JOURNAL OF OBJECT TECHNOLOGY

Online at www.jot.fm. Published by ETH Zurich, Chair of Software Engineering ©**JOT, 2002**

Vol. 1, No. 2, July-August 2002

# Learning is a Community Experience

**Adele Goldberg**, Neometron Inc., U.S.A.

## Abstract

Perhaps it is obvious - you do not learn alone, but you do take responsibility for your own education. Part of that responsibility is fulfilled by your creating or finding affinity groups - the collection of places or people that provide a motivating context in which you can learn. Your choice of affinity group reflects the social and experiential nature of the learning process. What interaction style suits you best - push or pull? What timeframe provides the best retention - preparing forward or just-in-time, on-the-job?

Much learning takes place on-the-job, but is retained by taking that experience and storing it as part of a more general knowledge base--the set of enduring principles that allows you to succeed in new situations, using new technologies or applying old ones. The enduring principles of object technology are often lost in the battle over programming language, engineering methodology, or architectural preference. The need to be an expert in transient technology creates an atmosphere of immediate training rather than long-term education, and often creates practitioners crippled by the currency of their expertise. But the lessons learned from the introduction and use of object technology highlight the need to foster a community of learners who help one another develop themselves as practitioners while they develop the practice itself.

This paper is a story about how we might experience learning in the near future, and the role of computers in that experience - not as computer-assisted instruction, but as communications-assisted learning. The story is based on my own history as a promulgator of object technology, and what I think I learned about education from that adventure. Many of us enjoy the learning experience I describe, and so the story is told to encourage wider spread inclusion of supportive learning experience as a regular part of our professional community building. In telling the story, I will share with you a little of my own activities in developing learning as supported by communications within a community context.

## 1 INTRODUCTION

Early in my computer career, I programmed an IBM 1500 in Coursewriter II to teach German to college students. As a graduate student, I worked at Stanford on teaching Symbolic Logic to kids 5-11 years old, and to college students, using online theorem proving software that I either integrated or invented. As part of early efforts to bring Smalltalk to schools, I taught computer literacy to 11- and 12-year olds without the intrusion of a computer in the classroom. Those were the days when the handheld

Cite this column as follows: Adele Goldberg: *Learning is a Community Experience*, in Journal of Object Technology, Vol. 1, No. 2, July-August 2002, pages 7-20.
http://www.jot.fm/issues/issue_2002_07/column1

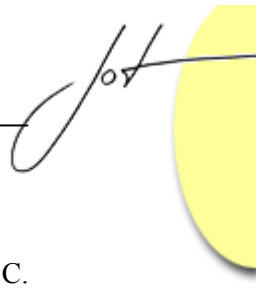
calculators provided in a classroom were chained to the desks. My involvement with education, of course, changed its form when I left research and became a vendor of object technology.

From a technology educator's viewpoint, I saw two conflicting worlds. On the one hand, we worry about the immediate - making sure our students know today's technology, can make optimal use of today's productivity tools, and, for all appearances, can cope with today's technology as it penetrates our daily lives. On the other hand, we worry about having an educated population - one that has learned to learn. What that phrase actually means has been changing in detail over time, but not in intention. We think that an educated person is one who knows a little about a lot, and a lot about some focused subject area - one who reads broadly so is conversant on many topics, but one who holds his or her tongue when the hard data is not there to back up the inclination of that tongue. We believe that an educated person is educable, that is, is ready and able to learn anew. The task of educating for immediate success is not the same as educating for lifetime competence.

This particular conflict was apparent when we first introduced object technology commercially, when we needed to work on both customer training and college courses. The technology itself was nascent. There was not broad based experience in its use. There was certainly not a well-trodden set of examples used to explain the terminology. And there was not the halo that surrounds such technology: the engineering methodologies; the application architectures; the incorporation of the basics in various aspects of a system such as the language, the database, the middleware; nor the understanding of the principles that would endure under implementation change. In short, we had an idea of a new way to practice, but neither a developed notion of skilled practice nor, for that matter, of skilled practitioner. A plan for education needs both targets - a definition of skilled practice and of skilled practitioner.

