Reuse: Truth or Fiction (PANEL)

Paul McCullough (Moderator)
Bob Atkinson, *Microsoft Corporation*
Adele Goldberg, *ParcPlace Systems*
Martin Griss, *Hewlett-Packard Laboratories*
John Morrison, *Technology Transfer International, Inc.*

Overview

On the basis of papers at conferences, the trade press, and vendor presentations, software reuse and object-oriented technology are linked hand in hand. What is it about objects that excites discussion about reuse? Operating systems are heavily reused but are not typically written in object-oriented languages. Subroutine libraries (such as the Statistical Package for the Social Sciences) have been in use for years to construct programs; again most are not written in an object-oriented language. So reuse has been around for a long time: Do objects provide better reuse? Why or why not?

There are many models for software reuse: source code libraries, class libraries, compiled libraries, operating systems, and so on. What drives these models of reuse? Practicality concerns? Fears of complexity? Social aspects? Financial? If you're allowed to change the source code is that reuse or is it something else? What are the impediments to the realization of further reuse? License fees? Programmers who just like to write code? What is it that makes something reusable? Is reuse talked about more than it is practiced?

Bob Atkinson

Code reuse is important. Effective reuse of code is perhaps one of the most important factors in being able to deliver the right software product at the right time. There are many possible ways that a piece of existing code might find its way into another piece of software in a manner that might be called "reuse." Code can be copied and pasted, code can be inherited à la C++, source files can be maintained in two simultaneous versions using conditional compilation, code libraries can be linked in (dynamically or statically), and so on. It is our experience, however, that the only *effective* way that code can be shared between multiple clients is in compiled form: without this approach, the unencapsulated source is simply too tempting to programmers.

In order that code can be shared in compiled form, it is critical that the functionality on which the sharing is based be precisely articulated; the programmer must be consciously aware of what he can rely upon and what he cannot. We did this in the past with function libraries relatively easily: we documented the function signatures along with a description of the function semantics.

The reason that object-orientation brings an interesting twist to reuse is we can now consider multiple clients and *multiple* implementations, not just one implementation as with subroutine libraries. So long as an object agrees to support certain functionality, it can be used polymorphically with other implementations of the same functionality. A significant problem to date is that object-oriented languages and development approaches have not focused on articulating these interfaces on which reuse is based. Inheritance is not by it-

self a solution to the problem. In fact, inheritance of implementation is a significant impediment: it is extremely difficult when using implementation inheritance to maintain the mental discipline of being precise about what your subclasses can rely on and what they cannot. This is more than just tagging methods as protected or private; the sequencing of the calls to "self" is an intimate part of the specification of the object. To my knowledge, no significant application framework has ever accomplished this precision: they all ship their source code, not insignificantly because this is the only actual documentation of their semantics. (don't get me wrong: implementation inheritance *is* valuable, just not as a technique for reuse, but instead as a technique for quick initial development.)

Is there a way that we can effectively and polymorphically reuse code? Yes: the key lies in divorcing (in the source code) the interfaces supported by an object that can be relied on for reuse, from the internal implementation of those interfaces. Historically, modules have similarly been divided into public and private pieces; what is different here is the dynamic binding of the functions being invoked according to the object instance in question.

Microsoft has very successfully employed such an approach to reuse in its Object Linking and Embedding (OLE) compound document architecture. For example, Microsoft Word for Windows version 2 supports OLE; it can contain embedded objects. As a result, the graphing and charting feature of Word could be implemented by simply including unchanged in the Word retail package the Microsoft Graph application that was originally built for Microsoft PowerPoint.

Bob Atkinson has been diddling in software much of his adult life, most of it in object-oriented software of one form or another. Currently he is one of the main designers of Microsoft's OLE 2 project.

Adele Goldberg

Effective reuse has to be planned as part of the software development process model. For the most part, few companies have a formal approach to reuse but there are good examples of promising approaches. When a formal process is in place, reuse regardless of the underlying technology helps create long term cost savings.

Adele Goldberg is Chairman of ParcPlace Systems. ParcPlace Systems provides a broad range of object-oriented technologies and services, including development tools for Smalltalk and for C++, that serve the application development needs of corporate programmers. Before that, she managed a research laboratory at Xerox PARC. She has written several books, Smalltalk-80: The Language *with Dave Robson, and* Smalltalk-80: The Interactive Programming Environment, *and led the effort to make Smalltalk-80 available on standard microprocessors. She is editor of the book* The History of Personal Workstations." She *was President of the ACM 1984-86, helped form ACM Press, jointly received the ACM Software System Award (1987), and PC Magazine's Lifetime Achievement Award (1990). She has a Ph.D. (Information Sciences, University of Chicago, 1973).*

Martin Griss

It is widely believed that systematic application of reuse to prototyping, development, and maintenance is one of the most effective ways to significantly improve the software process, shorten time-to-market, improve software quality and application consistency, and reduce development and maintenance costs. While many companies are developing proprietary software libraries, software reuse is not yet a major force in most corporate software development. The reuse community is now beginning to understand that this is largely because effective reuse depends on more socioeconomic than technical factors, while most workers have concentrated on library or language technology. While (improved) object-oriented technology seems to be a very promising vehicle for realizing the dream, and recent work in domain analysis, object-oriented methods, library technology and architectural frameworks should lead to a consistent methodology for domain-specific reuse, there remains a significant effort in these non-technical areas. People need to learn the most effective way of using, investing in and performing cost/benefit analysis on the technology, in setting up new processes, and providing incentives that encourage appropriate change.

