Performance Improvement Quarterly, 8(1) pp.47-93

Attributes and Behaviors of Performance-Centered Systems

Gloria Gery
Gery Associates

ABSTRACT

Work is becoming increasingly computer-mediated. Performance development is becoming increasingly difficult and expensive due to complexity, change, restructuring, and redefinition of who does work. In the consumer marketplace, software is emerging with characteristics and behaviors that actively support performance development while permitting learning. These performance-centered systems are the next step in the evolution of performance support. Performance Technologists must and will play an active part in both their advocacy and design. Ultimately, no consumer or large-scale software system should be considered acceptable until *day-one performance* is generated for novice performers while still accommodating the requirements of expert performers.

Part I
The Emergence of Performance-Centered Systems

The Progression Toward Computer-Mediated Work

It appears we are on an inexorable path toward computer-mediated work. Digital displays are in every device and on every desk top, supporting every type of task. While some workers have not been touched by the digital revolution, they are few in number. And while some work will always be performed primarily without electronic tools or support, most work includes some tasks that require direct interaction with technology. Increasingly, software will be task focused. Support for both task performance and learning will be come an integral part of the technological environment characterized by computer-mediated work.

The effectiveness of computer-mediated work environments on both business and individual performance will be a direct function of three things: 1) the assumptions, design philosophies, and skills of developers, 2) the expectations and needs of software buyers and consumers, and 3) the real and perceived accountabilities and consequences to software developers.

Article Objectives

In this article, I will examine characteristics, attributes, and behaviors of performance-centered software packages that are emerging in the consumer software marketplace, and compare and contrast them with the large-scale systems software being designed by internal Information Systems staffs and vendors of large-

scale software designed for financial, manufacturing, processing and administrative systems. I will examine the different underlying assumptions and accountabilities that operate within the two environments. The consequences of the two types of systems on human and organizational performance will be surfaced. And because of the considerably different performance outcomes associated with consumer vs. large-scale systems environments, I will advocate that Performance Technologists play an increasing role in software design to assure that organizations include certain requirements in their software acquisition or development specifications and incorporate necessary evaluation criteria in their performance and usability testing measurements. Ultimately, no consumer or large-scale software system will be acceptable until *day-one performance* by both novice and expert performers is generated.

Methodology: Emerging Perspectives Grounded in Observations

These initial definitions and descriptions of the emerging attributes and behaviors of performance support are at an early stage of development. Designers of performance-centered systems are creating new visions of interface design, new performance support structures, and new system functions in direct response to business and user problems and market competition. Few are guided by a set of integrated and fully articulated design principles. Many innovations are the result of individual or team creativity and iterative design employing rapid prototyping coupled with ongoing usability and performance testing. Articulation and communication of these emerging design structures and principles will be necessary to achieve wide-scale and rapid development of new and powerful software systems that accelerate individual and organizational performance.

To date, there is little or no empirical research to explore. While literature citations are available for relevant theoretical underpinnings, the existing definitions and descriptions of performance-centered design are largely a function of individual observations of large amounts of consumer products and personal participation in creative design activities. A chicken and egg observation and formulation process is occurring: I observe and describe products; I then articulate observations and influence product design with clients and software vendors. Essentially, I have synthesized what I am observing in the *consumer* marketplace because I believe this synthesis of observations portends changes that are necessary and underway in internal systems development. There are, of course, creative designs emerging in large-scale systems development, but the creative results are isolated and generally confined to more one-of-a-kind retail, entertainment, or artistic applications. Innovation is very limited in large-scale software that supports traditional financial, administrative, manufacturing, logistic, and customer service systems. Progress there is incremental and most often focused on improving interface design with new Graphical User Interface (GUI) objects—an admirable but marginal improvement in relation to the overall performance development need.