At the same time, the computer industry was taking over the mindset, if not the economies, of most technologically advanced countries. The personal computer of the early 80s was priced to sell to the average household, not just the average office. And the word processing and spreadsheet applications, coupled with edutainment software, turned an otherwise complex computational device into a personal appliance. Jobs could be found both to build and sell computers as well as to use the computer as a general-purpose productivity tool for the office, almost without regard for profession. Demand to learn about computers increased, but not demand to become a computer scientist. As a result of the inclusion of computing technology in everyday economics, computer science has become one of the largest and most active disciplines.

Much of the demand to learn about computers is to be able to use and extend productivity tools and to be able to program, and not necessarily to understand a mathematical theory of computation. Much of this teaching is therefore and preferably done outside a university computer science department.



> As an example, I point to the current effort at George Mason University in Washington D.C. to instill computer literacy in the liberal arts curriculum, independent of the computer science department, and with a focus on use of electronic tools for research and evaluation, spreadsheets and data bases, analysis tools, and geographic information systems for handling spatial data [http://www.educause.edu/ir/library/pdf/EQM0041.pdf].
>
> Widespread access to computer literacy courses sometimes opens the debate on whether someone is in the computer business. One colleague, who designs microprocessors, met a woman at a party who, on learning that he was in the computer industry, concurred that she was too - because she used Microsoft Word.

The introduction of object technology changed the educational situation somewhat, opening the door for more computer users to participate in developing computer applications through reuse and refinement. Moreover, object technology offered an opportunity to change the level at which we discuss information representation and computation - the very software we use to write applications could be used to write descriptions about how to structure and write applications. This opportunity fostered renewed excitement around research topics such as reusability, architectural patterns, and graphical tools for generating portable applications. The potential for intellectual studies was, and still is, extraordinary given the ability to write software in which the architecture of a system is readable, software that describes itself in terms of structure and intended use.

The promise of object technology was even more exciting in the commercial sector, where the increased use of computers demanded increased numbers of practitioners who would be more able to produce complex software through the composition of reusable components, as well as more able to maintain this complex software given a presumed localization of impact of change. This commercialization rang dollar signs in the eyes of both the professional object technologists as well as the students.

> In the late 80s, with my former company ParcPlace Systems already in business and committed to the development of practices making use of object technology and to the development of the practitioners who would be transitioning to the technology, I received a frantic email. The short note was from a college student in Florida. Desperately he asked me to talk with his professors to get them to stop teaching Smalltalk instead of C++. After all, he wanted to get a well-paid job when he finished school, and so, he said, he needed to learn to program in C++. I answered with a simple question: are you attending a vocational training institute, or are you interested in getting a more general education? I received no response.

Learning to learn was and often still is subordinated to learning transient technology, and only just enough to get a well-paid job the first year out of school.

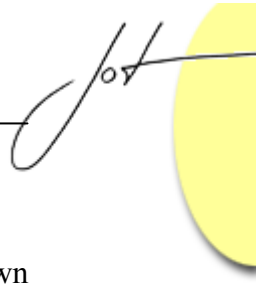
It is likely that the experiences we encountered in the introduction of object technology will not be unique in the future history of computer science. We will have to

accept that the practical will overshadow the intellectual, in schools of higher education as well as in vocational institutes. We certainly have to accept that technology continually changes, perhaps even radically again, that the practitioners will need to learn anew, and that the teachers of students and of practitioners will also have to learn anew. In considering teaching about computers and teaching with computers, we have to consider the impact on teachers as well as students. We will have to invent and encourage longer-term on-the-job approaches to acquiring and honing our art and our skills, and this applies as well to the on-the-job training for teachers. In addition, we will need to encourage and make it cost effective for employers to support these on-the-job approaches - including university employers whose lack of release time for preparing new curriculum certainly contributed to the university inclination towards C++, to be able to reuse problem sets already in use for teaching C, and to be able to assert that other courses, dependent on student programming skills, would not have to change. There is a rush now to put courses on line, to create electronic universities in order to increase the available target market. There are also commercial companies with online courses, either taking over elementary- and secondary-school instruction in the U.S. and other nations, or coordinating lesson planning, homework grading, reporting, and formal assessments.

