

Oh Dear, We Bought Our Competitor: Integrating Similar Software Systems

Rikard Land and Ivica Crnković, Mälardalen University

// How do you transition from several functionally overlapping systems to just one? A look at 10 case studies addresses the technological, personnel, and organizational challenges. //



WHAT WOULD YOU do if you realized your company owns two similar software systems—that is, it holds complete rights and control of two systems intended to fill the same user needs? You might first ask how the company came to be in such an awkward position.¹ But actually, this situation is typical after a company merger or acquisition or when a large organization realizes that two or more of its divi-

sions have been independently developing systems that address the same problem in different ways.

Your second reaction—or your management's—might be to attempt a tight integration and merge the systems. This is usually inadvisable. Rather than integrate the systems at the implementation level, it's normally better to reuse certain experiences and design solutions to evolve one system and retire

the other. Or, this might be an opportunity to reuse knowledge from the existing systems and develop a completely new system, with little or no code reuse. None of these options is easy or inexpensive—quite the opposite: they all require considerable commitment and bring many nontechnical challenges related to personnel, cultures, and organizations.

We present 10 industrial cases in which such a situation existed. They demonstrate the delicate balance between making integration decisions too early and too late. They also reveal the challenge of involving the right personnel at the right time and of garnering and maintaining commitment throughout the organization. Additionally, some cases are surprising stories of how independently developed systems might be quite similar.

Cases

The 10 cases involved seven organizations in different business sectors (see Table 1). Our data collection methods included our participation in projects, several rounds of interviews with project leaders and software architects, and several rounds of questionnaires with software architects and project managers, as well as project and product documentation.² Companies we studied included ABB, Bombardier, Ericsson, Saab, and Westinghouse. However, we can't disclose detailed information or relate case descriptions to specific companies or systems. Our observations regarding cultural influences might be skewed because all the organizations involved Sweden and other European or North American countries.

The Four Strategies

On the basis of the research literature

The 10 industry cases of systems integration challenges.

Organization	Countries	Case		
A	Newly merged international company	Global—predominantly Sweden and Germany	Human-machine interfaces of similar safety-critical systems with embedded software, developed by several previous competitors, were to be integrated and evolved into a platform.	
B	Daughter companies in a large corporation	Sweden	Two administration systems that were developed in-house to keep track of goods in the corporation were to be integrated.	
C	Newly merged international company	US and Sweden	Similar safety-critical products with embedded software, developed by two previous competitors, were to be integrated.	
D	Newly merged international company	US and Sweden	Two previous competitors had similar client-server products for offline management of power distribution systems.	
E	Collaboration between a Swedish government agency and industry	Sweden	E1	A new generation of models was needed for certain kinds of simulations. They were to be based on existing separate but functionally overlapping models.
			E2	Three simulation systems with significant functional overlap, built by different units of a Swedish government agency, were to be integrated.
F	Newly merged international company	US and Sweden	F1	Three systems for managing offline simulations, developed and mainly used internally by two previous competitors, were to be integrated.
			F2	The two previous competitors had two systems developed in-house for the same kind of simulations. The current systems needed improvement, and clear potential existed for integration.
			F3	Three systems for reporting software problems within the company were to be reduced to one common system.
G	Company acquisition	Sweden	Two similar publishing systems were to be integrated.	

and the 10 cases, we identified four strategies: *loose integration*, *merge*, *choose one*, and *start from scratch*. The last two don't actually integrate the systems; the business goals of integration are sometimes better achieved without integrating existing systems. In such cases, organizations can reuse experiences, design solutions, and so forth without reusing the actual implementations.

Loose Integration

For data-centric information systems, the most common type of integration creates adaptors and synchronization mechanisms such that data insertions or updates in one system automatically propagate to the other.^{3,4} Standardized solutions such as middleware facilitate such loose integration, and commercial consultancy services and middleware

products are a flourishing business.⁵

However, loose integration isn't a viable option for many other kinds of software systems, such as those that are interaction- or operation-oriented, that require a homogeneous user interface, or that have embedded software that requires fast response and has memory constraints.⁶ So, we investigated situations in which loose integration isn't the obvious or appropriate choice. However, loose integration can help keep data consistent while keeping user interfaces separate. Of course, if the overlap in functionality is too large, maintaining two systems (plus the added adaptor) in parallel isn't feasible in the long term.

Merge

In theory, reusing the best parts of existing systems—assembled in a new

system—reduces implementation time and cost while ensuring proven quality. In practice, however, this isn't so easy.

