Early reuse practice lives up to its promise

Galen Gruman, Soft News Editor

For many years a topic of research, software reuse has begun to enter commercial software production. Several large organizations are now applying reuse:

- AT&T Bell Laboratories in New Jersey is using reusable components as part of a three-year program to triple productivity. In two transaction-processing projects, one with 120,000 and the other with 210,000 lines of code, reuse rates were 65 percent. On a 1.2-million-line project, 86 percent was reused.
- GTE Data Services in Tampa, Fla., began an asset-reuse group late last year that works with development groups to seek out reusable components and to get developers to reuse existing components.
- The Microelectronics and Computer Technology Corp., an Austin, Texas-based group of 21 firms, plans to field-test early next year a combined design-recovery and program-generation system that reuses old designs to build components for new projects.
- The Software Productivity Consortium, a Reston, Va.-based group of 14 defense contractors, announced this summer a new development methodology based on reuse. That methodology is part of a development environment the SPC plans to complete by 1991.
- A Defense Science Board committee last fall recommended the establishment of a national clearinghouse for reusable components, a mechanism like what the music industry uses to collect royalties from radio play and record sales and distribute them to the artists. The clearinghouse would not validate components.
- The Defense Dept. this year strengthened its requirements that defense contractors use Ada and its reusable packages (see related story on p. 89).

Why reuse? The promise of reuse is to stop the continual reinvention of code for common functions and to improve the quality of code through the use of thoroughly debugged modules. The incentives are largely economic: development and maintenance — especially for large, complex projects. Another motivation are the fears that there will be too few programmers to handle the increasing workload or that firms will contract coding out to lower-paid programmers abroad — but there is disagreement on how valid these fears are.

"Software is becoming *the* important part of the systems that form the infrastructures of our business," said Eric Sumner, AT&T Bell Labs' vice president for operations planning. But the software industry's productivity has increased at roughly 5 percent a year, compared to 65 percent for the semiconductor industry and 100 percent for the communications industry, he said. "At AT&T, we set a goal two years ago of boosting the slow, almost linear growth and seek a tripling of productivity in three years. We are well on our way," he said.

The state of the practice is still less than the potential, less than needed, and less than acceptable.

"If there's a single key, it's reuse," Sumner said. "In useful, economic software — stuff you can sell — we expect an overall productivity in coding of two or three lines [for each coded today] for these systems, despite the fact we didn't have a large or carefully honed library," he said.

Estimates of productivity improvements vary greatly, depending on the component size, language used, and problem domain. "You can push productivity as high as you want by narrowing the domain," said Ted Biggerstaff, director of design information at MCC. "I think it's going to be done domain by domain, with the first successes and the greatest successes in the narrow domains," said Fred Brooks, a computer-science professor at the University of North Carolina at Chapel Hill.

Improvements of 20 percent to 50 percent are reasonable, Biggerstaff said, based on research he has seen while editing a forthcoming book on reuse. "Reusability already saves a lot of coding, such as in complicated function libraries," Brooks said. For example, "in Unix, the power of the whole pipes-and-filters-and-unified-file-structure regimen is that it lets people lash together pieces they have lying around. These kinds of reusability are immediate," he said.

Quality also increases with reuse, Biggerstaff said. "You find that quality comes along for free because you use the components that have been debugged a lot and for a long time during development," he said. Of course, "it can't forestall incompetence," Biggerstaff said. Brooks agreed: "By itself, it is not going to make our problems go away," he said.

Reuse challenges. As a research topic, reusability is an area that people have been working with for 10 years. "It has been a hot area in the last five or six years. There's a lot of promise for reusing things," said Peter Freeman, director of the National Science Foundation's Computer and Computation Research Division. "There are certainly some gains to be made from what we already know how to do. The problem is largely one of education and management," Freeman said, although he said that there are still technical areas to be developed.

The technical problems include creating components large enough to justify their investment but broad enough to be useful and creating a library system for components. Not everyone believes these problems can be solved. "These problems are exactly the ones that pre-

Many people not quoted in this report shared their insights. Three deserve special thanks: Susan Gerhart of MCC, Akihiko Yamada of NEC, and Don Reifer of Reifer Consultants.

vent the idea from realization," said Noboru Akima, planning director of Japan's Sigma, a software-development-environment project directed by Japan's Ministry of International Trade and Industry and jointly undertaken by several Japanese computer firms.

