Reusability Comes of Age

Will Tracz
IBM Federal Systems Division/
Stanford University

Of the seven articles selected for this special issue of IEEE Software, five are accounts of projects that have made the reuse of software components a reality. This focus is deliberate: By showing how it's been done in the real world, we hope to provide some insight and tools to implement reuse in the workplace. The final two articles examine what kind of help programmers need to evaluate and understand software intended for reuse.

For years, I have used an analogy that compares used cars to used programs (Computer, April 1983, June 1986, and May 1987). I think this analogy holds for pungent and pragmatic reasons: People are leery about buying a used car for many of the same reasons programmers are reluctant to reuse someone else's work. With this analogy in mind, I offer some of my own insights — as a seasoned veteran of many used-program sales — into the problems of software reuse and what factors have inhibited its acceptance as a viable form of software development.

New or used? Before deciding on whether to invest in a new or used car, a prospective buyer should first identify his needs (features, performance, price range). Other factors, like urgency, may constrain the selection process to a vehicle on the lot instead of one ordered from the factory. In any case, the buyer should develop a strategy for evaluating candidates.

The next step is to shop around. Will a potential candidate that looks good on paper really live up to everything a smooth-talking salesman says?

Standard features. The first question that needs to be answered is: Does the vehicle meet the customer's requirements? Some models have options (a convertible roof, four-wheel drive) that enhance their adaptability to future operating environments. But these features may just be extra baggage that interfere with overall performance or add to maintenance costs.
Clearly, if a software program meets the customer's basic requirements, then it warrants further consideration. But program options are mixed blessings: On one hand, parameters aid adaptation of the software to future needs; on the other hand, they can result in normal operating inefficiencies and increased maintenance costs.

**Mileage.** A new or low-mileage used car, in all probability, will require less maintenance than a high-mileage used car. Yet finding a low-mileage car on a used-car lot is cause for suspicion, since most people would not part with it unless it was giving them problems.

But software is unfatiguing, so the higher the mileage (the more users, the more systems it is available on, or the longer it has been in use) the more desirable it is. History has proven that the number of bugs found in software decreases with use, assuming other factors remain the same.

**Maintenance record.** Knowing the types of repairs made on a vehicle and the quality of the maintenance effort will influence a customer's decision. If serious problems occurred early in the vehicle's life but were properly repaired, they will not weigh as heavily in the selection process.

A customer can readily evaluate the quality of a program by looking at the type, severity, and date of problems found in a piece of software. If many miles have been put on the program since the last change or if the types of updates have been insignificant, the prospective buyer can place more confidence in the product. The customer should avoid a situation where the maintenance record indicates more problems are introduced each time one problem is fixed.

**Appearance.** Kicking the tires and checking the paint job and trim are other ways the customer can gain useful information. Sometimes it pays to have a skilled mechanic perform a close inspection under the hood to determine potential problem areas (temporary repairs or shoddy workmanship).

A software buyer can also tell shoddy workmanship by examining the exterior (the user interface and documentation) of a program. Looking under the hood helps the customer assess certain programming characteristics (naming and commenting conventions) that can indicate the overall maintainability, modifiability, and reliability of the component or product.

**Standards.** Standard instrumentation and compliance to safety and operating standards (seat belts and emissions control) give customers a feeling of confidence in the product. Standard compliance dispels some of the fears of being stuck with a lemon.

If a piece of software complies with certain standards (in documentation, interface design, and testing), its potential for reuse is increased because of the perceived quality and useability of the software.

**Warranty.** What happens if something goes wrong? Will the dealer fix the problem or will the customer be left holding the bag?

To establish the credibility and viability of new and used programs and components, the seller should provide both a policy for determining responsibility for error and a mechanism for resolving problems.