The degree of incompatibility between the systems' architectures can disqualify the merge strategy⁷ (an insight bought with a high price in case C; see Table 2). Many potential incompatibilities exist; there might never be an exhaustive taxonomy of them.⁸ In our case studies, we saw three main types of similarities and differences:

- *data models* (similarities in cases D and F2; differences in B, C, E1, E2, and F1),
- *technologies used* (similarities in E2, F2, and F3; differences in A, E1, F1, and F3), and
- *architectural structures* (similarities in D, E2, and F2; differences in B, C, and F1).

The cases' current status.

Case	Result
A	After some internal debate and perceived collaboration difficulties between the office sites, the company launched a new development project and assigned it to one of the sites. This project inherited solutions and knowledge from the previous systems' design.
B	It was imperative that one site perform development and maintenance. So, the corporation chose the larger system, after which it gradually reimplemented the other system's functionality within the new system's framework.
C	At first, management decided that for political reasons, equal parts from the two systems had to be reused and merged, within six months. Architects and developers on both sides felt this was totally unrealistic. After a period of mutual suspicion, no progress, and some contradictory decisions, management chose the architecture and most of the implementation of one system and adapted some smaller parts of the other system to fit it.
D	The company has chosen one human-machine interface and is retiring the other. A strategy exists to merge the servers in the long term, but this hadn't been planned in any detail at the time of the study.
E1	The collaborators all considered the technologies to be out of date and desired to use new technology. So, they launched and successfully executed a new development project.
E2	No solution seemed optimal: too few resources were allocated to do anything constructive. So, the collaborators chose the most complete system and discontinued the other, although users thus lost some crucial functionality.
F1	The company made a relatively early decision: because no party would accept a decision requiring abandonment of its own system, all systems should remain in use, although very loosely integrated. However, the architectures were totally different, and no one perceived any major benefits of integration. So, no part of the company was committed to the decision, and no site fulfilled its share of the integration part of the decision. Later, the company decided that the developers should chose one of the systems. However, they couldn't agree on any solution involving the retirement of the system with which they were already familiar. At the time of this study, no final decision had been made.
F2	The architectures are very similar, and the company is merging the systems by picking parts of each, together with adaptations and new development (see the "Case F2: A Merge in Action" sidebar). When this study ended, the integration was progressing steadily, the main challenge being to align customer-driven delivery projects with the internal goal of a single, integrated system.
F3	Because the company didn't consider this type of software its core business, it acquired and customized a commercial system. This required aligning the three acquired organizations' issue management processes, which was difficult.
G	Both systems were considered well designed and successful on the market. Management realized that a decision was necessary and wanted a short transition. So the company retired the acquired system and offered transition solutions essentially for free to existing customers.

Considering the compatibilities that existed in the cases, it's not surprising that D and F2 successfully chose the merge strategy or that C failed in the initial merge attempt (or that the remaining cases generally avoided a merge). The sidebar provides an in-depth study of F2.

Fortunately, our case studies suggest that two systems needing integration commonly have similar structures. This might be surprising and seem counter-intuitive. However, in exploring further, we found three possible explanations for this similarity.

First, a specific domain typically has

one well-known way of building systems. Similar hardware structures are often due to the products' physical nature (for example, A and C). The historical context in which systems of a particular type were first created results in recurring design solutions (D, E2, and F2). For example, in F2, Fortran was the natural choice of implementation language for both systems because this decision was originally made—independently for both systems—in the 1970s.

Second, standards and legislation exist for systems in the same domain, and these define part of the solution (A, C, and F2).

Third, systems often have a common ancestry—consider, for example, all the Unix variations. Similarly, earlier collaborations between organizations could result in the systems having common design and (possibly) implementation features. In D and F2, the systems to be integrated had a common ancestor some 20 years earlier, and the fundamental choices regarding technology (in both cases) and client-server architecture (in case D) still remained (although much had changed otherwise).

Despite the existence of similar structures, many stakeholders view a merge as a suboptimal compromise

CASE F2: A MERGE IN ACTION

The two simulation systems of case F2 (see Figure A1) have remarkably similar architectural structures, technology choices, and data models. They each consist of four Unix programs run in a batch sequence: a preprocessor, a 2D simulator, a postprocessor, and a 3D simulator. Moreover, both are written in Fortran and have surprisingly similar internal structures. Figure A2 shows the systems' current status; some parts are new and some shared.

