Smalltalk in the Business World: the Good, the Bad, and the Future (PANEL)

Yen-Ping Shan, *IBM, Software Solutions* (moderator)
Ken Auer, *KSC*
Andrew J. Bear, *AMS*
Jim Adamczyk, *Andersen Consulting*
Adele Goldberg, *ParcPlace Systems*
Tom Love, *IBM*
Dave Thomas, *OTI*

1 Background

During the past few years, Smalltalk has become one of the fastest (if not the fastest) growing programming languages for business computing. Many corporations, large and small, have decided to invest significantly in Smalltalk.

This panel will try to explain this phenomenon by examining the personal experiences of the panelists. By doing so, we hope to provide some insights for corporations that are making decisions on OO technology, point out pitfalls along the way, and identify potential opportunities for technology providers.

Each panelist will present one good example and one *bad example* from his/her experience. The panel discussion will be from a variety of perspectives, including but not limited to:

The Good

- What are the external factors that have made Smalltalk more attractive now than in the past? Hardware speed improvement?
- What are the intrinsic features that make Smalltalk popular? Are they OO or non-OO? In those OO features, what are general to all other OO languages and what are unique to only Smalltalk?

The Bad

- What bad experiences can be encountered in applying Smalltalk to business computing? Why?
- Are those bad experiences caused by characteristics intrinsic to Smalltalk or some other factors?
- What can we do to avoid repeating the bad experiences?

The Future

- What's the future of business computing?
- What are the roles that OO and specifically Smalltalk play in the future business world?
- Will Smalltalk continue to gain popularity in the future? What will be the market share of Smalltalk a year from now? Three years? Ten years?
- What's coming for Smalltalk in the future? Improvements to the base? New tools on top?
- What are the dangers for Smalltalk users in the future?
- What's the recommendation for corporations interested in trying out Smalltalk?
- Can Smalltalk go beyond the business world?

2 Ken Auer

The Good: Smalltalk has been the most productive general purpose development environment there is since it hit the market. In many ways it was ahead of its time. However, when people started realizing that GUIs, Networks, RDBs, and Client/Server were worth using, Smalltalk did not have the interoperability necessary. In the past few years, that story has changed and many are starting to realize that Smalltalk is not just for research prototypes anymore.

The productivity of the environment comes from the fact that it uses a simple, consistent metaphor that exploits the benefits of dynamic binding like no other environment has. Compared to its competition, Smalltalk is very mature in the areas of class libraries, development tools, portability, and low-level language features (memory management, dynamic translation, etc.). There are a lot of environments that demo well, but Smalltalk has had the benefits of time necessary to create a robust environment ready for mission critical applications.

Unlike many of the other "client programming tools" (e.g. Powerbuilder, 4GLs), Smalltalk offers a complete programming solution that is appropriate for more than nice front ends to a database. With good OO Design techniques, large, complex applications can be managed using Smalltalk and ENVY(R)/Developer. As large IS organizations move their systems into the 90s and beyond, COBOL doesn't cut it, 4GLs and the like aren't appropriate for mission critical business systems that exhibit more behavior than a database can provide, and languages like C++ are just too difficult to learn with too little pay-back.

The Bad: Most of the bad experiences I've seen with Smalltalk are not unique to Smalltalk at all, but rather are the result of the dangers and uncertainties of moving to unfamiliar technologies. Although Smalltalk does have some weaknesses that are not present in other environments, its strengths more than compensate for them. Much of the perceived weaknesses are not weaknesses at all, just uncharted territory for the new adopter.

Since new adopters haven't yet fully experienced the benefits, and are not really intimate with the technology, it is sometimes difficult to believe that the benefits are truly there or as significant as they are. Each perceived weakness can seem to be an insurmountable hurdle to managers and developers who are faced with building a mission critical system under tight deadlines with new technology. It's kind of like telling someone who has always taken the bus and has no driver's license, "Here's a Porsche. Now get to such-and-such a place in 3 hours."