This message of the need to educate students today for lifelong competence has been delivered by many people and is certainly not new to this paper. For example, there is a book you might check out, entitled *Being Fluent with Information Technology*, offered a few years ago by the U.S. National Academy of Sciences (NAS) [http://books.nap.edu/html/beingfluent/es.html]. This book addresses the question: What should everyone know about information technology in order to use it more effectively now and in the future?

> "This requirement of a deeper understanding than is implied by the rudimentary term 'computer literacy' motivated the committee [that prepared the NAS report on which the book is based] to adopt "fluency" as a term connoting a higher level of competency. People fluent with information technology (*FIT persons*) are able to express themselves creatively, to reformulate knowledge, and to synthesize new information. Fluency with information technology [i.e., what this report calls *FITness*] entails a process of lifelong learning in which individuals continually apply what they know to adapt to change and acquire more knowledge to be more effective at applying information technology to their work and personal lives [emphasis added]."

The NAS report focuses on three kinds of knowledge: immediate skills, foundational concepts, and ability to abstract. Acquiring such knowledge is considered a challenge because "FITness is fundamentally integrative, calling upon an individual to coordinate information and skills with respect to multiple dimensions of a problem and to make overall judgments and decisions taking all such information into account, …". The report therefore concludes that "a project-based approach to developing FITness is most appropriate" with the ultimate goal to "provide students with a sufficiently complete



foundation of the three types of knowledge that they can 'learn the rest of it' on their own as the need arises throughout life".

My thesis however is that learning on your own is preposterously hard given the quantity of new material regularly generated. Both filtering and selection techniques have to be taught so as to be able to focus without losing sight of the intellectually stimulating neighborhoods that surround any focus of attention.

## 2 EDUCATING ABOUT OBJECT TECHNOLOGY

The introduction of object technology was surrounded by a mystique that suddenly the impossible would be possible, what was hard to do would be easy. The question, asked even in the public press, was: Does using an object-oriented approach to design and to implement systems simplify the work or make the work more complex? In those days, I argued (and still do) that the work of designing and implementing systems had become inherently complex, because our ambitions for office and process automation, and information management and communications, had increased. This inherent complexity cannot be hidden. The role of information technology in the enterprise is to make explicit our understanding of the enterprise itself, and that means exposing the inherent complexities of that enterprise. Objects hide implementation complexity, yet they expose modeling complexity. The issues we have to deal with are whether such exposure ultimately encourages simplification of the model, in terms, say, of business objects, and whether the lack of abstraction skills is detrimental to the task.

The long time value of object technology is really in maintenance: through improved modeling in terms of well-factored object-based architectures, we strive to ensure that the impact of change will be better understood - more predictable in that truly incremental changes would be additive, while replacement can be safer if object interfaces are retained. The problem with using maintenance as the selling proposition was that the systems had to be built in the first place, before managers would even discuss the potential maintenance benefits; the training and staffing issues delayed progress.

The overwhelming problem for educators was that object technology did not improve the ability to design data structures or algorithms, but rather the ability to build complex systems. And teaching about complexity involved appealing to the student's ability to understand and model problem spaces for which they likely had no knowledge. Moreover, complex systems are built in the context of a team - whether small or large. Teaching teamwork was and is acceptable to schools, but only in so much as the team project can be done in the timeframe of a one- or two-semester course and the team consists only of students in the class.