Table 1
What Drives Software Development? A Comparison of Large-Scale and Consumer Software Development

Points of Comparison	Large-Scale Systems	Consumer Software
Assumptions about users' and workplace knowledge:	• Users will know the work the software will be part of or support. • Knowledgeable people will be available to users. • Users will attend training prior to using software.	• Users will know limited interface conventions (e.g., use of buttons). • Users will not know the content or task domain. • Knowledgeable people will not be available to users.
Development priorities:	• Bug-free code • High integrity data • Machine performance • Matched to contracted client specifications (i.e., the client who pays the development or acquisition bills) • Architectural compatibility • Operational performance • On-time delivery • Delivery within budget • System maintainability	• Market acceptability • Product reviews by; press and users • Impact on user performance • Low implementation, training requirements • Negligible ongoing support overhead • Time to market • Executable on installed hardware base or demonstrates such value that people will upgrade hardware to run software
Implementation time frames and expectations:	Short to moderate for initial implementation. Gradual utilization over time.	Immediate implementation and immediate performance outcomes.
Assumed user characteristics:	• Compliant to management directives • Captive. Cannot reject the software for an alternative • Used to difficult systems environments • Prefer on-time availability to usability and performance impact • Data driven • Willing and able to invest in learning • Not influential in the marketplace • Stable over time	• Working against deadlines • Impatient • Results oriented • Can reject software for marketplace alternatives or can return to non-automated task performance • Influential in the marketplace • Software users will change over time; high user turnover and/or new user populations emerging
Design Goals:	• Conform to standards (e.g., GUI) • Reflect current work process • Similar to current work environment requiring only incremental change	• *Killer* application with unique attributes and behavior • Fundamentally alters how work is done • Day-one performance by new users • Seductive to users • Create energy about working with the software
Measurements and rewards based on:	• On-time delivery • Development costs • Functionality • Technological superiority	• Consumer (i.e., performer) acceptability and mind share • Impact on efficiency, effectiveness, value-added or business strategy • Profitability
Not accountable for:	• Costs of implementation • Impact on results or business strategy • *User* satisfaction	• Marketplace makes the vendor accountable for all consequences through its purchasing power and alternatives

Consumer Software Development: Market Drivers Require Creative Solutions

The consumer-software market has been subject to different accountabilities and consequences than the market for software for large-scale organizations. Consumer product vendors have immediate and critical impacts when their products fail the market acceptance test: inadequate gain in (or actual loss of) market share, eroding profits due to escalating customer service requests, and customer dissatisfaction. Successful vendors are developing software that people can use to become productive immediately without training and with limited external documentation and human support. For both consumers and vendors, software priced at $100 should not require 45-minute phone conversations with the support desk about task procedures or problem resolution. Even if vendors charge for the support, consumers will not pay more for the support than for the product itself—nor will they wait on hold for 45 minutes for an answer to a problem limiting their immediate work performance. Some consumers have bought, tried, and discarded more software than they have kept. When the effort associated with task completion requires more than the time available at the moment of need, users decrease their goal, work around the limitations of the software to find some way to accomplish the task and, ultimately, reject the software itself. Word about supportive vs. hostile software spreads through the marketplace like wildfire. Table 1 compares consumer and commercial software drivers and accountabilities.

Design Responses by Consumer Software Developers

Consumer software developers are responding to the high overhead of support costs by developing creative structures such as Wizards and Cue Cards to support procedural and non-procedural tasks and by incorporating task- or performance-centered design into their interfaces. One explicit design goal is to radically decrease the performance development curve in relationship to that required by the competition and higher overhead designs. The goal of these consumer software developers is to make the software environment inherently or *intrinsically* supportive of task performance and deliverable creation.

Vendors are increasingly employing metaphors and visual design to capitalize on performer prior knowledge and experience. They are building support for performance development directly into the software interfaces and behavior.

Lag in Large-Scale Software Systems Design

Currently, large-scale software systems developed by either vendors or internal IS staff do not reflect the same focus and creative output that is found in consumer products. The reasons are many, including: 1) our cultural heritage of a data-driven (versus process-driven) world; 2) limited experience with graphic, visual, windowed and metaphorical interface structures; 3) low expectations and lack of explicit requests from software project sponsors or software buyers; 4) limited experience with new software development tools and languages such as object-oriented products; 5) organizational measure-

ments focusing on traditional outputs; 6) terror at the requirement for adding requirements that will further delay complex projects that are typically underestimated, late and over cost; 7) lack of skills; 8) lack of understanding of the complete work context; and 9) vested interest in the status quo.

