

# article summaries

MARCH/APRIL 2004, VOLUME TWENTY-ONE, NUMBER TWO

## RE'03: PRACTICAL REQUIREMENTS ENGINEERING SOLUTIONS

### Ongoing Requirements Discovery in High-Integrity Systems

by Robyn R. Lutz and Inés Carmen Mikulski, pp. 19–25. The discovery of new requirements and new requirements knowledge continues throughout the lifetime of many high-integrity embedded systems. Anomaly reports from testing and operations on eight spacecraft at the Jet Propulsion Laboratory revealed four mechanisms for how we discover and resolve requirements. Understanding these mechanisms identifies guidelines to help prevent anomalies found during testing from recurring during operations.

### ERP Requirements Engineering Practice: Lessons Learned

by Maya Daneva, pp. 26–33. Organizations implementing enterprise resource planning systems have been increasingly adopting generic, off-the-shelf requirements engineering process models. Yet, little information exists about the challenges of making a generic RE model a life process. This article discusses why bringing in an off-the-shelf process isn't a sufficient foundation. The author shares experiences acquired over the past five years of eliciting, modeling, and validating requirements in ERP projects and highlights typical issues and their solutions. Among the keys to success are planning RE model use in the client's context and installing processes to support key RE activities.

### Executable Use Cases: Requirements for a Pervasive Health Care System

by Jens Bæk Jørgensen and Claus Bossen, pp. 34–41. Executable use cases combine prose, formal models, and animation to represent new work processes and their proposed computer support. The represen-

tation enables stakeholders to make interactive investigations of a system's requirements in the context of envisioned work processes. The authors introduce executable use cases and describe how they applied them in requirements engineering for a new pervasive health care system for hospitals in Denmark.

## DISTRIBUTED SYSTEMS

### Embracing Dynamic Evolution in Distributed Systems

by Kam Hay Fung, Graham Low, and Pradeep Kumar Ray, pp. 49–55. Dynamic evolution is an important phenomenon in distributed systems that undergo changes without being shut down. Several technologies are available that can support dynamic evolution, but existing software engineering methodologies will need to be extended to fully exploit this process.

### Distributed Programming with Typed Events

by Patrick T. Eugster and Rachid Guerraoui, pp. 56–64. Type-based publish-subscribe (TPS) is an appealing candidate programming abstraction for decoupled and completely decentralized applications that run over large-scale and mobile networks. TPS enforces type safety and encapsulation while providing decoupling and scalability properties. Two TPS implementations in Java demonstrate this approach's potential. One relies on specific primitives added to the Java language; the other relies on a library implementation based on more general Java mechanisms.

## FEATURES

### Understanding Service-Oriented Software

by Nicolas Gold, Andrew Mohan, Claire Knight, and Malcolm Munro, pp. 71–77. Service-oriented software is being hailed as

the next revolutionary approach to software development. It allows organizations to rapidly form new software applications dynamically to meet changing business needs, thus alleviating the problems of software evolution that occur with traditional applications. The largest of these problems is understanding the existing software before it is changed. This article looks ahead at the issues involved in the automated construction of service-oriented software. While service orientation certainly helps solve some aspects of the evolution problem, software comprehension takes a new and potentially more challenging role.

### The Separation Principle: A Programming Paradigm

by Yasushi Kambayashi and Henry F. Ledgard, pp. 78–87. The Separation Principle, considered by the authors a new programming paradigm, is a simple and natural way of constructing programs. Its intuitive nature and ease of use make it easy to implement many different software designs. By expressing data-sharing, parameter-passing, and scope issues in simple, canonical drawings, the Separation Principle makes it easy to understand the full range of relationships between data and instructions. A preliminary study shows that this paradigm improves programs' understandability.

### Overlooked Aspects of COTS-based Development

by Marco Torchiano and Maurizio Morisio, pp. 88–93. Commercial-off-the-shelf-based development involves many issues, from requirements negotiation to product selection and product integration. Studies on this topic often disagree, lack product and project details, and are founded on uncritically-accepted assumptions. This exploratory study, based on interviews and a literature review, establishes key features of industrial COTS-based development and creates a definition of "COTS products" that captures these features.