software component market is far from mature and lacks industry standards and management guidelines. These emerging market dynamics have important implications for those managing COTS software component acquisitions.