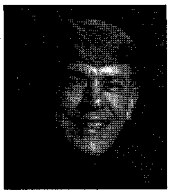


Robert L. Glass

The Loyal Opposition

Reuse: What's Wrong with This Picture?

Something is seriously wrong with reuse. If there is a motherpie-and-applehood topic in software engineering, reuse is it. Everyone believes in it; everyone thinks we should be doing more of it. So do I. Reuse does have the potential our industry attributes to it. But the question that keeps sticking in my mind is this: "Why hasn't that potential already been achieved?"

Most software engineering literature points the finger at management. Don Reifer, for example, has conducted lots of industrial-strength studies of businesses that practice reuse (see, for example, *Practical Software Reuse*, John Wiley & Sons, 1997), and to him the conclusion is clear: the companies that succeed at reuse do so because of managerial commitment, while the ones that fail lack such commitment. Several others say much the same thing. Ruben Prieto-Diaz turns the problem around and asserts that companies that fail at reuse do so because they treat it as a technical problem.

I disagree. It seems to me that reuse's fundamental problem is clearly not a lack of managerial commitment. Based on my own experiences, I think reuse hasn't succeeded to the extent we would like because there aren't that many software components that *can* be reused.

Useful vs. Reuseful

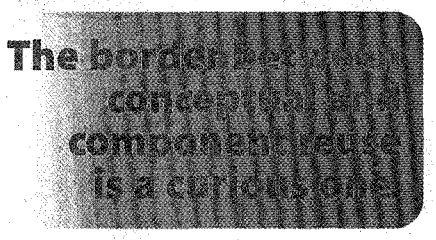
Reuse has been a career-long interest of mine. From my earliest days in the software field in the 1950s, I have sought to construct and institutionalize reusable components at the enterprises for which I worked. Back in the 1950s, reuse was a thriving idea—user organizations like Share constructed and maintained what I then called the Software Parts Catalogs—lists of basically reusable software components. The applications of the time were small by today's standards and so were those reusable components. Nevertheless, reuse thrived—you didn't think of writing a new application without first looking in the Share catalog to see how much of your product-to-be could be brought in off the shelf.

EDITOR: Robert L. Glass • Computing Trends • rglass@indiana.edu

0740-7459/98/$10.00 © 1998 IEEE

Time passed and the extent of component reuse failed to grow. Why?

Several things happened. First and most importantly, the field of reusable components had been pretty well plowed out. My reuse-focused comrades and I struggled to identify program pieces that could be built and reused to the extent those early components were. Each software program, at a detail coding level, was different—which is why we kept building all that new software. Once we stepped beyond the fairly small, implementation-focused components (the ones you now learn about in computer science courses) and into the world of larger, application-focused components (the ones computer science courses don't say much about unless the application domain interests the instructor), that which was worth generalizing became increasingly elusive. I built what I thought were reusable components, only to find that the needs of any specific and real application differed enough from what I'd done to make my product useless—or at least *re*useless.



Speed and Quantity vs. Quality and Reusability

Developments in the business of making software placed further obstacles in the way of effective reuse. As it matured, our field moved from an era when getting a good product out the door mattered most to an era when meeting a sometimes arbitrary schedule dominated. Few people had time to think about reusable components because, as most experts agree, it takes longer to build something reusable than something merely usable.

Another problem surfaced. Most software measurements focused on quantity. We began to measure new products in lines of code, thinking that the more lines written, the more significant the project must be. Reuse, which takes longer and results in a product with fewer new lines of code, bucked that trend. The reward system failed to reward reuse.

These last two problems involve management. So, in this context at least, I agree with the experts who say that management holds the secret to increased reuse. Management can ward off schedule pressures and gain time for building reusable components. Management can find new reward systems so that those who build and use reusable components will be doing something acknowledged as measurably positive.

But I fear that won't be enough. If I'm right about the dearth of potential reusable components at a meaningful level, then all those efforts will fall short of our unrealistic expectations.