Akima was involved in a software-component project several years ago. "It was difficult to define the component size, interface, and functionality. Software is *soft*; that makes the problem difficult," he said. (In 1987, the January, May, and July issues of *IEEE Software* included several articles examining these problems.)

Two years ago, when he was a computer-science professor at the University of California at Irvine, Freeman wrote that the state of the practice for reuse was "embarrassing." Today, while better, "it is still less than the potential, less than needed, and less than acceptable," he said.

How to reuse. There are many approaches to reuse. Brooks breaks them down into three levels: algorithm, code, and design.

Algorithm reuse. Brooks said he believes much productivity could be gained by reuse at the algorithm level. "I would do it by creating [on line] what [Donald] Knuth has documented. The common literature is Knuth. There has been nothing of that stature since then," he said. Knuth's three-volume series *The Art of Computer Programming* (Addison-Wesley) contains those algorithms.

Brooks cited the success of the nuclear-physics coding community, which "has had a lot of success in trading code" because it works on a narrow application and uses the same symbols. "If we [all] used the same names, that would be a hope" for productivity, he said. However, "it's not a trivial task. That's a substantial infrastructure problem," he said.

IBM is using standard algorithms to build up a library of building blocks, said Richard Butler, director of programming-systems development.

Component reuse. "At least for certain types of systems — especially transaction systems — we think we could establish a number of functional modules, as few as 25 to 40," AT&T's Sumner said. "These would be the building blocks," he said, much as in the hardware industry. "No one designs a memory anymore, they buy a memory chip," he said.

To make such a component approach work will require standard architectures and interfaces, Sumner said. "We need to

Continued on p. 90

Reuse experience

Continued from p. 88

borrow from VLSI and good hardware engineering," he said. "The modules have to be proprietary — they're like capital," Sumner said. He admitted that the question of open standards versus proprietary tools is a problem, but "that's a dilemma everywhere."

The SPC approach is to build reusable components with traditional methods but to specify the large systems they go into with its new reuse-based development methodology, Yudkin said. "We have now defined the set of tools to carry out this methodology," he said. The tool prototypes are now in production and are due out early next year, he said.

SPC has encountered the same problem that many face when creating reusable components: making reusable but flexible components. "The hard part is to get reusable things that have two opposite characteristics: offsetting lots of work but flexible enough to amortize the costs of using it," Yudkin said.

But, Yudkin said, it can be done. He cited the Common Ada Missile Parts, a generalized cruise-missile guidance system using Ada packages. "That subsystem has been widely reused," he said.

Design reuse. "Code reuse in a different area than the application is very difficult," said Yoshihiro Matsumoto, a fellow scientist at the Toshiba Heavy Apparatus Engineering Laboratory. One solution, he said, would be CASE tools that can store in a rule base some of the knowledge used to build the software. Japan's "Sigma will provide the sufficient support tools to begin with and incorporate advanced technologies whenever possible," Akima said. "I expect these technologies will be provided by academia and industry labs," he said.

MCC's Software Technology Program is one group focusing its reuse efforts on design, not code. The prototype design-recovery system, named Desire, works with a domain model and a human software engineer to recover existing code's design, Biggerstaff said. The current version lets you view code structure and the relationships between files, he said. MCC is now building a domain model and recognition model for Desire.

Another tool, named Rose, generates programs from user requests, which are entered as fuzzy requests like "I want to make an accounts-receivable system" or "I want to have these inputs and outputs," Biggerstaff said. After Rose suggests a specification template through dataflow diagrams, the user edits the template and adds constraint information to refine the design. Rose has design rules that fire when enough relevant information is entered, and may even suggest alternatives from a reuse library, he said.

When the template is complete, the user sends it to a back end with a second reuse library that contains algorithms written in a neutral language that uses very abstract data types. Rose decides how to implement the code using the template and the abstract algorithms, Biggerstaff said. The output code can be in C, Pascal, or Ada.

The current version works on small systems, and has created a 600-line program in 15 minutes on its first try, Biggerstaff said. That 600 lines is equivalent to 200 tightly hand-code lines, he said, but "we think we can get down to within 20 percent." MCC is enhancing Rose to handle very large-scale components — 10,000 to 100,000 lines each — and Rose will be able to handle almost 10,000 lines this year, Biggerstaff said.