Performance Technologists must learn from consumer software and develop expertise in building intrinsic support into large-scale applications to break the logjam of inevitable overhead and the long performance development cycle times associated with current applications software implementation. Performance Technologists must and will be the advocates for greatly expanding the definitions of performance-centered design and for the implementation of performance systems. Success will be a function of 1) creating dissatisfaction with current software designs and products and surfacing understanding of the tremendous negative impact on implementation, training, and support costs and on business' bottom line; 2) defining, describing, and illustrating the attributes and behavior of performance-centered software so explicit expectations can be communicated; and 3) obtaining and maintaining the required and adequate political, logistical, and economic sponsorship for change.

Types of Performance Support Within Computer-Mediated Work Environments

To effect change, the nature of the change and the desired end state must be clearly understood. Understanding the types and nature of performance support possible within software is fundamental. There are three fundamental types of performance support within computer-mediated work environments:

1) Intrinsic Support is performance support that is inherent to the system itself. This support is so integrated into the interface structure, content, and behavior and the application logic that it is impossible to differentiate it from the system itself. In fact, the attributes and behaviors of the interface and immediately accessible resources, combined with the underlying data and system functionality, make it a performance-centered system. Intrinsic support is most in evidence in task-centered systems where explicit understanding of the work and tasks, performers, and work contexts can be institutionalized into the software. When there are high levels of intrinsic support as evidenced by a large number of the attributes and behaviors described below, people do not have the psychological awareness of being *in software.* They simply feel that they are just *doing their work.*

2) Extrinsic Support is performance support that is *integrated* with the system, but is not the primary workspace. Extrinsic support is computer mediated and often context sensitive to the task and worker situation, but it is either a) invoked by the performer (e.g. Advisors, Wizards or Cue Cards) or b) is presented to the performer and can be accepted or rejected by turning it on or off (e.g., Tips, Cue Cards, Explanations, Checkers). Table 2 offers a high-level summary of various extrinsic support structures and their uses.

3) External Support is support for performance and learning that is *external to* and not integrated with the computer-mediated workspace.

Table 2
Extrinsic Support Structures Presented To or Invoked By Performers While Working

Structure & Behavior	Uses	Objective
Cue Cards • Provide Task Guidance • Sequence content by performer choice or underlying logic	Linear sequential or branched tasks: • Procedures • Troubleshooting or diagnosis	• Task completion • Learning while doing
Explanations or Demonstrations • Presentation sequence • Interactive or not	• Orientation • Understanding • Skill development	• Learning
Wizards, Assistants or Helpers • Present options or choices • Accept text, numeric, graphical or voice data • Progress user through task • Preview consequences • Summarize choices or conditions • Produce output or execute task • Transform data • Active • Change data, views, screens or states	• Task guidance • Task execution	• Task completion • Learning by modeling
Coaches or Guides • Structure interactive walk-through of system-related procedural tasks • Take control of system • Accept input and release control of system to user	• Active task guidance through system tasks	• Task completion • Learning while doing • Replaces human support
Searchable Reference • Content or knowledge database on concepts, products, processes, equipment, facts, principles, etc. • Organized for flexible search, retrieval and navigation	• Information search • Information retrieval • Browsing	• Information access • Learning
Checklist • List of items or task completion criteria Markable • Context sensitive	Quality evaluation	• Task evaluation to be completed by the system or performer
Process Map • Charts • Diagrams • Flow charts • Lists	• Process overview • Orientation to process completion (i.e.,"You are here..."	• Establish and maintain performer orientation • Learning
Examples • Database of instances	• Idea generation • Understanding	• Learning • Idea development
Templates • Pre-structured formats or shells	• Deliverable organization	• Consistent and rapid task completion
Tips • Hints • Tips • Alternatives	• Presentation of granular information at software start-up or within a context/condition	• Continuous learning • Problem solving
Practice Activities • Structured practice • "Let me try" activities	• Low-risk experience • Self-directed learning	• Skill building • Confidence building
Assessments • Timed or judged tests or practice activities	• Self assessment	• Performance evaluation • Knowledge evaluation

External support may or may not be computer mediated. What differentiates *external support* from extrinsic support is its lack of prior integration with computer displays and tasks. Performers must integrate the content and logic available from external support into their behavior at the moment of need. The appropriate content or procedure must reside within the performers' memory. Examples of external support include training programs (classroom, paper-based or CBT), documentation, job aids, peer support, Help Desks, bulletin boards, and computer conferencing environments. By its very definition, external support results in breaking the task context and the resultant performance flow.