Unfortunately, many who adopt new technology do so because their backs are against the wall and the old technology won't cut it. Often, when Smalltalk is brought in, there's already a challenge in addition to the challenge of learning the new technology. Not everyone thrives on double challenges. Lastly, the types of people who are not afraid of new technology are often "visionaries". Visionaries are not often very good at developing and executing solid and realistic implementation plans. Yet these are often the people who are brave enough to be the guinea pigs, and they often don't meet the expectations they've helped to set.

The first project(s) at a company are going to be the guinea pigs. They're going to hit all the surprises and have a rough road to implementation. In addition to being a bit anxious themselves, they are typically going to be surrounded by others who are afraid of what will happen if they succeed. They are venturing into uncharted territory without a lot of positive reinforcement. As they say, "You can tell the pioneers by the arrows in their backs". The best one can do is realize this as the situation, and try to leverage others through hiring or "renting" someone who's been there before and learned from the experience, while you're gaining your own. This is the very premise for KSC's Smalltalk Apprentice Program which has been used by a number of companies who have ultimately been successful with the technology.

The Future: Smalltalk will continue to gain in popularity as it continues to mature and more and more successes are achieved and publicized. I believe it will stay ahead of the competition due to the same factors that are making it successful now (see above) for quite a while, i.e. 5 years or more. However, in order to do this it will have to continue to increase its portability and interoperability with more and more emerging standards, while expanding in other areas of functionality.

I'm a bit concerned that the competition for the rapidly expanding client/server and distributed computing markets will bring extensions and new tools that are not nearly as robust or tractable as the early base class/environment. The demands of getting these extensions (e.g. more interoperability, better deployment options, bells and whistles) and tools (e.g. integrated design and life-cycle tools) to market and the lack of experienced, disciplined engineers paying attention to the quality of how the demands are met make this inevitable. This is true for all development environments in the '90s. However, the consistent development environment of Smalltalk with which these extensions and tools can be developed gives it a clear advantage over the others towards becoming the development environment of choice for business computing by the year 2000.

Unfortunately, there is no guarantee that the vendors will pay attention to the quality of these tools in the same way or the same time-frame the user community would prefer. There are so many aspects of quality, that it is easy for a vendor to pay attention to all of them but the ones the majority of the customers believe to be the most critical and/or urgent. If the "rushed" extensions and tools are improved in the proper aspects of quality over time, current adopters will be severely impacted (and annoyed, at best) if they want to take advantage of the new improvements... real improvements are never fully backward compatible. If the proper aspects of quality are not given the necessary attention, the door will be open for a different development environment of choice for business computing by the year 2000.

Personally, I'm betting these extensions and tools will improve over time, and Smalltalk will continue to gain momentum and become the development environment of choice. (I've been around Smalltalk prior to the first OOPSLA in 1986 and have a pretty good collection of data points with which to extrapolate). Therefore, I'd advise current adopters to realize the amount of change they're going to see in the development environment and that what's state-of-the-art now will have to migrate. Of course the other choice is to choose something that is inferior now and will have no good migration path in the future. To me, the choice is easy... but since I can't afford to retire now, I'll choose Smalltalk.

3. Andrew J. Baer

The Good: Smalltalk is becoming the language of choice for building the mission critical business systems that will give forward thinking companies a competitive edge. There are several factors influencing this move to Smalltalk.

Rapid Development - Smalltalk is more than a language, it is a complete, open, and extensible development environment for building applications (compiler, iterative debugger, code browser, incremental execution environment, etc.). This integrated development environment frees the application developer to focus on the rapid development of the business application instead of focusing on the problems of using a poorly integrated development environment. In addition, the base class library that comes as a standard part of the base development environment provides a very robust start to a reuse library. Finally, the Smalltalk development environment also frees the application developer from many of the low level technical issues, such as memory management, inherent in other object language environments. This allows us to build applications more rapidly, and allow new application developers to transition to Smalltalk more quickly.

