Very regular reader of software engineering literature has surely seen articles proclaiming benefits of software reuse. While sometimes portrayed as one of the elusive software silver bullets, reuse has also received a fair amount of negative publicity from those who just do not think it will ever work. I have seen the benefits of successful software reuse programs and want to take a moment to review the lessons learned from those who have been there.

Let’s look at the many achievements and positive effects that reuse has on many practical problems faced by industry. Early work on library methods gave the field a focus and allowed it to grow. More recently, research on reuse practices, organizations, domain analysis techniques, and metrics have found a place in the daily routine of large software development organizations [8]. Techniques such as object-orientation and software architectures often rely on software reuse as a unifying goal or objective. Finally, technologies such as visual programming (for example, Visual Basic, or VB, provides a natural medium, such as drag-and-drop, to unconsciously practice reuse.) As Ed Yourdon observes, “the real benefit of languages such as VB/Powerbuilder may be software reuse rather than the visual paradigm” [10]. Overall, I believe the field of reuse has made considerable progress.

Reuse Library Problem—Solved
For much of the past 10 years, reuse has focused on building libraries. We have explored and conquered numerous library issues, such as classification [9] and access mechanisms [7]. We have discovered we should concentrate on small libraries of greatly used, well-designed, domain-specific, high-quality components, and we should avoid large component collections of questionable heritage. The best libraries range from only 30 components to, in rare cases, as many as 250 components. In general, users find formal classification methods difficult to use [6]. Collecting large amounts of metadata to help retrieve components wastes time, money, and makes the library both difficult to contribute to and difficult to retrieve from.

We also learned we must provide quality components; the first time someone retrieves a component with a bug from your library will be the last time they use the library. Knowing this, IBM developed a 80,000-LOC collection of highly-used software that never experienced a single error [2].

Domain Analysis Problem—Solved
The key to high levels of reuse comes from building collections of components that all work together and that many applications will need. We call the problem area that contains these related applications a “domain,” and we call the process of identifying the common components and specifying how they fit together “domain analysis.” Many proven domain analysis methods exist [1]. These methods tend to share a common process that involves the following:

• Domain characterization and project planning which consists of the feasibility analysis and planning steps common to any engineering project;
• Data collection which consists of acquiring raw data and information from experts, existing applications, documentation, and so forth;
• Data analysis which consists of filtering, clarifying, abstracting, and organizing the raw data;
• Classification which consists of generalizing the findings for maximum coverage and in specializing the findings to iden-
tify the specific parts of the problem that change with the various possible uses; and

• Evaluation of the domain model, the critical step of validating the results of the domain analysis process.

Note that without domain analysis, a project can hope for little more than about 20% reuse. Because up to 65% of an application can come from domain-specific software, high levels of reuse—as well as the corresponding benefits—depend on the planning, discipline, and investment that comes though this kind of formal reuse. Organizations that desire to achieve high levels of reuse can choose from several existing, effective, and proven methods.

Metrics Problem—Solved
Advocates of any discipline eventually come to realize their ability to sell their idea to management depends on the idea’s return on investment. Likewise, reuse practitioners have learned the mantra: “business decisions drive reuse!” Since the early days of reuse, we have seen articles on metrics and economic models, many of them designed for use with specific programming languages or for specific purposes. For example, we have seen models for predicting the cost of development with reuse and models for finding the critical number of reusers necessary to break even on a reuse investment [5]. This metrics work has succeeded in that we have defined what to count as reuse and how to count it, we have defined metrics for reuse and can recommend these metrics for immediate use on projects and within organizations, and we know how to assess the financial benefits of reuse both in the short and long term [8].

Organization for Reuse Problem—Solved
The management structure of an organization often determines the failure or success of a reuse program. Face it: different reporting hierarchies and geographic distance (even if just down the hall) preclude strong, collaborative relationships. Many recently published reports correlate reuse results to the various ways of organizing for reuse to include lone producers, nested producers, pooled producers, and team producers [3]. Hewlett-Packard Labs found the team producers concept worked best; this model corresponds to the successful “Parts Center” approach advocated by IBM [2].

Based on these experiences, organizations should entrust the development and maintenance of shared software to a Reuse Development Team. As shown in the figure, the Reuse Development Team fits into the management structure on the same level as any software development organization. However, the project manager always puts the reuse team into place well before any of the other organizations so that it can create a foundation of shared software before any other development begins.

The budget for this reuse organization can come from contract investment funds, overhead funds, or a tax placed on each of the development managers. On one project, we funded the Reuse Development team with a 10% tax from each of our seven development managers. In return, the development managers averaged 23% unmodified reuse from components supplied by the reuse team; a benefit of over twice their forced investment! As mentioned earlier, justification for such investments depends on the efficacious use of reuse metrics.

Reuse allows us to efficiently build on previous work. It allows
us to reduce duplication of effort. Just as proper research allows us to stand on the shoulders of others, software reuse allows us to extend ourselves with products provided for us. Rather than using valuable resources to rewrite abstract data types and similar functions, we can plan ahead and design a domain-specific suite of software that will serve as the foundation for an entire product line of software applications.

We know we have solved the issues mentioned here because we see reuse success stories all the time. Surely, many of the issues related to reuse, such as obtaining management commitment and funding, take a lot of effort to overcome. Considering the problems we have solved, it amazes me we keep seeing papers, for example, on reuse libraries. While reviewing papers over the years for journals such as IEEE Software and conferences such as the International Conference on Software Reuse (www.cs.vt.edu/icsr5), I have noticed that recent submissions address many of the same topics that have appeared many times in the past. Furthermore, they do not recognize previous work. These papers often present casual opinions, unvalidated processes, or lack any scientific or engineering basis [4]. Those who proselytize reuse should apply the same logic to their research and not start from scratch. Reuse R&D needs to focus on the unsolved problems, such as interoperability, framework design, and component composition techniques.

Planning for reuse and investing in reusable software gives us the power to deliver more function, to deliver it faster, to deliver it with fewer defects, and to deliver it with less cost. Our experiences have solved many critical obstacles to reuse and led to many successful projects. By building on this work, I believe the field of reuse will continue its tradition of progress and significantly contribute to our ability to meet our society’s voracious appetite for software.

Jeffrey S. Poulin (jeffrey.Poulin@lmco.com) is a senior software engineer at Lockheed Martin Federal Systems, Owego, NY.

References
3. Fafchamps, D. Organizational factors and re-use. IEEE Softw. 11, 5 (Sept. 1994); 31–41.
10. Yourdon, E. Lipstick on a pig or a real silver bullet?" Amer. Prog. 9, 11 (Nov. 1996), 25–29.