This format is decidedly not how real teams are formed, nor how schedules are set. Educators really had (and have) no choice. They can either fall back on the teaching of fundamentals - assuming the enduring principles can be identified and example course

exercises devised, or they can set up symbiotic relationships with nearby companies to which the students can be apprenticed.

Even agreement on enduring principles is still an open debate. You can decide for yourself about whether there is agreement by looking, for example, at the proposal of the ACM/IEEE Committee on Computing Curricula 2001 (CC2001) for systems concepts to be covered in the introductory courses. The part of the proposal I present here is based on the strategy of "objects first" followed by a "systems approach" for intermediate courses (as versus "imperative programming first" or "functional programming first" and then a "topics-based" or "web-based" approach). What I picked out from the extensive course designs provided via the ACM Web site is:

recursion
encapsulation and information hiding
separation of behavior and implementation
polymorphism
relationships among encapsulated components (classes, inheritance)
collection classes and iteration protocols
virtual machines
layers of abstraction

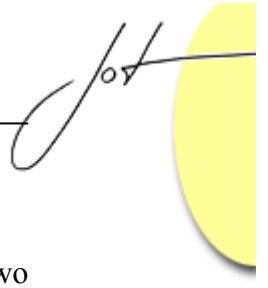
in addition to the usual fundamentals of algorithms and data structures, and the inclusion of design patterns and component-based programming as part of the advanced courses in the software design core. In contrast, when I was asked what I thought were enduring principles, the first thing that came to mind was factoring for customizability - which is not on this list, although might be inferred from several of the included items.

The NAS-suggested curriculum components are less technically specific. The ten suggested intellectual capabilities are interesting, including in them the ability to manage complexity, test a solution, organize and navigate information structures and evaluate information, collaborate, and communicate to others.

So according to the NAS, even when we train, what we have to train about is really general individual- and group-based problem solving skills, and not the idiosyncratic nature of any one transient solution.

> There is another point of view that says that computer science education is not about becoming a skilled programmer, and so one's programming skills are not the real measure of success as a computer scientist. Skilled programming might actually be better correlated with personality (and as my colleague David Leibs likes to joke: programming is a personality disorder that you can test for). For example, if you gathered a list of the good programmers you know, you likely find musicians, woodworkers, and introverts.

Aside from the question of inherent complexity of the problems to be solved and the lack of agreement on enduring principles to be taught, there is the question of quantity of information to acquire before feeling able to get started. Whether in the classroom - where the problem was somewhat easier to manage by picking toy problems with toy tool



sets - or in the workplace, customers asked proponents of object technology two questions: Where do we start? Where do the objects come from?

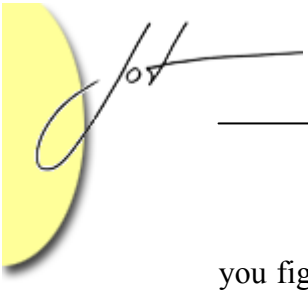
To answer the question of "Where do we start?" we have to know what you want to do. Again, that is easy in the context of a prepared course and toy exercises. Pedagogically, you pick the enduring principles, pick the several examples that illustrate the principles, and then constrain the environment to those elements that can contribute to a solution. You can then evolve the environment, adding or changing elements in order to keep the student focused. This approach is essentially the one we took with LearningWorks, a project from the mid 90s to create a special context for teaching about the basics of system architecting. The LearningWorks environment consists of browsers and inspectors and debuggers that "know" which objects and which tools should be visible to the student within the context of any one instructional exercise. The incremental control given to the curriculum designers permits the course environment to evolve to a fully functional and commercially available environment, without that initial shock of seeing everything at once typical of commercial object language systems. LearningWorks continues to be used with good success by the UK Open University in a first course in computer science.

In the context of business applications, to answer the question "Where do you start?" requires a discipline of requirements analysis, including: characterizing the users, and negotiating on features and functions, timeline, and resources. None of this is peculiar to object technology, although in truth there seemed in the early days to be a fear that object technology changed these software engineering fundamentals. It did not.