Maintainability - The applications we need to build today must be more flexible than ever. Business itself is becoming more complex and business requirements change very rapidly. Object technology applications in general and Smalltalk applications in particular provide the flexible environment we need to handle today's more complex business applications. This flexibility allows us to construct applications that better meet the current business needs and which are much more easily maintained.

Staff Transition - At AMS, one of the big advantages we have found with Smalltalk over C++ is our ability to transition our staff of procedural language designers and developers (primarily COBOL) to object technology. Smalltalk is an English based language while C++ appears to the new developer as a symbol based language. Our procedural developers (even those that are already proficient in C) find Smalltalk a very natural way to build applications. Our experience also indicates that the Smalltalk learning curve is shorter than the C++ learning curve. Also, only those developers that are already exceptional engineers seem to become C++ experts. On the other hand, because Smalltalk insulates the application developer from many of the technical complexities, we find that good engineers become excellent Smalltalkers.

The Bad: Smalltalk can become the standard development environment for building business systems, but prior to that becoming a reality there are still several technical and organizational issues to address.

Class Library Integration - While the integrated Smalltalk development environment provides most of the components necessary to begin application development, building large scale business systems does require integration of additional class libraries. As the number of third party class libraries increases, this coordination problem will increase. The industry needs some mechanism to better coordinate the release of these types of products.

High Performance Distributed Objects - Today's Smalltalk environment is an excellent tool for building three tier client/server systems (workstation, work group server, and enterprise server). However, most of the Smalltalk execution takes place either on the workstation or the work group server. We are only beginning to see industry standard products (based on OMG's CORBA) that will allow us to build highly scalable applications where the Smalltalk (or for that matter C++) objects can seamlessly collaborate in real time with objects that are executing on different platforms. As these distributed object tools mature we will then be able to build high performance transactional systems that take maximum advantage of the combination of hardware and software that comprise our deployment environments.

Integrated Design and Development Tools - The Smalltalk development environment is a highly productive integrated tool environment. However, we do not yet have a robust analysis and design environment integrated with this development environment. Many organizations are working to solve this issue, but until we have a solution, we will have difficulty selling Smalltalk for use on very large scale development efforts. It's not that the current crop of I-CASE tools for the procedural world are all that good, it's just that we now expect our support environment for the entire system life-cycle to be as good as the Smalltalk development environment is for system construction.

Staff Development - The Smalltalk community is being hurt somewhat by its own success. As we have more and more projects using Smalltalk, we need more and more trained Smalltalk designers and developers. As an industry, we have had difficulty producing the number of Smalltalk developers we need to meet the demand. Over the next few years, we need to find better and quicker ways to create the supply to meet this increasing demand.

The Future: At AMS, we are now beginning to see the results of the past couple of year's investment in object technology and Smalltalk. Even in our initial projects (i.e., without the benefits of broad scale reuse), we have begun to see productivity significantly higher than the best projects that used procedural design and development paradigms. However, our use of object technology and Smalltalk can become that much more productive and have a wider application with the addition of several new capabilities. These include:

Domain Libraries - The base Smalltalk class library is an excellent start to a reuse library. However we need to extend that library in several ways. At AMS we are focusing on extending the class library in two directions. First, we have built Object CORE, which is a reusable class library that provides many of the infrastructure classes required for building large business transaction processing systems. On top of these classes, we have also built highly reusable objects for specific domains. Here we are building classes to support applications in the telecommunications, financial services, insurance, government, and education domains. These will allow us, and our customers, to assemble a custom application primarily from reusable components. In the future, AMS and others need to continue the development of these libraries. It is only through the use of these reusable objects that we will see the realization of the promise of object technology.

Language Interoperability - Smalltalk is an excellent tool to use in building mission critical business systems. However it is not the only tool. We need a language independent object messaging engine to provide for a heterogeneous language environment. Again, there is significant progress in this area (primarily IBM's SOM and DSOM), but we need to continue this work to the point where we a have high volume production ready capability.

