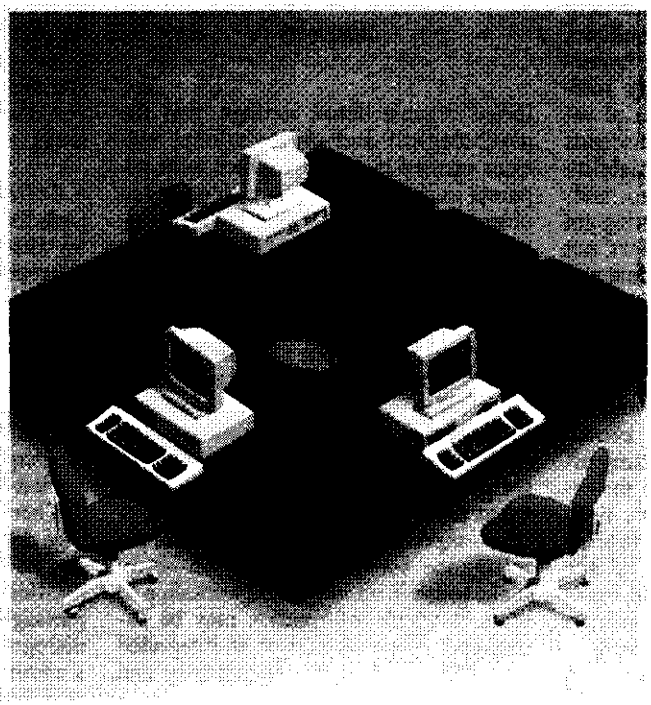
Working Together

The "human system" and the "tool system" are equally important in computer-supported cooperative work

Douglas Engelbart and Harvey Lehtman



The emergence of the personal computer as a major presence in the 1970s and 1980s led to tremendous increases in personal productivity and creativity. It also caused setbacks in the development of tools aimed at increasing organizational effectiveness—tools developed on the older timesharing systems.

To some extent, the personal computer was a reaction to the overloaded and frustrating timesharing systems of the day. In emphasizing the power of the individual, the personal computer revolution turned its back on those tools that led to the empowering of both co-located and distributed work groups collaborating simultaneously and over time on common knowledge work.

The introduction of local- and wide-area networks into the personal computer environment and the development of mail systems are leading toward some of the directions explored on the earlier systems. However, some of the experiences of those earlier pioneering systems should be considered anew in evolving newer collaborative environments.

Computer Supported Cooperative Work (CSCW) deals with the study and development of systems that encourage organizational collaboration. Most groupware products fall under this classification. CSCW projects can be classified into three categories: tools for augmenting collaboration and problem solving within a group geographically co-located in real time (e.g., CoLab at Xerox Palo Alto Research Center); real-time tools for collaboration among people who are geographically distributed; and tools for asynchronous collaboration among teams distributed geographically.

In our work at the Augmentation Research Center (ARC) at the Stanford Research Institute (SRI) International beginning in the mid-1960s, we developed a system called NLS (On-Line System) and tools that supported these forms of collaboration. However, we placed the greatest emphasis on collaboration among people doing their work in an asynchronous, geographically distributed manner.

Our original goal at ARC was to "augment" individuals doing knowledge work. (See the text box "The NLS/Augment Architecture" on page 247.) In fact, some of the tools, techniques, and artifacts we developed then have become widely used in personal computer environments. These include full-screen windowed editing systems, mouse-controlled cursors, hypertextual linking of documents, and consistent user interactions across all aspects of a system. As timesharing systems and then wide-area networks (such

continued

ILLUSTRATION: ROBERT TINNEY © 1988

as the ARPANET) were introduced, the domain we attempted to augment widened to include groups collaborating in the same place, as well as over distances bridged by the networks and over time bridged by tools for creating a recorded dialogue among the collaborators.

One of the key strategies at ARC was the notion of bootstrapping: making use of available technology to create tools, techniques, and methodologies for knowledge workers in general, and the ARC group in particular, to use in further development of the tools. We served as the developers of the technologies, as well as the subjects for the analysis and evaluation of the augmentation system we had been developing. Many of the surface features of the system appeared in fancier dress as bit-mapped graphical hardware that became available first at Xerox, then later, much more widely, at Apple.