Conceptual vs. Component Reuse

To comprehend the full picture of software reuse we must also consider conceptual reuse. Our field's architecture, framework, and patterns movements are built on the notion that more can be reused in our field than components. Although they are right, this is not exactly news: good practitioners have been using conceptual techniques for decades. Software professional Butler Lampson said, over 10 years ago, "Most of the time, a new program is a refinement, extension, generalization, or improvement of an existing program. It's really unusual to do something that's completely new" (quoted in *Programmers at Work*, Susan Lammers, Microsoft Press, 1986). Software researcher Willemien Visser ("Strategies in Programming Programmable Controllers: A Field Study on a Professional Programmer," *Empirical Studies of Programmers: Second Workshop*, 1987) said much the same thing at much the same time: "Designers rarely start from scratch." Good software professionals keep their mental backpacks filled with previously used concepts that they can apply when a familiar-sounding "new" problem comes along. It's not surprising, then, that much of the material in the early books on architecture and patterns was based on practitioner work, not theoretical discovery. Thus concept reuse thrives and will continue to.

The border between conceptual and component reuse is a curious one. Studies, especially those conducted by organizations like the NASA Goddard Software Engineering Laboratory, show that the borderline itself is surprisingly rigid. Most reuse scholars agree that if more than 20 percent of a component must be reworked for its new use, it is more efficient to start from scratch. Some even say 10 percent. Bending software is *that* difficult; modification can be harder than building anew.

Which brings me back to my main concern. I believe that few software components—whether pieces of code or software parts—can be reused at the 80-100 percent level in the average software product. Most of the components that can be

reused in this way were built 30 to 40 years ago. I hope I'm wrong, because I'm all in favor of finding and making available more such components. But few enterprises succeed at large-scale reuse, which suggests that the problem is more than simply one of will. That, in turn, means the problems with reuse run much deeper than management.

Optimism vs. Reality

Nevertheless, there are isolated pockets of component reuse success. The most important, and certainly the most verifiably unbiased example, comes from the Software Engineering Laboratory at NASA-Goddard, which routinely achieves a 75 percent reuse level. To be fair, we must note that the NASA work focuses on a very narrow and cohesive application domain: flight dynamics software. The Software Engineering Lab achieved their remarkable success in component reuse by adopting a thorough domain analysis approach. Domain analysis is a necessary but—by itself—not a sufficient approach to reuse. If large-scale reuse is possible in a domain, domain analysis will find it. But it's not always possible: there simply aren't that many objects and tasks that are 80 percent the same in most domains. Although the Lab achieved results of 75 percent using an object-oriented approach facilitated through Ada, they encountered several problems with that language. Forced to regress somewhat to using Fortran, the Lab still achieves 70 percent reuse with a non-object-oriented approach.

I know that people who use OO approaches tend to build such huge class libraries that the largest reuse problem they encounter is in searching the catalog to find what they need. Indeed, a thriving computer science research community is seeking better ways to organize and access reusable component repositories. I suspect this is a case of optimism overriding reality, however. I would like to know the reuse count on those classes stashed away in OO repositories: the number of times each class has been reused in a real production program. I suspect that the repository population is considerably larger than the reuse counts.

Breakthrough vs. Breakdown

Why have I spent so much time apparently naysaying reuse? Because all too often our field pins its hopes on some breakthrough technology that, in the end, turns out to be BS. Reuse currently enjoys that sort of status. One new book on reuse contains a section labeled "Software Waste," implying that writing software from scratch is a massive waste of energy. If my fears prove correct, our enthusiasm for reuse, eroded by such exaggerated claims, will dissolve. Our field deserves better than that. Most new technologies do offer some modest benefits. Reuse is one such technology. I want our field to reap its modest benefits without the disappointments that result when breakthroughs become breakdowns. ❖

Few enterprises succeed at large-scale reuse, which suggests that the problem is more than simply one of will.