Cross-Industry Reuse - Many organizations have now seen the benefits of reuse in the Smalltalk environment. However, I suggest that the benefits of reuse could be magnified exponentially if we had a way (both technically and legally) to share our reuse libraries across organizations. We strive to avoid reinventing the wheel within our own company, but I am sure several of us are, in fact, reinventing the wheel across organizations. I would like to see some from of public repository from which all with appropriate access can search and retrieve classes. Of course, we will also need an appropriate plan for compensating and recognizing those that contribute to this cross-industry library.

Repository Search and Query Tools - As the number of classes in our repositories grow, we will need powerful search and query tools to allow designers and developers to ask questions about the repositories contents. Even today, we find most developers have trouble finding the classes they need in our ENVY repository. In the absence of such a search and query tool (and of course, appropriate class information to allow search and query), we find developers deciding that is just as easy to build their own version of a class since it takes less time than looking for a class that already exists. This process will certainly lower productivity, and will reduce application compatibility since applications will have different implementations of common functions.

These are just a few of the new directions that will allow Smalltalk to have a prominent position in the tool chest of the corporate application developer. Object technology is a better way to build today's complex mission critical business systems. Smalltalk is one of the best tools (although not the only tool) to take advantage of object technology. Hopefully, your organization has already made a significant move to object technology and Smalltalk. If not, you should start your transition now so that you won't see your competitors introducing new products and services that you can't even come close to offering.

4. John Davis

MIS systems have traditionally mirrored their organizations. When businesses contained hierarchical organization structures and procedures, the supporting MIS systems had similar characteristics -- very hierarchical and procedural in nature. These traditional business models are being replaced by networked organizations and event-driven, customer focused procedures.

New business systems must reflect and model their organizations. They must become non-procedural. They must be built from interacting pieces. In other words they must become object-oriented.

MIS departments are recognizing that they must undergo a radical re-engineering of their organizations, processes and technologies in order to build these new systems. This change will be even more significant than previous drastic changes such as the transition from batch to on-line, or host to client-server, because it is changing at the same time the business itself is re-engineering.

MIS will carry forward its good things from the past - high level languages, high value on reuse, integrated development, CASE concepts. MIS departments will also carry forward their people as they re-engineer themselves.

I believe that many MIS departments will select Smalltalk today because it most closely matches their mindset and expectations for productivity. It presents one of the best alternatives for starting with objects, yet retaining many of their people and best concepts from their current approaches.

Smalltalk has a window to capture a significant share of future MIS development. However, Smalltalk and associated tools have to scale and grow to reflect the realities MIS departments and business systems. There are many competitors on the horizon who will be well positioned to compete for the MIS market: OOCOBOL, Microsoft and others. Smalltalk can do it, but the vendors have a lot of work to do in a short period of time.

5. Adele Goldberg

The critical assumption of Alan Kay's Dynabook vision was that people would want to program computers. And so Smalltalk was born. Inside Xerox PARC, the term Smalltalk named a wholistic effort to understand hardware media, hardware packaging, programming languages, programmer interaction models, and pedagogy. The inclusion of pedagogy placed importance on ease of getting started ("simple things should be simple") and ability to keep going ("challenging things should be feasible"). As we examine the status and future of the business of Smalltalk, we can hold our discussion independent of the original vision, or measure success in Dynabook terms. Since the former is simply a matter of reviewing revenue and profit forecasts, the latter portends a more interesting discussion.