While it was exciting to see bits and pieces of the original NLS, now called the Augment system, appear commercially over the years, many elements of the system's conceptual core have only recently been recognized: outline editors (for easy manipulation of ideas); hypertextual linking capabilities fully integrated into the system; a system of recorded group dialogue that transcends most mail systems; user programmability and customizability of the system; and, most important, tools for augmenting not just individual knowledge workers but also teams of people both coresident and distributed over the world interacting through a networked environment.

We thought that success in creating tools for collaborative knowledge work was essential to the necessary evolution of work groups in increasingly knowledge-rich societies and to increasing organizational effectiveness. Until the recent growing interest in CSCW, most developers limited their analyses to technical issues and ignored the social and organizational implications of the introduction of their tools; such considerations were, however, key to our work.

There is growing recognition that some of the barriers to acceptance of fully integrated systems for augmenting groups of knowledge workers may be more significantly social, not solely technical. The availability of rapidly evolving new technologies implies the need for concomitant evolution in the ways in which work is done in local and geographically distributed groups.

ARC experienced this phenomenon continuously. The bootstrapping approach, so important to the continuing evolution of the system, caused us to constantly undercut our world: As soon as we became used to ways of doing things, we replaced platforms to which we were just becoming accustomed. We needed to learn new roles, change attitudes, and adopt different methods because of growth in the technological system we ourselves produced.

We brought in psychologists and social scientists to serve as observers and facilitators. They were as important to our team as the hardware and software developers. The resistance to change, which we soon realized was an essential part of introducing new technologies into established organizational settings, and the psychological and organizational tensions created by that resistance were apparent in ourselves. We were required to observe ourselves in order to create appropriate methodologies and procedures to go along with our evolving computer technologies.

We brought in psychologists and sociologists to serve as observers.

Our lab was concerned with *augmentation*, not automation. The choice of this term was significant. Aspects other than introducing new technological tools into the workspace (e.g., conventions, methods, and roles) are at least as important to the success of any CSCW system. The elegant tools available now and in the future—superlative graphics, artificial intelligence services, and so on—only make sense in an integrated workshop of tools in which information may be exchanged. The tools in such an integrated workshop need to be conceptually and procedurally consistent.

We expect that as tools are introduced and used, a co-evolution will occur between the tools and the people using them. Thus, WYSIWYG systems eased the acceptance of computer systems by nontechnically oriented users; however, these systems produce a map of what you would see on paper as opposed to a hyperdocument with structural links evolving over time. We are now seeing the increasing acceptance of other presentation metaphors (such as Apple's HyperCard and Owl International's Guide) incorporating some of the nonlinear linking capabilities that were present in Augment.

The architecture and character of Augment were directly oriented toward augmenting the capability of humans to deal with tough knowledge work and to process effectively the large volumes of information with which knowledge workers must deal. A subgoal was to support active collaboration among groups of workers. To gain experience with the issues and needs associated with this support, we developed and operated the Network Information Center (NIC) for the original ARPANET user and researcher community.

Creating a Collaborative System

The following elements are necessary ingredients in a system designed to support collaboration in a community of knowledge workers. The sequence represents an explicit progression that begins with tested techniques whose "cultural shock" and financial investment are relatively low; it proceeds through paced, open-ended evolution with time, experience, and perceived payoff toward tools and techniques that involve a greater investment in both financial and social areas.

• *Collaborative dialogue.* Computer tools for the composition of messages and for their subsequent reviewing, cross-referencing, modification, transmission, storage, indexing, and full-text retrieval are a necessary part of a CSCW system. A "message" in such a system can be of any length. It can contain formalized citations pointing to specific passages in prior messages, so that a group of related messages becomes a network of recorded-dialogue contributions.

There should also be automatic message delivery; full cataloging and indexing; on-line accessibility both to message notification and to the full text of all messages; and open-ended storage of the dialogue records. These services enable a community of people who are distributed in space and time to maintain effective, recorded, collaborative dialogue in a manner that qualitatively differs from most ordinary electronic-mail systems.