The systems' 3D simulators are subject to a merge. That is, the company is replacing some of each system's internal components with either the other system's components or new components. The most important aspect is that the systems use a common data model (which is central to such systems). Previously, data in the programs was implemented in Fortran as large, untyped memory blocks. The company took this opportunity to improve the data structure design and implementation, introducing the ability to perform stronger type and array boundary checks as well as data-access control. It's adapting the two systems' simulation engines to use this data model and data-access component. Most likely, the company won't merge the 3D simulators completely; there might be a stronger business case for maintaining two variants for somewhat different markets and purposes. Suddenly, this merge has become a transition to a new product line.¹

Reference

1. F.J. van der Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*, Springer, 2007.

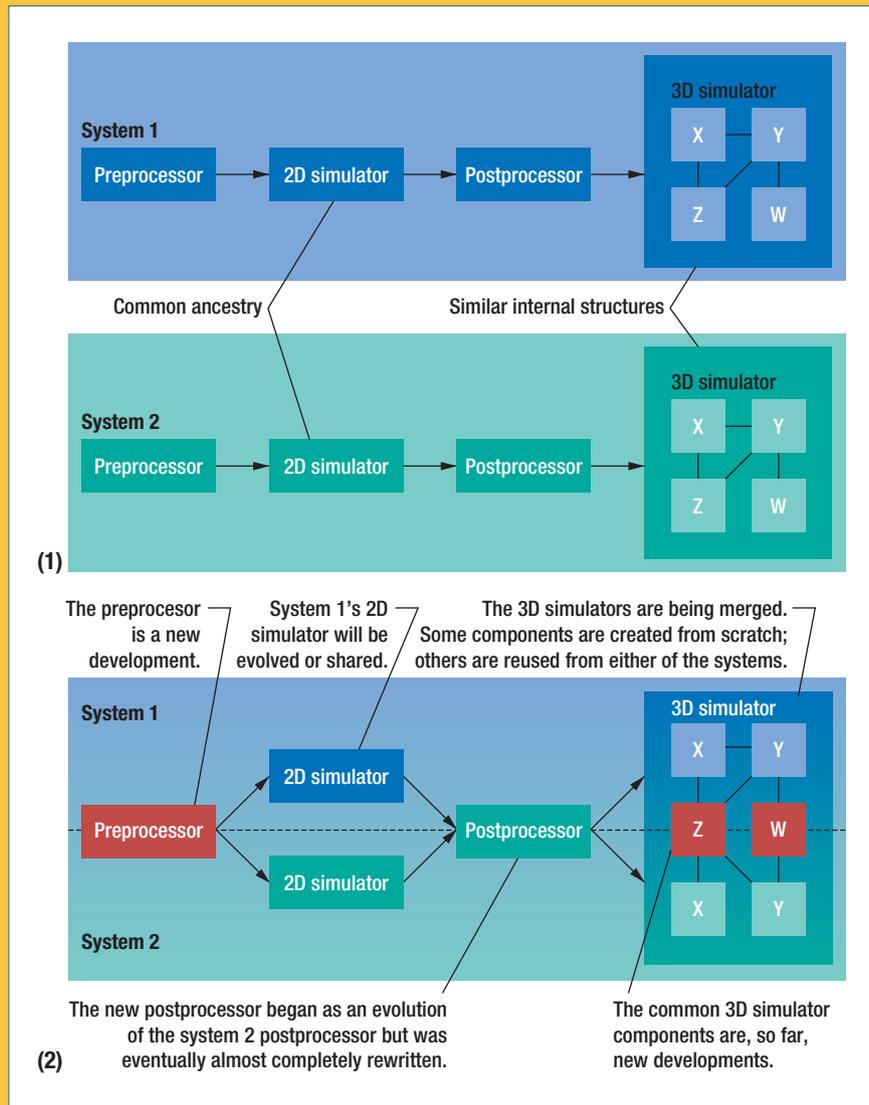


FIGURE A. The two simulation systems of case F2 in the main article: (1) Basic structure. (2) Current status. These systems show remarkably similar architectural structures, technology choices, and data models.

(for example, C and F2). In particular, we've seen that architects and developers aren't keen on planning a compromise they perceive as inferior to both existing systems. Even when each sys-

tem's developers disagree on everything else, they tend to agree on this. This lack of enthusiasm to merge systems leads to personnel and organizational challenges.