Design reuse has several advantages over code reuse, Biggerstaff said. "It's fairly costly to populate a reuse library," he said, because "the simple stuff has been done with subroutine libraries." Design reuse also "gets you away from the combinatorial explosion [you get] when you try to reuse code. With templates, it only grows additively," he said.

Design reuse gets you away from the combinatorial explosion you get when you try to reuse code.

SPC's methodology also has a design-reuse component. One SPC tool is a synthesizer that captures reusable parts of a system's architecture. Yet to be built are generalized rules that the synthesizer can use to apply these reusable parts to architectures in other domains, Yudkin said.

Design reuse has been the norm in Australia for 20 years, mainly because of the large commercial-package industry (50 percent of developments) there, said Karl Reed, a computer-science professor at the Royal Melbourne Institute of Technology. "The way most packages come into being is the reuse of design and modules from a client's design and environments," he said.

Large or small? The effect of component size is an area of debate. Some argue that small components are easier to write and use because they are not very complex.

IBM's Butler said he believes small components — those with one function — are better than multifunction components because programmers have a better understanding of the what the component does. "Even if it costs as much to use a [small] reusable part than to implement it yourself, you still have the benefit of a zero-defect part," he said.

A good example of where small, varied components work is Toshiba's Heavy Apparatus Engineering Laboratory in Fuchu City, Tokyo, Biggerstaff said. The industrial-process control systems there have broad requirements, good customization, and a constrained abstract design, he said. The reuse rate is about 55 percent of code delivered to outside customers, said Toshiba's Matsumoto. Component size ranges from a few hundred to a few thousand lines of code.

But piecing together small components "becomes a rather dull job," AT&T's Sumner said. "You see the pictures of rows and rows of people at workstations doing a small part of the job. This makes the work very repetitive," he said. At AT&T, "we think we'll get a few larger modules and customizing scripts."

There are trade-offs between small and large components, MCC's Biggerstaff said. "There are bad things that happen [when] you make them fairly large — which you have to do to get any payoff," he said. As is true in VLSI-circuit design, "unless you do a lot of function *X*'s, you won't get anyone to develop function *X*," he said. If you find such a function, "you need to abstract it across a wide set of programs," he said, but still keep it useful.

Brooks said he suspects profitability and reusability will be greater on large components. He cited Unix tools such as Lex, YACC, queue managers, quicksorts, and file-format transformers. How much of that is what programmers use every day? "Right much," Brooks replied.

"We have had more success with the large, subsystem-type components than with many small building-block components, said Diane Collier, supervisor of the software-asset organization at GTE Data Services. With a large system, "we can take a chunk of what they [a division developing a new system] do and say, 'Here it is.' It's more self-contained," she said. With smaller components, it is "harder to hone in on [whether] it will be reused" and to make it generic enough, she said.

Languages. Although many new languages —especially object-oriented ones — boast reusability features, many organizations have found that they can reuse

code written in their current languages.

Toshiba's Heavy Apparatus Engineering Lab uses real-time Fortran for about 60 percent of its code, Matsumoto said. It does another 20 percent in an intermediate system-description language created at the lab, he said, although it is replacing that with C.

GTE Data Services uses Cobol and assembly as its two main languages because "our environment is transaction processing and business," Collier said. "We have philosophical debates about object-oriented languages and Ada," she said. While other GTE divisions use Ada because they do work for the Defense Dept., the Data Services Division has no such economic pressure, she said.

Object-oriented languages may also be too new for widespread use. In creating its methodology, the SPC encountered several surprises, Yudkin said. "A lot of the vaunted object-oriented technology doesn't really exist," he said, such as database-management systems and tools. "It's a good idea, but the support technology isn't there," Yudkin said. And the same holds true for Ada, he said.

But "object-oriented languages are probably the best, cleanest kind of reusability, especially object orientation with inheritance," MCC's Biggerstaff said, because they "give you a nice ability to set up architectural details and defer binding. I'm not as keen on Ada and other abstract data types."