To answer the question of "Where do the objects come from?" we have a mixed solution history. We have a myriad of object analysis techniques that have been offered, including scenario or use case approaches, which themselves suffer from the same problem - Where do we start, and Where do the choice of scenarios or use cases come from? I have witnessed the authors of these analysis techniques, when confronted with a new analysis situation, stammer and ask to come back the next day. The fact is that the problem is hard, and most likely there is not just one answer. Where the system situation is familiar - as in constructing a graphical interface or a three-tiered architecture - we rightfully expect to find solution frameworks that can be customized. The answer to the question "Where do the objects come from?" is then answered: From an expert, or from a library constructed by a set of experts.

The notion that you ask someone in order to find the objects - or get assistance in selecting an architecture, a tool, or a subsystem - is perfectly acceptable in the work world. The community of experts we can access because of Internet-based communications capability is already proving to be a key to solving the question of educating for lifelong competence - you ask someone you find, including someone from the Internet.

The trouble is that asking someone for the answer is not generally acceptable in the classroom, where individual accomplishment is graded and the honor system demands



you figure things out for yourself. The undergraduate computer science education, again exemplified by the ACM/IEEE efforts to define core competency and curricula, recognizes that students must learn to work as members of a team. The CC2001 proposal recommends that students be able to work in teams. This general requirement for all students is included along with mathematical rigor, the scientific method, familiarity with applications, and communications skills. The draft proposal states that:

> "Few computer professionals can expect to work in isolation for very much of the time. Software projects are usually implemented by groups of people working together as a team. Computer science students therefore need to learn about the mechanics and dynamics of effective team participation as part of their undergraduate education. …
>
> "To ensure that students have the opportunity to acquire these skills as undergraduates, the CC2001 Task Force recommends that all computer science programs include the following:
>
> - Opportunities to work in teams beginning relatively early in the curriculum.
> - A significant project that involves a complex implementation task in which both the design and implementation are undertaken by a small student team. This project is often scheduled for the last year of undergraduate study, where it can serve as a *capstone* [emphasis added] for the undergraduate experience. "

The problem of course is that the team consists of the members of the class - all peers, no mentors, all expected to pretty much contribute equally as they are all expected to have reached pretty much the same skill level. Remember, these are undergraduates, and not graduates who might have had work experience either before or while returning to the classroom.

The educational dilemma is clear. You learn to program and participate fully on a team through an apprenticeship program. Unfortunately, today's employers do not see themselves hiring apprentices, nor do the salary expectations of new graduates support such a role (which historically is allocated a lower salary in exchange for on-the-job training). We have little choice but to provide the apprenticeship in the context of online communities, in which software engineers can participate and learn on their own time.

Real teams are not so homogeneous as these undergraduate project teams. The learning that occurs on a team is a result of the differences in the team members - team members learn from one another because they know different things needed to get the job done, and are stimulated by the desire to discuss their knowledge and know-how with their team members (whether because we are individually naturally inclined to teach or perhaps to show off; regardless, we enjoy learning together).

## 3 AFFINITY GROUPS

Perhaps it is obvious - you do not learn alone, but you do take responsibility for your own education. Part of this responsibility is fulfilled when your create or find affinity groups - the collection of places or people that provide a motivating context in which you can learn. Your choice of affinity group reflects the social and experiential nature of your learning process. Given that tasks are distributed among team members, that teams are often small so that technical details have to be acquired and practiced by individuals independently, it is natural to seek help from outside the team.

A practicing software engineer is a member of at least two teams: the ones to which your employer assigns you - let's call this your Project Teams; and the ones to which you assign yourself - your Affinity Groups. In unusual cases, these might be the same.

Here is a small experiment to try when in a room with a group of 10-30 people.