Two events mark the commercialization of Smalltalk-the recognition that the problem that Smalltalk proposed to solve was hard and so it would help to have more participation, and the willingness by several otherwise-comfortable technologists to go into business. At Xerox PARC, we carried out a multi-vendor project to obtain a review of a specification for implementing Smalltalk, and a document that became a series of publications, including the August 1981 issue of Byte Magazine. More than a decade later, two things are true. First, Smalltalk customers are able to implement the ideas they read about in 1981 to create new information systems, and to do so with measurably better productivity. Second, few new ideas that contribute to the Dynabook software vision have been offered, as the commercial vendors bowed to the necessary but insufficient need to deal with the language, tools and methods of systems engineering. Unquestionably, real customers challenged one's assumptions about such systems issues as appropriate system boundaries, interoperability, distribution, and learnability. They also confirmed the research error that creating a Smalltalk for children has much to do with creating a Smalltalk for adults with independent sources for functional requirements.

The business of Smalltalk began in earnest in the late 1980's with the formation of companies whose success depended on the strategic acceptance of Smalltalk. This success continues to depend on acknowledgment by both vendors and customers that the choice of programming language plays a minor role. The original wholistic view correctly stated that pedagogy, which translates into reusable frameworks and components and techniques for leveraging these, is the critical success factor.

The several-years adventure in the so-called real world has taught four other lessons, all pointing to why Smalltalk may not be successful-but all reparable. First, the Smalltalk community is so emotional as to create hype that makes the selling story unbelievable. Second, the carpet baggers have now entered the market. Smalltalk lives today because its proponents were driven by a vision that sustained the burns created by the C++ flame throwers. Carpet baggers are not persistent objects-they take their money and run. Third, Smalltalk would have fared better in the MIS world if the vendors had duplicated all the programming and project management tools already available to the mainframe COBOL community. There is a bit of a scramble to improve this situation, although I doubt anyone will provide the core dumps one customer once requested.

Fourth, Smalltalk is currently not the language of choice for educational institutes. It should be. The problem is that, unlike C and C++, no already existing curriculum-no long-lived set of examples and class assignments-can be reused to create Smalltalk courses. More established languages have the benefit of concentrated efforts to present computer science and business school faculty with useable and useful materials, including funded ACM curriculum projects. C++, for example, benefits from courses that can be based on modifications of C programming course materials. Smalltalk is the most under-published language. This makes some sense since there is very little about the language to report; Smalltalk suffers from being a remarkably uninteresting language syntactically. Most courses teach syntax and grade programming assignments. Lots of books can do this as well. Smalltalk has not lent itself to this sort of publication.

Most other languages are good enough for teaching about basic algorithms and data structures-the topics of most first courses in programming. Of course, you can teach the same programming concepts using Smalltalk's library of data structures-teaching about the general ideas as well as the specific given the explicit availability of both. But most faculty see that they might have to teach at a more abstract level than they are accustomed to doing, and perceive that there is unnecessary overhead in the use of a system based solely on objects. Perhaps there is a lesson to be learned from the difficulty grammar school teachers experienced when confronted with teaching the new math-it did not fit their model of how math should be taught, and it was not what they had learned and could reuse.

Smalltalk becomes interesting when the topic is systems architecture and design for reuse-both difficult topics to present in book form. Both require more preparation to teach than most teachers of first-year programming courses wish to devote. This situation is compounded by the shift to vocational training by the majority of educational institutions, causing one student at a southeastern university, who was panicked that he would not be able to find a job, to seek help over the Internet to stop his foolish professor from teaching Smalltalk rather than C++.

Real customers look for business maturity and commitment, which should help us weather the storm created by emotion, insufficient tools and methods, and carpet baggers. But more directed effort to get schools teaching Smalltalk is needed. There is not much short-term money to the bottom line to solve this problem, but I do not see Smalltalk persisting without such an effort.

6. Tom Love

We have all seen Smalltalk transformed from the favored plaything of ivory tower computer scientists into the programming tool of choice for corporate computing. Many who are unaware of this transformation think of objective programming in general and Smalltalk specifically as bleeding edge esoterica; in fact, there is no better way to develop large-scale business applications.

