PANEL: OOP in the Real World

Rick DeNatale, *IBM Corp, Cary, NC (chair)*
Charles Irby, *Metaphor*
John LaLonde, *Extrel*
Burton Leathers, *Cognos*
Reed Phillips, *Knowledge Systems Corp.*

Some advocates of OOP have promised that it will make all code reusable, shorten development cycles, remove the applications backlog, cure the common cold and plug the hole in the ozone layer.

How well does actual experience bear this out?

Success stories are published much more often than failures and false starts. This is very unfortunate since the former two are often much more instructive.

This panel will attempt to bring the less successful experiences to light, and hopefully, will provide lessons on how to do better.

Rick DeNatale, IBM Corp., Cary, NC

Let me state that I am dedicated to the promise of OOP. I have been actively pursuing and pushing OOP technology within IBM for the past seven years.

I am concerned that there are some technical problems which are addressed to varying degrees by different languages and implementations.

One particular issue is the degree to which OO implementation allow changes without requiring recompilation. This is an issue for successful commercialization of application frameworks for integrating environments (such as Xerox Star, Apple Lisa, and more recently HP New Wave and to a lesser extent OS/2 PM and Microsoft Windows.)

Different languages and implementations react differently to the various types of changes typically made to a system. Almost all require recompilation of clients when a class structure changes. Some require recompilation when a class structure changes. I once designed a language (ClassC) which required recompilation when adding a message. It was a disaster!

I will argue that OO language and system designers must give much more attention to the life-cycle aspects of their designs and implementation decisions.

Charles Irby, Metaphor

Charles Irby managed the design of the Xerox Star user interface from 1976 to 1979 and played a major role in the software design of the early Xerox Ethernet products. The iconic desktop metaphor was pioneered in the Star user interface. From 1982 to the present Charles has led the product development efforts at Metaphor Computer Systems producing a sophisticated workstation environment in a distributed LAN architecture. In both of these efforts an object oriented user interface was employed and in both cases object oriented techniques were used in the internal implementation. Charles will present an historical view of OOP and discuss Metaphor goals for a next generation OOPS.

John LaLonde, Extrel

Speaking as a project manager and as a software engineer I believe there are substantial benefits from using Object-Oriented Programming (OOP) techniques and tools. I have managed analytical instrumentation development projects that have used OOP and I have developed software for these systems using the Objective-C and C++ languages.

While there is much promise, there are some important considerations related to project management and programmer coordination that should be addressed prior to starting development projects that will use OOP technology. This is especially true for projects with time-critical schedules and projects where a mix of conventional languages and OOP languages will be used.

Much of the interest in OOP starts from a grass-roots level with individual engineers who first study the technology, utilize it for a small prototype project, and then try to convince their management that OOP should be used for other, potentially much larger, projects. Unfortunately, there is a significant probability that their management is not prepared to deal with a very difficult software design methodology which forces change in how software is developed and managed. Moreover, the software engineer who successfully used an OOP language for a small prototype may get overwhelmed when designing OO software on a mucn larger scale. Even with good intentions, desire, and OOP technology, development projects can get seriously sidetracked or delayed due to a lack of proper project planning with respect to OOP. Even worse, these projects may end up with a more severe software maintenance problem than before using OOP!

I would like to stress that OOP technology in of itself is not enough to ensure that the organization employing OOP will automatically reap all of the benefits. The organization must be prepared to manage projects differently than it has in the past. Training and practice prior to the start of a project, small prototype projects prior to full-scale development, OOP design notation consistency, thorough software life-cycle planning, and OOP software configuration management are all important issues that should be addressed prior to starting important development projects that will use OOP.

There are, for the most part, language independent issues, although some language dependencies do creep in (i.e. programmer training and software design consistency with respect to pure vs. bybrid OOP languages.)