With Augment, real-time remote dialogue (teleconferencing) was supported by a "shared screen" facility through which users could "link up" their displays; each party to the link sees a common display view. Any party to the link is able to point to or control or execute

continued

The NLS/Augment Architecture

The On-Line System, or NLS, was designed to support members working in varied disciplines, including software engineers, managers, and social scientists. There were core tools used by all these knowledge workers, as well as specialized tools developed for particular requirements. All the tools shared the commonality of design principles that we thought essential to the success of what we termed a knowledge workshop. Early development began in 1963 and proceeded until 1976. (See photo A.)

The physical environment on which Augmentation Research Center (ARC) members (and collaborators across the country) worked evolved along with our system and externally available technologies. Back when the project started, display technologies were extremely primitive: Most people were still using punched cards and paper tape. Few computer users had direct access to a computer.

A Revolutionary Console

In that context, the NLS terminals were especially revolutionary. The display consoles were equipped with typewriter-like keyboards, a five-finger keyset for one-handed character input, and a mouse, invented in our lab, for cursor control (see photo B).

The keyset was useful for most members of ARC, as commands were generally recognizable by single-character mnemonics, with appropriate feedback provided by the system. Most team members became proficient at one-hand text input, leaving the other hand available for cursor control by means of the mouse as they moved through the information space on their terminal screens.

Initially, screens were generated on small CRTs in our machine room and transmitted via closed-circuit television to the ARC workstations. Later on, as character-based displays became commercially available, we created external boxes to those terminals for attaching mice and keysets and controlling the cursor and screen updates in the manner required by our essentially nonlinear system devices, which were developed principally as "glass teletypewriters."

Those boxes, or line processors, were eventually made available to users over the ARPANET so they could experience the display-based version of NLS. However, because of the initially limited availability of displays, we also created a typewriter version of the system (TNLS), which had a complete mapping of the display NLS (DNLS) interface and permitted ready access to information across the country through the then more cost-effective typewriter terminals.

NLS was the core workshop software application system. It centered around the composition, modification, and study of structured textual material. Graphics were available in a primitive manner on the early terminals; the later line-processor-based systems made graphics available on additional, external graphics displays.

The type of bit-mapped graphics systems and hard-copy printers readily available today were not available to us at the time, although later evolutions of our file-system content architecture could accommodate graphical entities as data nodes. Moreover, there were important areas associated with the text domain that needed exploration.

A Hierarchical Structure

The underlying NLS document architecture was hierarchically structured; the structure of a file was separated from its content. Originally, content nodes were strictly textual in nature; eventually, each structural node referred to a property list of content nodes of varying types, including other hierarchies (i.e., text, graphics, code, and so on).

The structure made for rapid navigation through the information space created by a file or collection of files. Its complexity was hidden from novice users (who didn't need to know about its implementation and, in fact, could ignore the hierarchy if they wished as they created linear documents in the NLS editor).

However, more sophisticated users

continued

Photo A: *A 1967 augmented meeting. This configuration is similar to more current sustems, such as Xerox PARC's CoLab.*

could address any point in any file throughout a network via a link—a syntactic address, which could be embedded anywhere in other files. These links were essential to the first implementation of the sort of system later called hypertext by Ted Nelson. (See the October BYTE.)

The basic node in an NLS file was a statement, most often used to represent a paragraph in text, a line of code in a program. The user could impose filters on the content or structure through tools either built into the system (view specifications) or installed through a user-programming facility. Thus, users could look at a particular number of lines of those statements at a particular level in a file. This facility was similar to those in so-called idea processors, such as Living Videotext's More. Associated with each statement was the date and time of its last edit as well as the identifier of the community member who created or edited it. Document filters over authors and time could also be installed.

Because of the collaborative nature of the development of NLS, there were tools and conventions for group authorship. Only one person could have write-access to a file at a time. Other team members could have read access to the file, minus the edits currently being made. A lock was placed on a file being written; if another team member accessed the file or attempted to write on it, that person would be told who had the file locked.