Choose One

By choosing one existing system and retiring the other, system replacement is immediate. The chosen system will likely need further evolution before it

can fully replace the retired one. So, this strategy might be the best if one system already includes most of the other's (required) features. Of course, organizations looking at this strategy should also consider such qualities as the system's reliability, ease of maintenance, and—not least—user and customer satisfaction. In general, choosing one system requires careful consideration of how to evolve it to compensate for some lost functionality—and how to present it as a natural evolution of the retired system to customers. This strategy also requires significant effort in managing issues related to backward compatibility of, for example, existing user interfaces, user processes, and file formats, as well as (for many systems) issues related to automatic migration of existing data.

Start from Scratch

Discontinuing existing systems and implementing a new system means that the resulting system can utilize the newest technology and most recent architecture and design advances. Starting from scratch might be the most natural strategy—particularly if the existing systems are considered obsolete or are difficult to maintain. However, the same backward-compatibility or migration challenges that exist for the choose-one strategy also exist here. But with the start-from-scratch strategy, these challenges exist for more systems. The crucial factor to consider is what will happen if the existing systems are retired; this insight can guide the negotiations with the existing systems' stakeholders.

Combined Strategies

Organizations often combine these strategies. Most notable among our cases is F2. In F2, the overall merge strategy meant that developers decomposed the system at one level and considered the same strategies and excluding factors for each pair of components.

Of course, there's also the option of not attempting any integration. Doing nothing requires no extra effort or resources in the short term. However, this option won't solve any long-term problems or provide any investment return.

Challenges

The personnel and organizational challenges are as important as the technical ones. Here we describe them in terms of the following two phases.

The Strategic Phase

This phase begins as soon as the two previously separate organizations understand that there's an "other" site and system.

In the studies, managers often were eager to make a decision about the systems' future as soon as possible (for example, C, D, F1, and G). They felt that the sooner they made a decision, the shorter the time of speculation, the sense of insecurity, and the associated drop in productivity and creativity would last. (These psychological factors are all important—and were particularly so in A, C, F1, and G.) However, well-founded decisions take time to mature. Our studies suggest that organizations can make a choose-one decision relatively quickly on the basis of business considerations (as in G). However, if this option isn't immediately appealing, the process will take longer—whether or not management likes it (especially in C and F1).

Personnel associated with each system must acquire sufficient knowledge about the other system to fully understand an integration-related decision's technical implications. In C and F1, management launched projects to evaluate the existing systems as objectively as possible.

This required a significant amount of time in meetings from each system's lead players: architects, expert users, managers, marketing personnel, and so forth. It also required that each system's

lead players spend a significant amount of time in meetings. These meetings were important for not only gathering information but also providing participants with a more personalized (and thus friendlier) picture of "the other side." Through these meetings, participants came to understand the other side's mentality, company culture, and informal development procedures. Several interviewees pointed to this as an important step toward future cooperation—particularly for the merge strategy. Additionally, visiting the other site and seeing that the other side consists of ordinary people with their own fears and problems⁹ makes integration seem less of a threat. It also paves the way for constructive, problem-solving discussions (for example, C, F1, and F3).

Meetings should be frequent enough that the staff won't lose strategic focus in favor of their local tasks. Additionally, by taking turns visiting each other, staff will perceive the power balance between sites to be equal (for example, C, F1, and F2). Management should be aware of the staff's motivation and personal priorities—while building new organizations, many personnel still have the old organization in their minds (and hearts). All these insights apply to any distributed team.¹⁰

After a series of such meetings, personnel will more likely consider the organization's decision realistic and support it. This phase ends when a plan for the systems' future has been devised. This plan should include a description of the target system in terms of reuse, retirement, and new implementation, together with a time and resource outline.

The Implementation Phase

Formulating and committing to a plan is an important achievement, but a long, rocky road lies ahead. Integration might take years to fully implement—whether this involves a merge or a completely new system. During this phase, it's important to keep sight of the goal

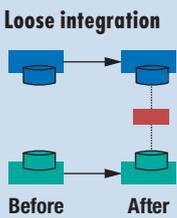
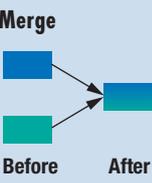
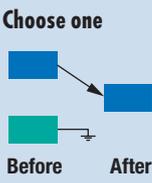
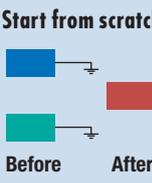
TABLE 3

The strategies and their most crucial factors.

