Casual reuse not sufficient

To the editor:

In his article "Reusability: The Case for Object-Oriented Design" (March, p. 50), Bertrand Meyer discusses different designs of an on-line airline reservation system and discards them on various grounds until he finds one that will make parts of the system reusable: It turns out that the system can be split into a general-purpose screen handler on one hand and into application-specific procedures to handle key-ed data on the other.

In the real world, one of two scenarios would apply:

- If the task were to develop a reservation system, the reusability-conscious approach would be to select a general screen-handler available in the marketplace.
- If the objective were to design a reusable screen handler, the by-product of an application development such as Meyer's example would be totally inadequate. A reusable screen handler must accommodate many possible applications - and must be designed with that breadth in mind. And the documentation for a reusable product would be totally different than that for a built-in subsystem. The effort to make the screen handler reusable would eclipse the effort that went into the design of the handler as a by-product.

A language like Eiffel might be useful to create reusable software, the development of reusable software is a complex engineering issue that should not be confused with the occasional sharing of by-products.

Bo Sanden
Visiting Associate Professor
Wang Institute
Tyngsboro, Mass.

Object-oriented front end

To the editor:

Bertrand Meyer's article on reusability and object-oriented design (March, p. 50) demonstrated one of several new ways to design software.

Jackson has developed two such methods: JSP and JSD. JSP is a data-directed program design method. This method is suitable for designing programs that process specified data. In a data-directed method, the data is described with a generalized grammar, and the program's structure is derived directly from the grammar.

JSP processes differ from Meyer's objects chiefly by lacking inheritance. This is true because Cobol, the most common JSP programming environment, does not support inheritance. However, JSP is easier to use in sophisticated environments like Ada, C, and Icon than in environments like Pascal, Cobol, Basic, Fortran, and assembly. No doubt the use of objects (with inheritance) will make it even easier to use.

JSP fails when the problem's data is not specified. JSD, a dynamic analysis and design method, addresses this lack. Unlike object-oriented design, the JSD entities are not normally built by inheriting features from other objects because it is designed not to require specialized tools. But a JSD designer can use existing entities to create new ones. Meyer's Eiffel gives developers the freedom to make JSD more effective.

In theory, JSP and JSD should act together as a front end to object-oriented design by determining what objects will be needed and how they are structured.

Richard Botting
Chairman, Computer Science Dept.
California State University
San Bernardino, Calif.

The author responds:

Both letters raise interesting issues about reusability and the Eiffel approach. Sanden makes several interesting points about reusability. However, his criticisms are apparently based on two misunderstandings.

The first has to do with the level of abstraction of the reusable tools discussed: classes State and Applications describe abstractions at a higher level than a screen handler; in fact, they must rely on a screen handler to implement variance of the display procedure in State. The object-oriented concepts developed in the article may be applied to the development of such a screen handler, as we have done with Eiffel at Interactive.

The second misunderstanding has to do with the notion of by-products. I agree with Sanden that one does not get reusable software just as a by-product of specialized application development; so the normal scenario is, as he said, to get reusable components from a group specialized in developing such components.

Botting suggests the integration of object-oriented design with Jackson's design method. It is true that some of the reasoning that leads to both approaches is the same — and there was indeed a brief mention of Jackson's work in my article. However, one major difference seems to be the attitude towards and time and sequencing. JSD and JSP rely heavily on a priori specification of scheduling constraints, where certain actions must be executed before certain others.

The Eiffel view tends to consider operations on a data structure as a mere shopping list of available actions, without any prescribed order. This is in line with the notion of abstract data types, and is confirmed by my experience: I have seen too many system architectures that were based on scheduling constraints that had to be changed later in the process, resulting in painful redesign. I informally discussed these issues with Jackson three years ago, and it seems that the two views were hard to reconcile. I would certainly be glad if someone — perhaps Botting — could do it.

Bertrand Meyer
President
Interactive Software Engineering
Santa Barbara, Calif.

Wasted space?

To the editor:

The SDI articles ("Star Wars' Research Feeling Boycott?" and "Early SDI Deployment Considered," Soft News, March, pp. 94-95) are a waste of paper. Use a third of the space next time and leave the crusading journalism to Geraldo Rivera.

John D. Wolf
Programmer
SCS, Inc.
Fort Wayne, Ind.