Design Goals

The designers' goal for a performance-centered system is to integrate as much as 80% of the required performance support as intrinsic support with plus or minus 10% each in the extrinsic and external categories. The reasons for this goal include leveraging designer and performer time, decreasing overhead associated with performance development, establishing inherently maintainable work environments, and accomplishing desired impact. The consumer products described later in this ar-

Table 3
Attributes of Performance-Centered EPSS

1) Establish and maintain a work context.
2) Aid goal establishment.
3) Structure work process and progression through tasks and logic.
4) Institutionalize business strategy and best approach.
5) Contain embedded knowledge in the interface, support resources, and system logic.
6) Use metaphors, language, and direct manipulation of variables to capitalize on prior learning and physical reality.
7) Reflect natural work situations.
8) Provide alternative views of the application interface and resources.
9) Observe and advise.
10) Show evidence of work progression.
11) Provide contextual feedback.
12) Provide support resources without breaking the task context.
13) Provide layers to accommodate performer diversity.
14) Provide access to underlying logic.
15) Automate tasks.
16) Provide alternative knowledge search and navigation mechanisms.
17) Allow customization.
18) Provide obvious options, next steps, and resources.
19) Employ consistent use of visual conventions, language, visual positioning, navigation, and other system behavior.

ticle reflect very high levels of intrinsic support.

Attributes and Behaviors of Performance-Centered Systems

My recent work has involved evaluating highly effective consumer and large-scale software with an eye to identifying, labeling, and categorizing the common characteristics, activities, and behaviors that seem to quickly generate competent and confident learners. Testing the software on sample tasks, I react emotionally—then intellectually. While working with software alone or in groups—and while conducting software demonstrations to develop understanding and stimulate design discussions—I began to develop labels and other descriptive language to make communication easier. My observations of high-impact, intrinsically supportive software have resulted in a list of nineteen key attributes and behaviors.

The list in Table 3 reflects some task sequencing (e.g., first set a context, then figure out what you're going to do, and then do it in the best way). Items 5-8 describe things that appear on the display (or alternate display). Items 9-12 describe what is presented as a function of user or system action. Attributes 13-18 describe system behavior, options and underlying functionality, but they also describe what appears in the interface as well. Item 19, Consistency, is last simply because conformance to standards or conventions is the exclusive goal that many are advocating as sufficient for designing graphical environments. While consistency and conformance to standards is necessary, it is not sufficient for performance. And so I put it in last place because it is the lowest leverage attribute in terms of the performance developed.

These terms, I am certain, will evolve and take new shape over time and with critical review and input by others not so immersed in this activity of constructing descriptive language.

Initially, this list may appear overwhelming and, in some cases, attributes may appear to overlap or duplicate others (e.g., Number 8, *Provide Alternative Views Of The Application Interface and Resources* and Number 17, *Allow Customization).* Effort to further collapse these attributes, however, results in considerable omission of the subtle details that accumulate to result in powerful software designs. For the purposes of this article, I will briefly elaborate each attribute and focus mostly on creating understanding by providing examples. Because performance-centered design integrates many of these variables in a single display or set of alternatives, integrated discussion is the best. In fact, in performance-centered design resulting in heavy intrinsic performance support, the whole is greater than the sum of its parts. Much like attempts to understand the functionality and behavior of the human body, we can separate out the component systems and elements, but true understanding requires an integrated view since none of these systems or elements operates in isolation. For example, a given display can establish context, aid goal establishment by presenting options, contain embedded knowledge, structure work progress, *and* serve to automate tasks. Interface and support system design must be based on detailed performance and technology analysis to establish both requirements and possibilities. Not all tasks or performers require or would ben-

efit from every attribute. Nor do all technology platforms support all attributes or behavior. Needs must be defined in terms of tasks, outcomes, performer populations and risks. Payoffs must be established in relationship to strategy, value added, and effectiveness and efficiency considerations, including the costs associated with non-performance.