Step 1. First, ask each individual to decide whether the group should consider the individual to be an expert in object technology. If you are an expert, then signal (by standing up) and remaining standing. (Note definition of "expert" vs proficient professional or virtuoso as defined by Peter Denning in the August 2001 issue of the *CACM*.)

Step 2. The non-experts, those who did not stand up, should find an expert to be physically near (they should physically leave their seats and move near the expert so as to be able to converse).

Step 3. Each non-experts should think about a question that relates to object technology, one whose answer the non-expert does not know and that is a reason why he or she is not to be considered an expert. Ask this question of the expert near you.

Step 4. If the expert was asked a question he or she could not answer, could the expert now designate himself or herself to be a non-expert and look for an expert who has the answer? After finding an answer, the individual can decide whether to stand up again as an expert.

An interesting outcome of this experiment is that, ultimately everyone designates him- or herself as a non-expert at some time in the process, with minor exception. And the exception is typically a developer who, although acknowledged to be an expert, can always find an unknown as a learning challenge but who, in the context of the experiment, was not stumped.

What we were doing in the experiment is demonstrating the formation and evolution of affinity groups.

The second interesting outcome is how quickly people will agree on what knowledge is representative of "expertise" in object technology, but that the experts will inevitably argue about whether a particular question is something an expert should know, or whether the question is properly formed, or whether a particular answer is right, or

whether there even is a right answer. Affinity groups are places to have these debates as well.

Notice that we could do this exercise over and over, repeating Steps 2-4, as experts change place with non-experts repeatedly, and the questions and answers are propagated through the room. In some cases, the self-proclaimed non-experts will realize they are experts, just not about everything. The knowledge is distributed in the group, but accessible to everyone. What needs to happen is better recognition of where the knowledge is held. This notion, that knowledge is distributed, that everyone knows it is distributed, and that everyone knows where it is located (that is, who in the group has what knowledge) forms the basis of the psychology of distributed cognition [see my column in the May 2002 issue of *JOT*]. Of course, since we are not likely to be all in the same room, better communications facilities are needed to deal with geographic dispersal of knowledge.

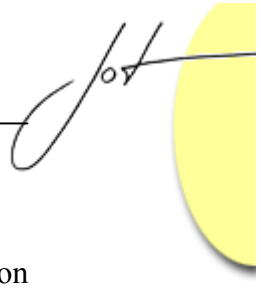
An affinity group is a simple idea, something you do naturally. Study groups in school - people who share a common need to learn a topic and so gather together to help one another study. Professional development groups - for example, for school-based teachers to help one another in lesson planning and delivery. On the Internet, affinity groups form regularly - they are the combination of people who go to a Web site for information, who comment on that information expecting to influence the opinions of other site visitors. There are many interesting online sites for software engineers, where you can read about technology, ask for assistance, download solutions, and give assistance. Online support of this nature started with Usenet and continues with such web sites as SourceForge.

An affinity group is a source of information and assistance, either as pointers of what to read and what is new and interesting to learn, or, more extensively, as an exchange of assistance on how to accomplish some task. Many users of such sites are passive visitors. They browse and read, but rarely contribute their own material. Typically such visitors are called *lurkers*. Lurkers were visibly obvious in a physical meeting room, but are not so visible in an online situation.

> Back to the experiment. Consider whether participants introduced themselves when asking or answering a question. Did it matter? How many potential participants just sat and listened?

## 4 COMMUNITIES

An affinity group is not a community. It is perfectly reasonable to participate in an affinity group anonymously. It is perfectly acceptable to lurk. It is not reasonable to do so in a community, which has a greater emphasis on shared social or experiential responsibility. In a community, shared interest in a topic is extended to a shared interest



in the process and results of a collaborative effort. (See for example a similar position postured by the Institute for Research on Learning in Palo Alto, California, which claims that social closeness enables learning).