The good: Smalltalk's advantages over other programming languages are numerous. First, there is the built-in superiority of objects: reusability, rapid prototyping, and easier maintenance. Second, Smalltalk brings to objects a mature, fully-integrated programming environment. This speeds development and encourages reusability. All of these advantages together lead to higher quality products and lower programming costs.

Businesses that have adopted Smalltalk and used it with care and skill have realized tremendous savings in both time and money. This is well documented. And while successes have far outnumbered failures, there are examples of Smalltalk used poorly, of Smalltalk development projects that have fallen through, and of companies that have gone back to procedural programming. Without proper preparation, programming in Smalltalk can be a daunting-even a doomed-exercise in frustration.

The bad: Many lessons have been learned in twenty years of Smalltalk programming, and it's important to know these to reap its advantages.

Smalltalk has a difficult learning curve. While it is far easier to master than C++, generous training time must be allocated for any switch to Smalltalk programming. This is perhaps the most important investment you will make in object-oriented programming.

Reuse is not free. Sharing objects is difficult without proper planning. Objects must be designed with care and reusability in mind. Failure to heed this will lead to wholesale rewriting of libraries, so that the chief productivity benefit of objects is lost. The object paradigm presents special challenges for management. Managers must be able to read and understand every line of code. Inadequate software management has been a common cause of failure.

Time and space always matter. There is no excuse for bulky programs or slow execution. In this way, object programming is no different from procedural programming.

These lessons can be summarized as The Seven Deadly Sins of Object Programming:

1. Gluttony-Procedural programmers, like Roman Gladiators, are rewarded for their excesses, but too much code will kill many development projects. Bulk is not better. Aim for leanness.

2. Envy-I want what you've got and hate what I've got. The continual introduction of new products makes it easy to develop a love/hate relationship with tools. And while it is important to be quick to adopt those tools that can help speed development and improve quality, others must be avoided because of additional training times, unnecessary complexities, and inappropriateness to the job at hand.

3. Desire (lust)-Inheritance is a powerful tool, but only when used judiciously. The desire to use inheritance indiscriminately must be curbed in favor of a planned approach to structure and reuse. Otherwise you end up with the object equivalent of spaghetti code.

4. Anger-Object programming is a team effort, but when a group or team member indulges in one of the other six Deadly Sins, feelings of anger result. Courtesy and grace are required for successful teamwork.

5. Sloth-Sloppy engineering leads to programs that perform poorly. Program optimization must be pursued with the same diligence as the creation of objects. Performance effects not only the success of the project, but also the entire corporate well-being.

6. Avarice (greed)-Wealth is not gained by hoarding cleverly designed objects and nifty new algorithms or making them difficult for others to obtain. Sharing is one of the primary benefits in object programming.

7. Pride-One is never too important or superior to learn from others or to take the time to become educated in new techniques and tools. Failure to train properly, to reuse the work of others, or to change to meet new challenges will bring certain doom.

These Seven Deadly Programming Sins together can be remembered using the following acronym: G E D ASAP (Get Everything Done As Soon As Possible).

The future: Predicting the future is difficult, yet planning for it is mandatory. Smalltalk has a few obvious shortcomings: too few commercial class libraries available, relatively immature object databases, and a paucity of trained Smalltalk programmers. Clearly, it is only a matter of time until these are overcome and forgotten. But what else will the future bring to Smalltalk programming?

One would hope to see a fully standardized version of Smalltalk, but if the history of UNIX is any indication, multiple standards will likely persist, and there will be intense competition to supply and upgrade the millions of copies of Smalltalk that will be in circulation by the end of the century.

Smalltalk will benefit from advances in hardware. Faster processors, more RAM, and the standard use of large color monitors all will make Smalltalk programming easier. More exciting though are the likely advances in full-motion video conferencing and massively parallel processing. These will make networking essential and multimedia an indispensable element in computing.

Pure object languages, especially Smalltalk, will be the norm and will make widespread adoption of reusable components more likely. Nonlinear increases in both computing and programming power have never happened before. This will be an exciting decade.