A Note on Software Illustrations

I have selected a limited number of well-designed commercial products to illustrate concepts and structures within this article. Criteria included the software's 1) immediate capability to generate performance of complex tasks; 2) diversity in types of tasks supported; 3) uniqueness in interface and support system design structure; 4) generalizability of the designs to internal software development activities supporting similar task types; and 5) affordability and widespread availability to readers wishing to purchase the software for further exploration, advocacy of these concepts, or stimulation of design ideas.

Software illustrations are of necessity restricted to individual or sequenced screen shots to illustrate an attribute or support structure. Dynamic behavior cannot be adequately captured in a static medium. In addition, space limitations require that most software screen displays illustrate multiple attributes. The displays are annotated and cross referenced to the attribute(s) they support. It is hoped that the reader will be motivated to interact directly with the referenced software.

Part II
Illustration and Elaboration of Attributes

Software Illustration of Attributes 1-4: *Performance Now!*

The screen exhibits shown in Figures 1-8 illustrate Attributes 1-4 (among others). *Performance Now!* from Knowledge Point, Inc. is one of a suite of products supporting various human resource or management activities such as constructing job descriptions and performance appraisals and developing personnel policies and procedures. Each screen exhibit is annotated to describe the specific display structures, behaviors, and outcomes and is cross-referenced to the complete list of nineteen attributes (in numeric order). It is best to look at the full sequence of displays to get the effect of the task flow.

Attribute

1 - Establish and maintain a work context.

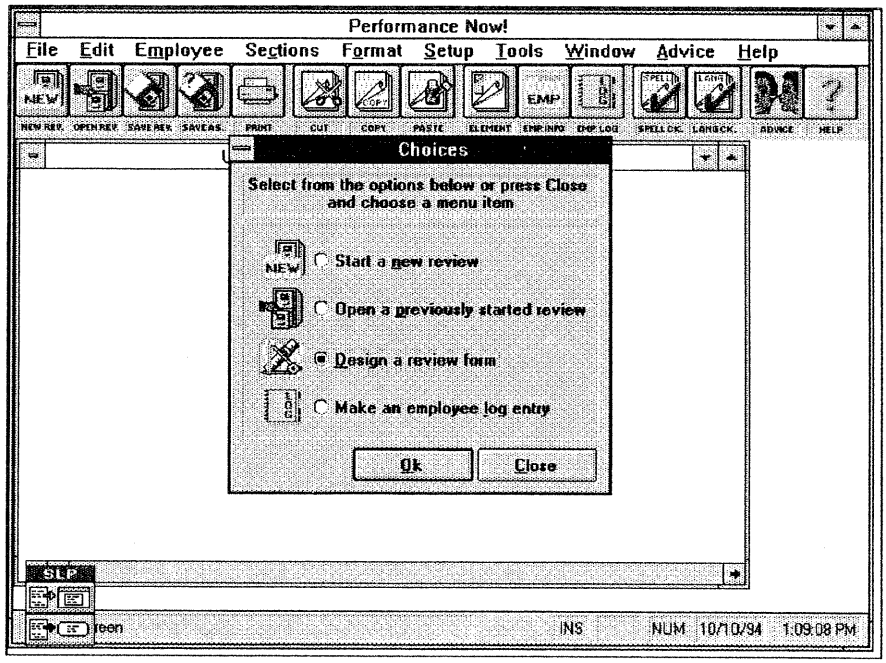
2 - Aid goal establishment.

3 - Structure work process and progression through tasks and logic.

4 - Institutionalize business strategy and best approach.