One reason I prefer to focus on "community" is that I believe it was the sense of community that helped to build the original market opportunity for object technology. Certainly this was true of the Smalltalk subset of that market, which grew out of the personal and commercial interests of people connected primarily by their belief that a particular type of technology could solve a very hard problem, and their collaborations to create and sell the many different parts of and implementations of the solution. The existence of choices enhances a market. It was never a calm, orderly, nor formally organized community. I am sure I am not alone in saying that many of the community members did not even like, trust, or personally respect each other (although, with some small exception, they might have retained some professional respect). I am also sure that this lack of trust was one of the reasons for its demise. (Specifically, the lack of trust created a schism that made it easier for the Java juggernaut to split the Smalltalk community's resolve. Foolish lack of recognition that membership in the Smalltalk community did not automatically give a vendor access to the Java community further contributed to market problems.)

Another reason I focus on community is that I believe that if we can understand how to nurture a community supported over the Internet, we can extend what has already been started naturally - affinity groups - and so we can build a way in which an individual can become part of a lifelong virtual classroom.

I currently am involved in two activities related to the use of technology in education, involving community building: programmatic oversight on behalf of the Board of the SF Exploratorium [http://www.exploratorium.edu/], a center for research on learning and teaching dedicated to an inquiry approach to learning; and with an effort centered at the Charles A. Dana Center at the University of Texas at Austin to bring online professional development support to teachers of Advanced Placement and college preparatory courses in U.S. high schools. At the Exploratorium, we have been developing both the notion of virtual museum as well as a networked museum (a global community of similar museums and learning centers that can share public exhibits, teacher education courses, and research). The Exploratorium, in inventing its future online access to its inquiry-based pedagogy, has considered linking kids directly to research scientists in the field through its Webcast facility, to see math and science exploration as conducted in the field by experts. The University of Colorado, in conjunction with the University of Michigan's Weather Underground and Weather Channel, has a similar program called the Kids as Global Scientists program [http://www.onesky.umich.edu/kgs/htdocs/home1.html]. Students use the Internet to learn about the weather and environmental science issues as they also get to know other students from different regions of the United States and the world. There are student-directed discussions, real-time data submissions, predictions about storms, and comparisons with expert predictions and actual events. Students have online access to experts, teachers, and other students, to ask questions and

participate in discussions. These two examples clearly treat education as a groupware project.

> By the way, there is a movement afoot in the U.S. to encourage project-based learning in the pre-college grades. The proposed example projects that I have been discovering focus on business and personal uses of technology. I recently came across one emphasizing network-based information retrieval and presentation in which each student makes a home page with particular characteristics, considers which newsgroups are useful for different purposes, and then goes on a treasure hunt to answer some questions. One of these questions was: What is Donald Duck's middle name?
>
> When I searched the Web using Google to find this bit of treasure, I found only one helpful reference - it was a threaded discussion located at the Uppsala University website, in which a mother was asking anyone out there, on behalf of her daughter who was given this assignment. And someone answered that in one of the Donald Duck movies, Donald was drafted into the U.S. Army and there was a close up of his draft card that showed his middle name[1].

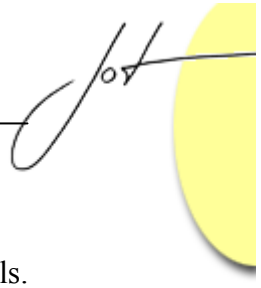
## 5 A TEACHERS' PROFESSIONAL DEVELOPMENT COMMUNITY

The lesson learned from our experience in fostering the object technology market is that, whenever we develop a community related to professional work, our strength comes from nurturing a community of learners who help one another develop themselves as practitioners, while they develop the practice itself.

The design criteria for online communities, based on the distributed cognition work of Edwin Hutchins [Hutchins, 1995], represents a community as a group of members who form a team with a shared purpose. To accomplish that purpose, members need a shared vocabulary by which information and distribution of labor can be represented and discussed, and a shared awareness about tasks and tools and skills. Knowledge flows in the community: what work is to be done, how that work is allocated to individuals, how results are gathered and combined, how tools are used individually and collectively, how groups use tools to coordinate and cooperate, and how technology resources and communications channels are organized. An individual maintains active awareness of his or her own work, work context, and the work of others; such awareness contributes to shared learning.

