Integrating COTS into the Development Process

Alexander Egyed, *Teknowledge Corp.*

Hausi A. Müller, *University of Victoria, Canada*

Dewayne E. Perry, *The University of Texas at Austin*

Building and evolving software systems is an arduous, costly, lengthy, and complex task. The resulting systems are similarly complex. We're thus constantly searching for ways to reduce such costs, time, and complexity while increasing system functionality and quality. Not surprisingly, our quest for "silver bullets," as described by Frederick Brooks back in 1987,[1] hasn't produced significant

improvements. Instead, we've achieved steady, incremental improvements in the enterprise of building and evolving software systems.

The complexity of "simple"

One strategy that seemed promising when Brooks wrote his article was the notion of "buy not build." Using COTS products is one way to implement this strategy, because software development then becomes the process of "simply" integrating COTS components. However, it turns out that dealing with COTS is a high-risk activity for a variety of reasons.

The initial manifestation that integrating is something other than a simple integration activity appears in the form of the *architecture mismatch* problem.[2] COTS components make several assumptions about architectural issues. Then, when these assumptions conflict or don't match, the simplicity of using COTS quickly disappears in a cloud of complexity and integration scaffolding. The root of the problem is the lack of access to the components' source code and developers.

Moreover, we seldom find ourselves in a pure COTS component integration environment, because we usually use a hybrid of specially developed components mixed with COTS components or products. Thus, instead of an endgame activity, using COTS products becomes an integral part of the entire software development life cycle. It transcends social, economic, and development concerns and affects all traditional aspects of software development—requirements engineering, architecture, design, implementation, testing, and long-term maintenance.

Over the past decade, data- and control-integration mechanisms and standards have matured significantly. Moreover, COTS products have become end-user programmable, extensible, and interoperable. Although we shouldn't hype the combination of these improvements as a silver bullet, they have led to new avenues for incorporating COTS into software systems.

Innovative COTS integration

The idea for this special issue of *IEEE Software* grew out of two recent COTS workshops. The first was the International Workshop on Incorporating COTS into Software Systems (IWICSS), held at the International Conference of COTS-Based Software Systems (ICCBSS 2004). The motivation behind IWICSS was to address the integration of COTS into the entire development life cycle. The second was the Fourth Workshop on Adoption-Centric Software Engineering (ACSE), held at the International Conference on Software Engineering (ICSE 2004). The motivation behind the series of ACSE workshops was to investigate how to extend COTS products to build software engineering research tools. Users are more likely to evaluate and adopt such research tools if they're already familiar with the host COTS, so we need to leverage the COTS products' functionality, cognitive support, and interoperability.

The focus articles in this issue explore innovative ways of integrating COTS products into software systems for purposes often unimagined by their creators. They investigate the challenges, risks, and benefits of building COTS-based software systems.

"An Active Architecture Approach to COTS Integration," by Brian Warboys, Bob Snowdon, R. Mark Greenwood, Wykeen Seet, Ian Robertson, Ron Morrison, Dharini Balasubramaniam, Graham Kirby, and Kath Mickan, presents a mechanism for recognizing COTS products as integral parts of any information system's environment. The authors' ArchWare framework addresses the integration problems of COTS and other components using an active architectural model that captures the composition and integration of these elements and changes as the system evolves. The goal is to manage both the predicted and emergent changes that result from using COTS components as elements in a system's architecture.

Addressing the critical issue of coordinating COTS components is the focus of the next article, "Coordinating COTS Applications via a Business Event Layer," by Wilfried Lemahieu, Monique Snoeck, Frank Goethals, Manu De Backer, Raf Haesen, Guido Dedene, and Jacques Vandenbulcke. Recognizing the need for a dynamic structure within which components can execute and interact, the authors raise the level of the coordination mechanisms. Moving away from the message-oriented middleware prevalent today, the authors use a more appropriate level of abstraction—namely that of business events. These events reside in the problem space and provide an appropriate domain-specific model for coordination and concurrent execution.