Attributes 1-4 are tightly coupled. Their primary outcome is to create intrinsic support by establishing a task-centered work space so a specific result can be accomplished. Most of the intrinsic support is provided by the primary software interface structures and associated behaviors such as stepping performers through task logic, presenting options, and so forth. For this purpose, "task" is de-

Figure 1.

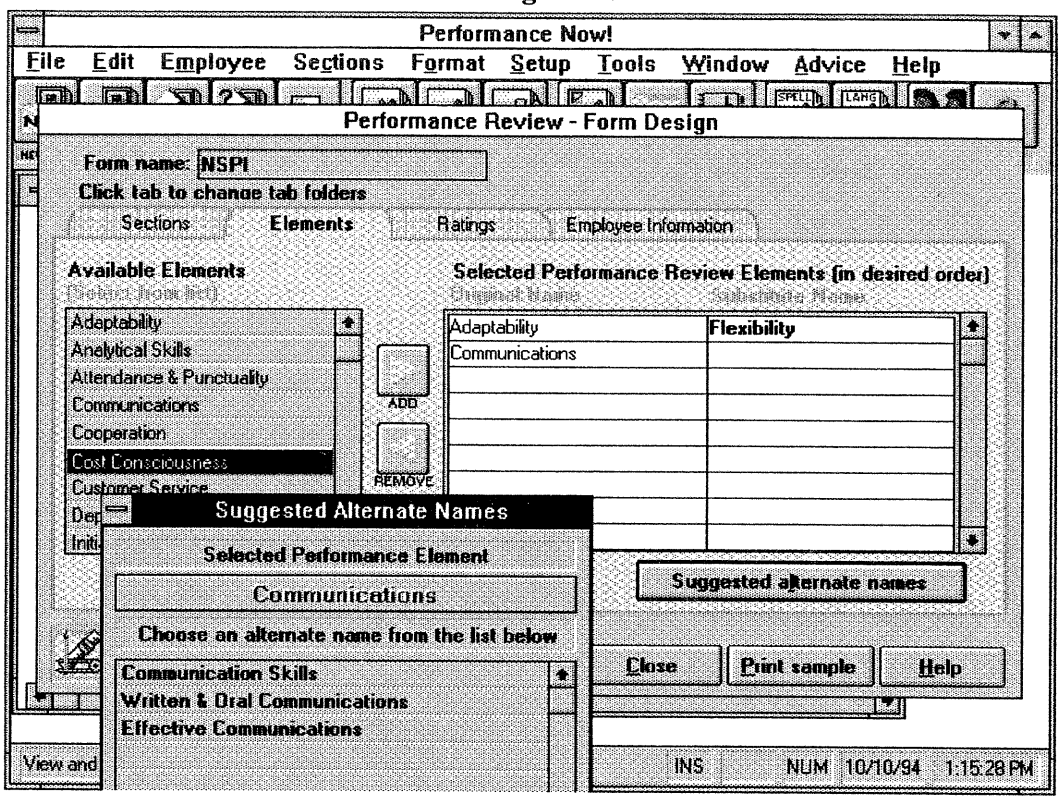


Performance Now!, a performance evaluation system by Knowledge Point, Inc. assists in creation of performance reiew forms, structuring the analytical process associated with an evaluation, and creation of a deliverable. It also provides Advice about the performance review process. In this opening screen, the work context is created and all options available within the software are presented.

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.

Figure 2.