		Strategy			
		<p>Loose integration</p>	<p>Merge</p>	<p>Choose one</p>	<p>Start from scratch</p>
Advantages and opportunities	<p>Organization: Low short-term costs and small impact (“business as usual”). An established strategy with standard solutions.</p>	<p>Organization: Long-term viable results. Involves both sites and might create a positive atmosphere.</p>	<p>Organization: Long-term viable results.</p>	<p>Organization: Long-term viable results. A positive atmosphere related to starting a new project.</p>	
	<p>Customers: No changes.</p>	<p>Customers: Existing customers experience improvements.</p>	<p>Time: Rapid replacement.</p>	<p>Architecture: The opportunity to replace old technologies.</p>	
Enabling factors and indications of suitability	<p>Architecture: Suitable if the systems are oriented around databases.</p>	<p>Architecture: Sufficient similarity of existing architectures (structures, data models, and technologies).</p>	<p>Architecture: The opportunity to replace old technologies.</p>	<p>Functionality and quality: The opportunity to improve the user experience and user processes.</p>	
	<p>Functionality and quality: Suitable if data transfer and synchronization are sufficient.</p>	<p>Functionality and quality: Suitable if each system brings unique capabilities.</p>	<p>Architecture: Suitable if one system is more modern (technologies and structures).</p>	<p>Functionality and quality: The opportunity to replace an inadequate system.</p>	
Costs	<p>Transition: No costs.</p>	<p>Transition: Might produce an indefinite period of two systems sharing too much (for example, common components) to make independent decisions, but too separate to reap any benefits. Alignment of local goals with strategic goals might require compromises with suboptimal plans.</p>	<p>Architecture: Suitable if one system covers (most of) the other’s capabilities or conceptually supports inclusion of the other’s lost functionality.</p>	<p>Architecture: Suitable if both systems are considered obsolescent or the desire exists to employ recent technology advances.</p>	
	<p>Transition: Support and evolution of two systems.</p>	<p>Transition: Support and evolution of two systems.</p>	<p>Functionality and quality: Suitable if one system covers (most of) the other’s capabilities or conceptually supports inclusion of the other’s lost functionality.</p>	<p>Functionality and quality: Might be appropriate if no existing system covers most of the desired capabilities.</p>	
	<p>Short term: Building the adapter.</p>	<p>Short term: Analysis for this strategy.</p>	<p>Transition: Support and evolution of two systems.</p>	<p>Transition: Support and evolution of two systems.</p>	
	<p>Long term: Maintaining multiple systems and the adapter.</p>		<p>Short term: Development of one system to add functionality and provide backward compatibility with the other system. Migration of data from the other system.</p>	<p>Short term: Development costs for the new system (which might not be “short term” in an absolute sense).</p>	



The strategies and their most crucial factors.

		Strategy			
		 <p>Loose integration</p> <p>Before After</p>	 <p>Merge</p> <p>Before After</p>	 <p>Choose one</p> <p>Before After</p>	 <p>Start from scratch</p> <p>Before After</p>
Important planning aspects	Not studied.	<p>Architecture and user experience: How to bridge existing differences. How to find a compromise with a high degree of reuse while achieving a conceptually integrated system.¹¹</p>	<p>Architecture and user experience: How to enable backward compatibility with the retired system.</p>	<p>Architecture and user experience: How to bring knowledge about existing design solutions into the new system. How to provide backward compatibility with both systems.</p>	
			<p>Customers: Determining what's an acceptable disruption for the retired system's existing users and customers and how to make a smooth transition for them.</p>	<p>Organization: Determining which site can best handle the new development or whether both sites should cooperate.</p> <p>Customers: Determining what's an acceptable disruption for the existing systems' users and customers and how to make a smooth transition for them.</p>	
Risks	Not studied.	<p>Organization: Requires tight, long-term, distributed development and requires alignment of merge activities with parallel development of the existing systems.</p>	<p>Customers: Loss of the retired system's customers.</p>	<p>Organization: Requires long-term commitment and possibly a distributed development effort.</p> <p>Customers: Loss of the existing systems' customers.</p>	

and not focus too much on local and short-term issues (as in F1; see Table 2). If possible, integration activities should align with local development goals and thus create a sort of inner momentum. For example, consider F2, in which integration aligned with local needs (on both sides), and the participants created a common, improved data model and other common components. Following such initial efforts, existing systems will converge, if not automatically, then at least much more easily.