Using COTS products transcends social, economic, and development concerns and affects all aspects of development.

About the Authors

Alexander Egyed is a research scientist at Teknowledge Corp. His research interests include requirements engineering, incremental and iterative software modeling (transformation and analysis), traceability, and simulation. He received his PhD in computer science from the University of Southern California. He's a member of the IEEE, IEEE Computer Society, ACM, and ACM SIGSOFT. Contact him at Teknowledge Corp., 4640 Admiralty Way, Ste. 1010, Marina Del Rey, CA 90292; aegyed@ieee.org.

Hausi A. Müller is a professor of computer science and director of the Bachelor of Software Engineering Program at the University of Victoria, Canada. He is also a visiting scientist at the Center for Advanced Studies at the IBM Toronto Laboratory and the Carnegie Mellon Software Engineering Institute. He's a principal investigator and chair of the technical steering committee of Canada's Consortium for Software Engineering Research. His research interests include investigating methods, models, architectures, and techniques for autonomic computing applications. He also concentrates on building adoption-centric software engineering tools and on migrating legacy software to autonomic and network-centric platforms. He received his PhD in computer science from Rice University. He's on the editorial board for *IEEE Transactions on Software Engineering* and is vice chair of the Technical Council on Software Engineering. He's also a member of the IEEE and ACM. Contact him at hausi@cs.uvic.ca.

Dewayne E. Perry is a professor and the Motorola Regents Chair in Software Engineering in the Department of Electrical and Computer Engineering at the University of Texas at Austin. His research interests include rigorous empirical studies, transforming requirements into architecture, architecture modeling and evaluation, and program analysis. He received his PhD in computer science from Steven's Institute of Technology. He's a member of the IEEE and ACM. Contact him at The Univ. of Texas at Austin, Electrical and Computer Eng. Dept., Mailstop C0803, Austin, TX 78712; perry@ece.utexas.edu.

the way it is for other assets that we use and reuse. In "Evaluating COTS Components Dependability in Context," Paolo Donzelli, Marvin Zelkowitz, Victor Basili, Dan Allard, and Kenneth N. Meyer provide an empirical COTS evaluation process that addresses risk issues by focusing on the context in which the component will be used. A case study illustrates their approach.

Traditional process models assume that development and evolution processes are largely based on a fresh beginning and internal control of the architecture, design, and source code. Based on their experiences of empirically analyzing COTS-based applications, in "Value-Based Processes for COTS-Based Applications," Ye Yang, Jesal Bhita, Daniel N. Port, and Barry Boehm describe a set of value-based processes to minimize the risks of integrating COTS components into an application. They provide guidelines, an associated framework, and a set of processes to support COTS-based application development.

Another prominent issue in software systems is performance, which is typically addressed at the architecture, design, and implementation levels. Maximizing performance usually requires making trade-offs in the various components when the code is available for manipulation, but this is difficult with COTS components. In "Performance Techniques for COTS Systems," Erik Putrycz, Murray Woodside, and Xiuping Wu use new tracing techniques as the basis for building performance models of COTS components. These models can then be used when reasoning about performance trade-offs in the architecture planning, design, implementation, and deployment stages of built systems incorporating COTS components.

A variety of issues are involved in selecting appropriate components for a system, whether they're pre-existing product-line assets, COTS components, or COTS products. As in performance, the problem is exacerbated by the fact that the code isn't available for evaluation

In the future, COTS products will play an increasingly important role in not only software product engineering but also software engineering tool development. Over the past decade, we've witnessed a consolidation of data, control, and presentation integration techniques, which will significantly ease the integration difficulties of COTS. Investigating the challenges and risks involved in leveraging COTS products in systems and tools will be a hot research topic for years to come. 🎵

References

1. F.P. Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, vol. 20, no. 4, 1987, pp. 10–19.
2. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch or Why It's Hard to Build Systems out of Existing Parts," *Proc. 17th IEEE/ACM Int'l Conf. Software Eng.* (ICSE 95), IEEE CS Press, 1995, pp. 179–185.