7. Dave Thomas

The Pragmatic Reality: Smalltalk is successful today because it has a sound intellectual foundation which has matured in the hands of multiple implementors. researchers and most importantly application developers. If you have an ridiculous schedule on which to develop an application with vague and changing requirements and need to deploy it over night on multiple platforms what other choice is there? In my view Smalltalk makes building complex customer driven applications barely possible.

The more worthy OO competitors such as Eiffel, Trellis, Modula3 have lacked the commercial momentum of the more complex C++ leaving Smalltalk alone as the only serious alternative for those not engaged in systems software. The Smalltalk community has won where the Forth, APL, Lisp communities have failed because the community has focused on building applications and tools which are needed to deliver those applications. Team programming, packaging, practical design approaches, interfaces to other languages, implementation of widely available PC platforms snubbed by the Lisp and APL community that felt they were not "real computers". Smalltalk implementations remained close to each other allowing tedious by still minor conversion between different vendors.

The Good: There is no feeling like that of shipping a successful product or application. Smalltalk technology from multiple vendors makes that possible, but good architecture and disciplined engineering are the real secret to success. Smalltalk allows the iteration of the design which means that much more of the essential architecture can last through the first release.

The major reason that Smalltalk is attractive for complex business and engineering applications is that the level of discourse with Smalltalk is in the application domain. The programming environment despite its age is still the only complete and open environment. To build even simple tools, other technologies force developers to use a plethora of command languages and programming languages to build tools and utilities. Smalltalk remains one of the most productive platforms for interactive tool construction.

The Bad: Successful Smalltalk development requires an architecture and a disciplined team that work as a concurrent engineering team to build a product. In our experience the major problems have been the inability to "see the big picture" and "focus and finish the details".

OOPS and Smalltalk require an investment, if you don't make it you will waste our effort. It is far better to build 3 successful 4-8 person teams than 300 mediocre 3 or 4GL programmers who are using Smalltalk as a 4GL.

There has been an over emphasis of the importance of surface and scaffolding technology such as GUI builders and database interfaces. They are clearly useful and necessary, but not nearly as important as a business architecture and model.

Finally there are some applications such as embedded systems, high performance transaction processing where off the shelf Smalltalk technology just isn't ready. These systems can only be developed with custom technology and mixed C, COBOL programming.

The Smalltalk implementations while stable utilize terminology which is neither meaningful to a computer scientist or an commercial programmer. Many of the libraries are showing the age, not only in their names but in their algorithms and their organization. The existing implementations contain too many bad examples mixed with the many new ones. Unfortunately it seems to be the bad ones that people copy. Despite the existence of research and commercial implementations of Name Scopes (packages, modules, projects, applications) there is still no widespread implementation of separate name spaces which are needed to reduce accidental library name space collision. 34 years after ALGOL 60 you would think that Smalltalk could have a publication format.

Only recently have the commercial Smalltalk platforms stabilized enough to provide a commercial platform form component developers, yet there is no agree upon commercial binary format for them to distribute their libraries.

These weaknesses dim in comparison to the poor quality and performance of the mainstream operating system platforms, and windowing systems. It is really unfortunate we have not moved beyond operating systems into environments which allow software to grow rather than be blasted out of granite and stuck together with mud.

The Future: In 1988 I said that in the 1990s, Smalltalk will run on everything from an "mainframe to a watch". In the next 5 years multiprocessor and distributed Smalltalk environments will provide application level control of complex multi-processors. I remain confident that both will happen within the next two years.

The emerging ANSI Smalltalk standard promises to provide a solid base for the language based on successful vendor implementations specified using modern protocol based techniques which leave open alternative compliant implementations. Smalltalk's will support industry standards and hopefully begin to influence their direction. There will be new tools for refactoring, metrics, testing, analysis and design but most importantly there will be commercial libraries for business objects which raise the level of discourse above "ordered collection" to "sales order" and "waveform".