Burton Leathers, Cognos

> "It is a very humbling experience to make a multimillion-dollar mistake, but it is also very memorable." Frederick P. Brooks, Jr. *The Mythical Man-Month*

Cognos had the unenviable experience of making a multimillion dollar mistake in its use of Object-Oriented Programming. It was indeed memorable. If there is to be any value in a mistake of such magnitude it is important to learn from it. Cognos has done so and recognizes the importance of sharing its learning experience with others.

In retrospect, it is possible to discern those errors which were a result of naivete and those which were the result of plain stupidity. The errors of naivete were largely a consequence of the limited experience base which was available at the time the decision was made to use an Object-Oriented Language as the basis for product development.

The principal naive error lay in the method for the selection of a programming language. At that time Cognos was involved in creating its own OOPL. This was recognized as foolish and a choice was made from commercially available languages: C++, Eiffel, Objective-C, and Smalltalk. The selection criteria were: object orientedness, performance and portability. Eiffel was selected largely because of the first and last factors. Only limited performace testing was done and the results obtained were ambiguous.

Experience showed that the choice was bad. In the first instance, Eiffel turned out to have serious performance problems. These were foreshadowed in the earlier testing but were not properly pursued. There were many other difficulties encountered in using Eiffel, most of which were related to problems of 'programming in the large'. As the result of our experience, we have expanded the selection criteria from the original three to twelve and now conclude that there is no language which is adequate for our purposes in terms of our revised criteria. This is not to say that all languages are unusable but only that there are no implementations now available which are suited to our specific requirements. The moral of the story is that one should not select a language but a programming environment which is fitted to the task to be performed.

The stupid errors were much more extensive and far raching in their impact. These were by and large, failures of management and many of them were quite independent of the use of an OOPL. Nonetheless, the fact that we were using an OOPL was important because it contributed to an attitude which would not otherwise have existed. It was very true and is still somewhat true that OOP protagonists are true believers. The very real benefits of using OOP are presented in a very one-sided fashion which too often leads to the view that OOP is a panacea. This better than life outlook induced a euphoria in management which caused suspension of the normal procedures and judgment criteria. As a result, the product was inadequately defined, schedules were unrealistic and the design process was wholly inadequate. As difficulties manifested themselves, panic response set in and all the problems so graphically described by Brooks in The Mythical Man-Month appeared. The final insult was that the lack of adequate management controls caused the project to carry on - at great expense - far beyond the time it should have died.

Despite the agony of failure, much good has come of the project. Although the direct experience with OOP was painful, it has not soured either the technical staff or management on OOP. There is a desire and commitment to resume use of OOP whenever it is practical to do so. The product definition process and project management methods have both been substantially improved as a result of the experience. Cognos is producing better products in a more predicable and cost effective fashion.

What advice could we give to someone who wishes to avoid our mistakes? Start small so that if, even after careful selection, OOP turns out to be inappropriate the cost of learning is small. At least during the learning phase, work on relatively small and extremely well defined products. Plan and design with extreme rigour then, as work progresses, monitor progress closely so that deviations can be detected and understood. Above all, use the object-oriented planning, analysis and design tools which are now becoming available. Without careful planning and sound management it is possible to fritter away time and resources in non-productive activities in ways which are inconceivable in terms of traditional programming languages. One can succeed with OOP, but only if one approaches it with hard-headed realism about what it can and cannot do.

Reed Phillips, Knowledge Systems Corp

Knowledge Systems Corporation (KSC) has participated in the delivery of several complex software systems created using object-oriented technologies. Most of the challenges associated with the successful completion of these efforts were related to management, not technology.

We have observed several areas where OOP has presented significant management challenges. Some of these include:

- Setting proper expectations
- Establishing common vocabulary
- Understanding the prototyping process
- Training requirements
- Determining team size and talent mix
- Testing
- Dealing with moral problems when switching technologies
- Understanding the side effects of high performance development environments on
  - prototyping
  - feature expansion
  - documentation

KSC will present case studies from its own experience to highlight these areas of concern.