If a program does not meet all the customer's requirements, the manufacturer, dealer, or customer is faced with the task of modification. What options are available, and how easily is a program customized? These characteristics play an important role in determining the overall reusability of a program. Finally, the buyer should ask how customization affects the warranty. If things go wrong, the new owner might have difficulty soliciting assistance from the original manufacturer if the program has since been modified.
Options. Because customer requirements and tastes vary, a manufacturer provides options to satisfy the customer's needs as closely as possible. Certain options must be installed by the manufacturer; others can be installed by factory-trained mechanics; still others can be installed by the customer. In any case, the risk associated with adding an option decreases as the expertise of the person making the changes increases.

Accessibility. Customers don't want to waste their time driving all around town to find the car that meets their needs. Dealers who advertise a large selection have a better chance of attracting business.

Similarly, the convenience of shopping or getting a program serviced locally is very appealing. The less effort customers have to expend in finding candidates for reuse the more likely they are to buy them.

Price. When buying a new or used car off the lot customers have to pay for whatever options come with the car, whether or not they were on the customer's original list of options.

Programs require investments in both capital and time. Off-the-shelf software is often sold as a package deal. In this case, the customer may be paying for more functions than required and will end up dragging around extra options that they neither need nor have space for. Similarly, the customer has to pay the price for learning how to drive the new software effectively and how to maintain it. These hidden prices, plus the price of failure, must be factored into the buy-or-build decision.

Test drive. A car's environment is the test drive. The buyer can experience the true feel for how a car handles under different driving conditions and can project realistically what it would be like to own.

A customer who tries a program on for size can determine if any rough spots exist in the user interface and how the program performs under simulated working conditions (if possible). If the problems are minor, the seller might be able to customize the software before consummating the sale.

Intangible inhibitors. The reputation of used-car salesmen and the products they promote is somewhat negative. Getting a lemon is a major concern of most used-car customers. This same lack of trust in programming products has been the major inhibitor in advancing software reuse. Unfortunately, because it is often easier to write an incorrect program than to understand a correct program, programmer productivity (which would increase if software developers didn't reinvent the wheel each time) and program quality (which would improve if the reused had high-quality parts) have not evolved.

Two other reasons may explain why the used-program market originally envisioned by M.D. McIlroy in 1968 has failed to materialize:

1. There are no clearly defined standards, either for developing reusable software or for systems based on reusable software.
2. There are neither large repositories of reusable software and components nor the tools to access and synthesize systems from them.

What will it take to create a successful used-program business?

- Quality parts: Customers should have confidence that what they buy will perform without error.
- Standard interfaces: Customers should be able to use what they buy in a manner that complies with standard operating conventions. Software should be easily integrated into new or existing systems.
- Documentation: Customers should understand what the software they buy does, how they can use it, and how they can modify it if necessary.
- Selection: Customers should have a choice of options available on what they buy.

Nothing is better than cruising along in a high-performance, well-tuned program with complete confidence in the safety of all those who depend on you to get the job done. Unfortunately, the state of the practice today has us lumbering along in a clunker that spends most of its time in the shop undergoing repairs and has the distinct possibility of crashing due to some unforeseen manufacturing defect.

To some the choice is obvious, and to those I say, "Read on!" Your skepticism should be abated by the progress of this issue's authors. The technical foundations from making software reuse a viable alternative to program development have been identified and demonstrated, thus adding credibility to the used-program business. Reuse is not a reality for us all, but the question of whether or not it is — and always will be — the technology of the future has been settled.

Will Tracz is an advisory programmer for IBM Federal Systems Division, Owego, New York, where, until 1984, he was responsible for the design and development of microcode support software for military and aerospace processors. During his 12 years at IBM he has also taught classes as a visiting professor at the Rochester Institute of Technology, and was an adjunct professor at Syracuse University.

Tracz is the ACM SIGMicro/IEEE TC-Micro newsletter editor, and past chairman of the 15th Workshop on Microprogramming (Micro-15). He has written over a dozen papers and technical reports on microprogramming, programming languages, and software reusability, including a satirical collection: Confessions of a Used Program Salesman.

Tracz received an MS in computer science from Pennsylvania State University, an MS in computer engineering from Syracuse University, and is currently completing work for a PhD at Stanford University.