The cases we studied emphasized step-by-step deliveries (particularly B, D, and F2). When aligned with local

improvements, delivering improved systems to customers and users (who don't even need to know that their system has converged with another one) can provide short-term returns on a company's investment. Internally, a functional delivery provides proof of progress and feasibility and might be necessary to maintain personnel's commitment and motivation. From our case studies, this approach seems a fundamental prerequisite for the merge strategy to succeed.

During implementation, the staff no longer needs to meet as often as in the strategic phase because implementation is an ordinary development project

with distributed teams (although admittedly still a challenge). The merge strategy, in particular, implies distributed development. This is illustrated by F2, which required close collaboration and involved frequent telephone calls as well as quarterly intercontinental travel for many project members. In A and B, anticipated problems with distributed development heavily influenced the decision to involve only one site (and were one reason to not choose the merge strategy).

The desired distribution level depends on not only the chosen strategy but also other factors. In E1, an



RIKARD LAND is a software specialist at Cross Control. His interests include software architecture and design, and processes and practices for embedded and safety-critical software-intensive systems. Land has a PhD in software engineering from Mälardalen University. Contact him at rikard.land@crosscontrol.se.



IVICA CRNKOVIĆ is a professor of industrial software engineering at Mälardalen University. His research interests include component-based software engineering, software architecture, software configuration management, and software development environments and tools. Crnković has a PhD in computer science from the University of Zagreb. Contact him at ivica.crnkovic@mdh.se.

important task was to exchange experiences between the partners in order to learn and build relations. This led to a division of work in which they collaborated more closely than when motivated from a purely implementation-efficiency viewpoint. In F3, the new system was a support system to be deployed and used at all sites; this required

considerable communication between implementers and users at each site.

Table 3 summarizes important integration considerations based on these case studies; it can serve as an actionable checklist. The advantages it lists should help your company get started and decide where to focus its investigation. The remaining considerations listed should help you collect the information you need to develop a plan. To collect this information, it's best to involve appropriate personnel (architects, users, management, and so forth) and assign different groups to investigate the relevant issues listed in Table 3 from their

particular viewpoints. This will help form a well-founded decision and build the commitment needed for success. 

Acknowledgments

We thank all interviewees and their organizations for sharing their experiences and letting us publish them. Thanks also to Stig Larsson and Laurens Blankers for previous cooperation that led to this article. The Swedish Foundation for Strategic Research partially supported this research through the Progress strategic research center.

References

1. S.R. Wall, *The Morning After: Making Corporate Mergers Work after the Deal Is Sealed*, Basic Books, 2002.
2. R. Land and I. Crnković, "Software Systems In-House Integration: Architecture, Process Practices, and Strategy Selection," *Information and Software Technology*, vol. 49, no. 5, 2007, pp. 419–444.
3. B. Gold-Bernstein and W. Ruh, *Enterprise Integration: The Essential Guide to Integration Solutions*, Addison-Wesley Professional, 2004.
4. J. Lee, K. Siau, and S. Hong, "Enterprise Integration with ERP and EAI," *Comm. ACM*, vol. 46, no. 2, 2003, pp. 54–60.
5. F. Biscotti et al., *Forecast: Enterprise Software Markets, Worldwide, 2008–2013, 3Q09 Update*, Gartner, 2009; www.gartner.com/DisplayDocument?id=1179913.
6. M. Stonebraker and J.M. Hellerstein, "Content Integration for E-Business," *ACM SIGMOD Record*, vol. 30, no. 2, 2001, pp. 552–560.
7. D. Garlan, R. Allen, and J. Ockerbloom, "Architectural Mismatch: Why Reuse Is So Hard," *IEEE Software*, vol. 12, no. 6, 1995, pp. 17–26.
8. L.A. Davis et al., "Patterns of Conflict among Software Components," *J. Systems and Software*, vol. 79, no. 4, 2006, pp. 537–551.
9. T. DeMarco and T. Lister, *Peopleware: Productive Projects and Teams*, 2nd ed., Dorset House, 1999.
10. E. Carmel, *Global Software Teams: Collaborating across Borders and Time Zones*, Prentice Hall, 1999.
11. F.P. Brooks, *The Mythical Man-Month: Essays on Software Engineering*, 2nd ed., Addison-Wesley Professional, 1995.



stay connected.
IEEE computer society

twitter | @ComputerSociety
 | @ComputingNow

facebook | facebook.com/IEEE ComputerSociety
 | facebook.com/ComputingNow

LinkedIn | IEEE Computer Society
 | Computing Now

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.