A Variety of Tools

NLS had tools for moving through the information space, using the mouse to select locations on the screen or the addressing capability (using the link syntax) to specify locations not directly accessible from the screen. You could jump to locations related to structural entities (successor, predecessor, and so forth), or you could jump via links by pointing to a textual link in a file or typing one in when prompted. Users could have up to eight windows on a screen with different files or different parts of the same files visible. Material could be copied across windows.

Programmers had access to a number of languages we created: Tree Meta, a compiler-compiler, was used to bootstrap us onto different machines (XDS 940, PDP-10, PDP-11, and DEC 20) and to create the other compilers and assemblers we used. L10 was a block-structured language with pattern-matching and string-construction facilities. The same pattern-matching syntax was used by less sophisticated users to generate filters in the core workshop. The Command Meta Language (CML) was used to create user interfaces that were independent of terminal type (display or typewriter) and individual user preferences. CML grammars were interpreted. Contextual entries into syntactic and semantic help systems were generated from the CML grammars. The Output Processor interpreted a comprehensive document-formatting language.

Programmers could look at procedures on the display and, encountering a reference to another procedure, jump to it. If it was not within the currently open file, the jump took place indirectly through a procedure catalog automatically generated by the automated program librarian.

The program librarian operated over system databases at night (or whenever it was invoked). If a code file had been modified, it would be automatically compiled; if all compilations took place without error (errors were recorded in other NLS files), a new system would be linked and created. The catalog was sorted alphabetically and, in addition to links to the files containing the procedures, included comments and calling sequences that were extracted from the procedure.

Programmers could view and modify procedures, compile them independently into their own address spaces, and automatically "replace" the existing versions of the procedures in the system to try out variations. Users could install (automatically when entering the system) alternative versions of standard system procedures. A symbolic debugger could be called up in a separate window, and breakpoints could be set by pointing at procedure names in the source-code file with the mouse.

We had tools for creating recorded dialogues with other users: Our Journal provided the usual message-passing facilities available on other timesharing and networked systems. However, we

Photo B: *A display NLS workstation with video overlay. Note the chord keyset input device used as a supplement to the keyboard. (The mouse may be seen in the video overlay on the screen.)*

could also submit larger documents or parts of them for permanent storage and retrieval or for the information and collaboration of other users. Shorter messages could be transmitted directly to a user's Initial File (the file seen on entering the system, similar to the desktop on current systems). Citations to larger documents would be delivered.

On seeing one of those citations, which included links to the document's location in the Journal, a user could jump to that document. The documents in the journal were permanent, read-only records of the dialogue within the community. Links to these documents were created, and evolving commentary on the design and implementation issues were always available. These facilities are similar to those currently advocated as "hypertext publishing systems."

NLS also had tools for interactive real-time collaboration. For example, users could link their terminals together and share screens; this made it possible for them to view the same material and collaboratively edit it.

As the ARPANET became available, we were among its first users. We found it necessary to tune the network to the then unique characteristics of our highly interactive system. It was also useful to separate the architecture of the system into a front end (which handled the user-interface interactions) and a back end (which handled the execution of commands).

The front end could operate on a separate machine and communicate with back-end resources through a network. Commonly used resources could be resident on the front-end machine; resources that were most usefully shared would reside on the back end.

We also created the Network Information Center (NIC) at the Stanford Research Institute to serve as an information resource for the emerging ARPANET. We used our tools to create the ARPANET Resource Directory, which was made available in both online and hard-copy form.

NLS included facilities for document development, production (including early computer phototypesetting facilities), and control. These facilities incorporated tools for successive refinement and editing by teams of writers, editors, and reviewers and were built on other parts of the core workshop, such as the editor, Journal, and programming tools.

any of the capabilities of the workshop. Such capabilities assume a high degree of responsiveness and bandwidth in the communication channel in order to support the high degree of interactivity in the system. (Our developments in this area required extensive tuning of the original ARPANET algorithms.)

• *Document development, production, and control.* This system capability includes tools for composing, studying, and modifying document drafts and for high-quality photocomposition. In addition to the page-layout tools that have become widely available, Augment offered tools for collaboration between several authors and editors in the process of evolving a final draft. These included tools for controlling changes, new version distribution, and automatic index generation for complex documents or sets of documents.