Management role. To develop software productively, "you need very careful project management," AT&T's Sumner said. "Many projects are rarely started right, so they have serious flaws in the requirements that become very expensive to fix later," he said — an even more severe problem for reusable components.

Getting reusable components requires more work at the beginning of the development effort. "Break the system into pieces and see what is the same at all levels and then optimize [the design] for reuse," RMIT's Reed advised. This early decomposition into reusable pieces also results in components that can form the basis of a reuse library, he said.

In Japan, NEC's SEA-1 project takes slices out of Cobol programs, making it easy to reuse existing code, Reed said. This approach helps managers derive the components they identify, he said. At Toshiba, a special 10-person group maintains the component library for 2,000 users, Matsumoto said. "The project manager holds a design review meeting at the beginning of the project. He discusses which existing modules are reusable. After several meetings, the project members agree which modules they are going to reuse," Matsumoto said. "This is a very effective way to get reusability," he said. At Toshiba, 70 percent of the managers have mastered this, he said.

Toshiba strictly controls the components' documentation format to make finding and understanding components as easy as possible, Matsumoto said. "We enforce the builder to write a very detailed user's manual of the content of that reusability module. There is a tight format on how to write," he said. The descriptions are stored on-line with the component and contain keywords for searching. Section managers and project managers jointly agree which new code should be added to the library, he said.

GTE Data Services also has a special reuse group. "We work with development projects to see what projects are dealing with similar domains so we can target application development [at the start] for other projects," Collier said.

In 1987, the reuse group helped projects at their start and then withdrew, Collier said. The projects soon returned to the old methods they were familiar with, so, beginning last January, the group started assigning a member half-time to work on these developments for the entire project life to make sure the projects stay on track, she said.

Software people tend to be very procedural. You need to get the initial confidence level and then you can move forward.

Transferring the technology is also difficult. "There are not many successes in that area," SPC's Yudkin said. Still, having reusable tools and making an large investment in the consortium gives SPC's member companies "a lot of motivation to do so," he said.

Management can ease the transition by "spending as much time preparing the people about the changes as you do on the change itself," AT&T's Sumner said. Yudkin agreed: "Software people tend to be very conservative, very procedural. You have to get an initial confidence level, and then you can move," he said.

"A generic approach doesn't work. There's so much tailoring site to site, company to company," Yudkin said. Even within a company, experience levels differ, he said. "But if someone is succeeding by doing something, everybody follows," Yudkin said.

Training independent of hands-on experience also does not work, Yudkin said. "You have to get real users into the training cycle and then disseminate. You can't train in a vacuum," he said.

"It should be a grass-roots thing," GTE's Collier said. "We've got to convert the unconverted that this is the way to go," she said. GTE Data Services uses several techniques to encourage reuse: a quarterly newsletter describing its benefits, $50 rewards for programmers whose components get reused within a year, and making successful reuse part of project managers' job evaluations and a condition of getting pay raises, Collier said.

One problem in getting programmers to reuse code is that their productivity is often measured by dividing the number of lines of code they originate by the effort in man-months, IBM's Butler said. The more code they reuse, the lower their productivity figures go, he said. IBM is considering a different metric: adding up original lines of code, lines reused from libraries, and original lines used by other programmers before dividing by man-months, Butler said. "Our goal is to produce *less* lines," he said.

Cultural changes. Reuse also requires changes in attitudes. "In the early days of software, when the world abounded with gurus, it was considered almost bad form to take what anyone else had done," AT&T's Sumner said. "We gave that up to some extent when we developed operating systems: We realized we had to take some things for granted," he said.

With its advantages, why is reuse only now getting so much attention? RMIT's Reed said he believes reuse fell out of popularity in the 1960s, especially in the US, for three reasons:

- The introduction of strongly typed languages made subroutines hard to write because of their type restraints.
- The Cobol programming approach then advocated by consultants discouraged multifunction subroutines in favor of simple, reduced coupling — but this hindered reuse across domains.
- There was a now-reversing trend to "deskill the profession" by emphasizing assembly-line techniques.

"Software design should be like a black box, a value-added process. The raw material is the specifications and the architecture," Sumner said. "The term 'software factory' implies that we will employ people with only moderate skills," he said. But that does not mean that programming will be a profession with no challenges, Sumner said. Design, specification, and integration will all require creative thinking, he said.