Since 1984, Hewlett-Packard (HP) has had several visible software reuse projects in divisions and laboratories. Some use DBMS and library systems to store and distribute software components. Others use Objective-C or C++ to develop class libraries (for UI, data-structures and instrument sub-systems). Several libraries have been widely distributed within HP, and some outside. More recently, several HP entities have started multi-division domain analysis to develop common architectures, components and libraries, for instrument firmware and for chemical and medical system software, producing frameworks and major components to be used in several products.

In 1990, HP initiated a Corporate Engineering Software Reuse program aimed at making software reuse a more significant part of HP's software process. This is a broad, well coordinated effort involving a significant management, process, education, and technology infrastructure. We are not building a single Corporate reuse library, nor do we stress a single language, but instead assist divisional efforts to set up and improve their customized reuse infrastructure and processes. Most of our processes, methods, training and tools are largely independent of particular languages, but OOA/OOD derived methods for domain analysis and component design seem destined to play a significant role.

The program consists of a core team involved in: reuse assessment, and metrics; the development of processes and methods for developing, using and managing reusable work products; the collection of reuse best practices and guidelines in a reuse handbook; and consulting, training, and workshops. To develop and demonstrate a viable economic model, and to provide early validation of methods and benefits of reuse within HP, we have set up several pilot projects in the divisions to help bootstrap and evaluate systematic reuse programs. We provide additional people, funding, and consulting, assist in getting management support, and supply appropriate training.

More recently, we established an HP Laboratories software reuse research program, to explore how component-based software construction processes can be better supported by technology such as hypertext-based reuse library management tools, collaborative work environments, application frameworks, software-bus architectures, generators, and object-oriented analysis and design methods.

Martin L. Griss is a principal laboratory scientist at Hewlett-Packard Laboratories, Palo Alto. He leads research on software reuse, software-bus frameworks, and hypertext-based reuse tools. He works closely with HP Corporate Engineering to systematically introduce software reuse into HP's software development processes. He was previously director of HP's Software Technology Laboratory, researching expert systems, object-oriented databases, programming technology, human-computer interaction, and distributed computing. He serves on HP software engineering councils and is a consultant to HP management on software engineering. Before that he was an associate professor of Computer Science a the University of Utah, working on computer algebra and portable LISP systems. He has published numerous papers, and was an ACM national lecturer. He has a Ph.D. (Physics, University of Illinois, 1971).

John Morrison

Based on a recent study of Japanese software factories which showed high levels of reuse, we conclude the following:

- Currently, Japanese software factories implement systematic reuse using non-object-oriented production technology. However, these same companies are experimenting with object-oriented technology and may adopt it in the future if they are convinced that (1) object-oriented technology is mature; and (2) the investment in re-tooling is worth it.

- A "parts-oriented" or "library-oriented" view of reuse is incomplete. Making reuse work depends on a number of management and engineering factors which impact multiple levels of an organization, and all phases of the development lifecycle.

- The impact of systematic reuse can be quantified in terms of productivity and product quality.

- Software reuse is a key technology area supported by collaborative funding from the Japanese government and industry in order to achieve strategic competitive goals.

John Morrison (Lt. Col., USAF, ret.) is President and founder of Technology Transfer International, Inc. In concert with Adele Goldberg, Professor Michael Cusumano, author of Japan's Software Factories, *and Lawrence Putnam, CEO of Quantitative Software Management (QSM) he recently completed the Japan phase of a world-wide* Market Research and Field Study of Software and Information Factories. *The report,* Software Reuse in Japan, *was published in June, 1992. TTI is planning two follow-on projects: an evaluation of software production in Europe (focusing on object-oriented development of large scale systems) and software production in Russia (focusing on sources of low-cost, high-competence software production in Russia, Byeloruss and the Ukraine).*

Mr. Morrison was formerly the Director of System Engineering and Development for the National Test Bed (NTB) Program, a $550 million Strategic Defense Initiative project developing supercomputer simulations of the Strategic Defense System. During a previous assignment he was the U.S. Joint Command, Control and Communications (C^3) Advisor to the Saudi Arabian Ministry of Defense and Aviation, responsible for Kingdom-wide C^3 integration. Other Air Force assignments included responsibilities for strategic and tactical battle management; operational intelligence and intelligence fusion at the national and tactical levels; interoperability of automated systems and telecommunications; and systems acquisition. He has a Master of Science in Systems Technology (C^3) from the Navel Postgraduate School (1980).