Page-layout programs such as PageMaker have entered widespread use in recent years. However, the tools for collaborative control of other aspects of a document's evolution are equally important. Augment permitted establishing superdocuments that were hypertextually linked combinations of the whole or parts of many pieces of information. This linking implies and reflects underlying meaning in ways that mere typesetting, which deals primarily with layout, cannot. While the typeset, WYSIWYG view should be available, it should not be the only way to view a document in its larger sense.

We also assume the need for tools to authenticate submissions and comments, provide administrative support to editors, offer sequential delivery and tracking for approval chains, and show automatic "ticklers" to those who do not respond to requests for comments, modifications, and approvals.

A backlinking facility within the recorded dialogue system is also necessary to handle superseding of old documents by new. Recent versions of the Augment Journal provide such a capability, permitting users to request current or older versions of an evolving document.

• *Research intelligence.* The tools within the Collaborative Dialogue Support System for cataloging and indexing internally generated items should also support managing *externally* generated items—bibliography, contact reports, clippings, notes, and so forth.

With centrally supplied (and hence uniformly available) services such as these, a community can maintain a dynamic and highly useful "intelligence" database to help it stay up-to-date on external happenings that affect it. Citations of external items from within the internally generated dialogue base, in the form of annotations, commentary, or supportive references, offer computer-sensible interlinking of the external information with the internal information and facilitate browsing, retrieval, searching, back-citation, and so on.

• *Community handbook development.* This includes extending this research service toward the coordinated handling of a very large and complex body of documentation and its associated external references. This material, when integrated into a monolithic whole, may be considered a "superdocument." Tools for the responsive development and evolution of such a superdocument by many (distributed) individuals within a discipline- or project-oriented community could lead to the maintenance of a "community handbook," a uniform, complete, consistent, up-to-date integration of the special knowledge representing the current status of the community.

The handbook would include principles, working hypotheses, practices, glossaries of special terms, standards, goals, goal status, supportive arguments, techniques, observations, how-to-do-it items, and so forth. An active community would be constantly involved in dialogue concerning the contents of its handbook. Constant updating would provide a "certified community position structure" about which the real evolutionary work would swarm; flexible tools for on-line navigation and view generation would be very important, as would the facility for generating hard-copy equivalents.

The "handbook cycle" includes the incorporation of ongoing dialogue and intelligence mediated by professional facilitation to create evolved versions of the community handbook.

• *Computer-based instruction.* We assume that the special training needs of a community of collaborating knowledge workers will be supported by computer-based instructional tools. These would make use of the other knowledge workshop services described, especially dynamic filtering of the community handbook.

A "shared screen" facility is useful for instruction so novices can get access to expert users or coaches in parts of the system for which other instructional tools are inadequate and for which local teachers are unavailable. Having an expert take you along for a ride is an extremely effective learning technique.

continued

• *Meetings and conferences*. At ARC, we made extensive use of augmentation tools in our local and distributed meetings. Projected display images, video overlays, and split-screen image superimposition were first used to great effect by Engelbart in the 1968 IFIP Fall Joint Computer Conference in San Francisco.

Dynamic control of the agenda and the collaborative creation of position papers are some typical uses of these services.

• *Community management and organization*. Conventional project-management operations can be augmented through the use of computer-based project-management tools with the enriching services of dialogue support, document development, and the handbook, which would include plans, commitments, schedules, and specifications.

• *Special knowledge work by individuals and teams*. The tools supporting a collaborating community should be available to the team members in their roles as individuals and members of other teams. A user-programming facility in Augment made it possible for individual users to customize parts of the system according to their needs and abilities. Some of these specialized extensions became part of the more widely available tools for the entire workshop community.

A Formula for Success

As Augment evolved, we realized some assumptions that we think are applicable to any successful CSCW system:

• *Coordinated set of user-interface principles*. There should be a common set of principles over the many application areas. This does not mean that the user interface itself is necessarily the same across all domains. It does mean that a common underlying style of communication is present. While each domain within the core workshop area or specialized application system may have a vocabulary unique to its area, this vocabulary should be used within language and control structures common throughout the tool environment. Users learn new functions by increasing vocabularies, not by learning separate "foreign" languages. When in trouble, they will invoke help or tutorial functions in a standard way.