The three factors that affect the structure of the work and what actually transpires are: the conduct of the activity, the development of the practitioner, and the development of the practice. From prior experience, we create a community by describing a model that defines an initial interaction process and member roles in carrying out that process, and

[1] The answer is Fauntleroy.



assign tools for individuals and tools for coordinating the contributions of individuals. The model describes how information is shared, and how ongoing work on one project can be connected to and influenced by work on another project.

We have used this modeling approach in the design of a community to support teachers responsible for advanced high school (college preparatory) courses. A teacher professional development community is really two interlocked projects. The first project is the development of core curricula on the subject the teacher must be able to teach, described primarily in terms of reviews, interactive explorations, and informal and formal assessment of student competence. Here the curriculum developers participate in a community of practice, expecting to learn from the use of their work products.

The second project is the delivery of the core curricula to both students and teachers. The teachers help one another prepare for classroom-based interactions, thereby improving their teaching practice as well as their individual skills. The real goal, of course, is the success of their students on formal assessments, which are practiced online by the students. Student behavior and practice test results are reported to the teachers so they can understand what the students are doing, and be alerted to any need for special attention. In addition, the teachers have access to professional development services in the form of video tutorials, featured essays, and real-time expert consultations. Each of these is cast in the context of a forum in which the teachers can share their experience in practicing what is preached.

Experience in the second project creates improvements in the core curricula, and especially in the advise given to the teachers - classroom presentation strategies, classroom management suggestions, explanations of how to handle anticipated student concerns, and other stories about actual teaching experience. Stories gleamed from classroom experience can be shared, especially improved understanding of the where and why of student misunderstandings, which can then be translated into additional or modified content.

In designing the teachers' professional community, we are interested in those features and functions that serve to facilitate an exchange of skills as well as content. These features should develop trust among the members who have a shared interest in building one another's skills, as well as trust in designated experts who create video tutorials, real-time online consultations, featured essays about classroom experience, and answers to teachers' questions.

Experimentally, we are interested in learning more about how teachers self organize to set up workable communications channels, and what styles of communications best serve the teachers given diversity in geographic location, local school and school district expectations for planning and reporting, frequency with which issues or ideas can be exchanged, and the ability to describe their classroom experience in following the core curriculum given differences in student capacity to perform in these advanced courses. There is diversity in background and skills of the teachers, not just the students. Some teachers are master teachers of these advanced courses, some are designated

professionally as graders of the special exams, while others have little background preparation to teach the subject area. Just as we had to do for students, workers, and so-called experts in object technology, our goal is to help the teachers help one another learn on the job and learn on demand, with activities that support their ability to reflect on practical experiences. The first use of this new community in fifty high schools in the United States will occur this Fall, as part of a research grant.

These few examples of my current educational activities are all attempts to understand how to structure a community so that we learn how to evolve that structure based on members' actual actions. I am interested in how to move away from the computer as the center of attention, and to focus on how community-based communications create a context for lifelong learning.

*Paper originally presented in a Workshop on Education at NetObjectDays, Erfurt, Germany, September, 2001*

## About the author

Dr. Adele Goldberg is a director of Neometron, Inc. Neometron is a California-based consultancy working towards the use of virtual communities to support more effective teamwork. (http://www.neometron.com). Previously, she was the Chairman of the Board and a Founder of ParcPlace Systems, Inc. She served as CEO and President from inception until 1991, and Chairman until 1995. Prior to the creation of ParcPlace, Dr. Goldberg received a Ph.D. in Information Science from the University of Chicago and spent 14 years as researcher and laboratory manager at the Xerox Palo Alto Research Center. From 1984-1986, she served as president of the ACM, the U.S. computer professional society.