This point has become apparent in the Apple Macintosh environment. Users of different applications have a common method of interacting with each application. This makes it easier to learn new applications and to move between systems.

A single interface metaphor is neither required nor ideal. Interaction styles suitable for a particular application domain and user group may differ from those for other domains and users. Apple's HyperCard provides an example of an environment that offers interaction metaphors different from the original Apple Desktop with minimal confusion to users.

• *Grades of user proficiency*. Users who are not experienced in using the system are part of the community; they will want to be able to get at least a few straightforward things done with a minimum of learning. Even an expert user in certain domains of the collaborative workshop environment will be a novice in less frequently used domains. Attention to novice-oriented "easy to use" features is required.

However, users should be rewarded for their increasing proficiency with a rich tool environment that offers advanced vocabularies and the opportunity

continued

for individual customization in every specialized domain.

• *Ease of communication among, and addition of, workshop domains.* We think that there will be many different parts of an augmented-knowledge workshop, each with its own tools. You should never be bound to isolated areas of the workshop. It should be possible to move and communicate information between domains easily. It should also be possible to install new tools as needed.

• *User-programming capability.* Users must be able, with various levels of ease, to add or interface new tools and extend the language to meet their needs. They should be able to do this in a variety of programming languages in which they may have training, or in the basic user-level language of the workshop itself (e.g., through a macro facility.)

We can't ignore the social implications of our technical progress.

• *People-support services.* The computer-based tools will be insufficient by themselves. The CSCW technologies will create opportunities and needs for highly specialized professional services, such as database design and administration, training, cataloging, and retrieval formulation.

• *Recognition of standards for information interchange and ranges of hardware.* We should not have to assume the presence of a particular type of machine in a user's work environment. It should be possible to exchange information and get a reasonable representation of the information shared across system environments.

• *Careful development of methodologies.* The elements involved in augmenting communities of knowledge workers include the development of both "tool systems" and "human systems" (the set of skills, methods, languages, customs, procedures, training, and organization structures needed for effective use of tools). New technologies, even those such as CSCW that aim at improving group interaction, contribute directly only to the tool system. The cultural evolution that led to the current state of the human system took place with a very primitive tool system.

As much care and attention needs to be paid to developing the procedures and methodologies associated with the people-support services and the organizational and societal effects of introducing new technologies as is spent on developing the technologies themselves.

• *Co-evolution of roles and organizational structures and technologies.* The widespread availability of successful CSCW services will create the need for new organizational structures and roles. These structures and roles need to co-evolve with the technologies. For example, we found there was a need for what we called knowledge-workshop architects who served as "change agents" in introducing new technologies into their organizations.

To take advantage of the radical, emerging tool-system inventions associated with CSCW, it is inevitable that the evolution of the human system will begin to accelerate. The optimum design for either a tool system or a human system is dependent on the match it must make with the other. The high degree of mutual dependence implies that a balanced co-evolution of both is necessary. The bind we are in is that our society encourages and rewards progress in the technological and material sense and often ignores the human and social implications of that progress. ■

FURTHER READING

Ambron, Sueann, and Kristina Hooper. *Interactive Multimedia*. Redmond, WA: Microsoft Press, 1988.

Greif, Irene. *Computer-Supported Cooperative Work: A Book of Readings*. San Mateo, CA: Morgan Kaufman Publishers, 1988.

Johansen, Robert. *Groupware: Computer Support for Business Teams*. New York: Free Press, 1988.

Grimsdale, R. L., and F.F. Kuo, eds. *Computer Communication Networks*. Leydon: Noordhof, 1975.

Goldberg, Adele, ed. *A History of Personal Workstations*. New York: ACM Press, 1988.

Rheingold, Howard. *Tools for Thought*. New York: Simon and Schuster, 1987.

Douglas Engelbart, a senior scientist at McDonnell Douglas, recently created the Bootstrap Institute to further CSCW research. Harvey Lehtman is manager of the New Media Group at Apple Computer. They can be reached on BIX c/o "editors."