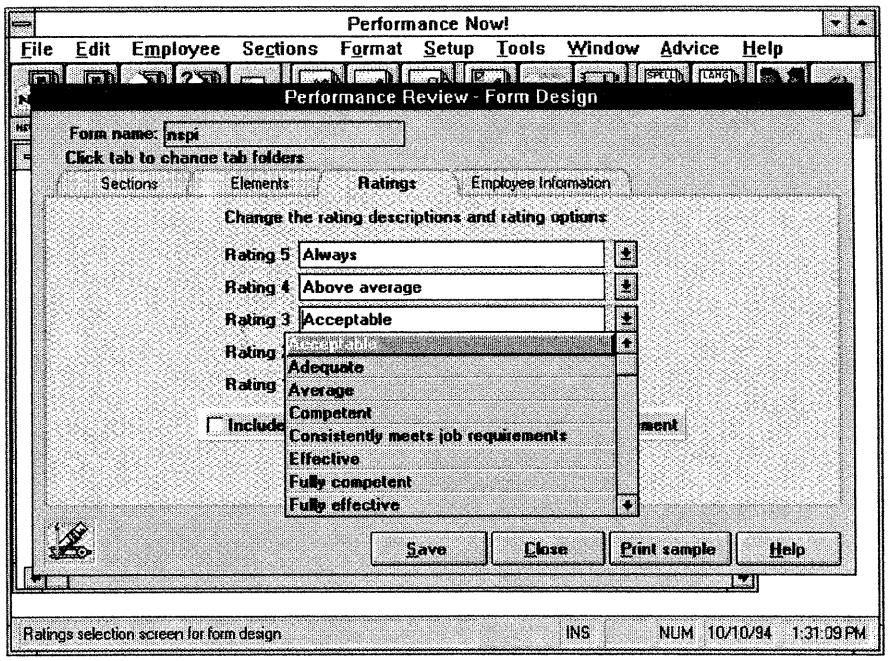


In Knowledge Point's Performance Now!, evaluation form design is accomplished in the folder environment where each section of the form is accessed by clicking on a tab. In this illustration of the Elements section, users are able to click on items to include from the options presented on the left and add then, via the Add button, to the selected elements section on the right. If alternative names to those suggested are desired, options are presented following a mouse click on the Suggested alternate names button. Both system and user-defined equivalents appear if alternates are suggested. Choices are accumulated and a customized form is automatically generated.

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources and system logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
7. Reflect natural work situations.
10. Show evidence of work progression.
15. Automate tasks.
17. Allow customization.

Figure 3.

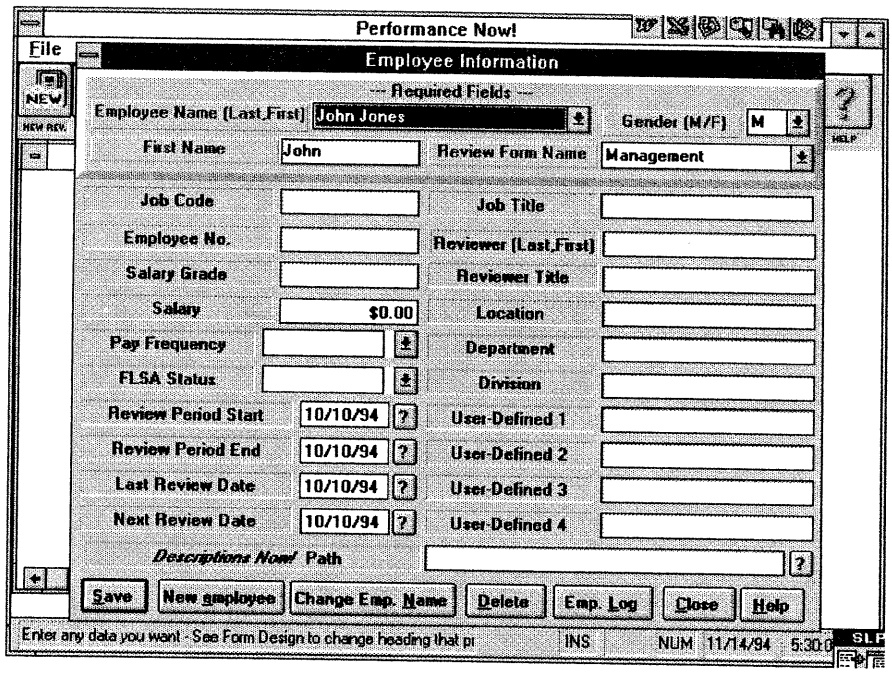


In Knowledge Point's Performance Now!, researched and professionally recommended rating schema are presented and can be selected for each rating using list boxes. These scales reflect expert evaluation on term clarity and equivalents on rating scales. Selected options are included in the automatically generated appraisal form following selection of all options. Rating scales can be included under a single Element -- or established as a default by clicking on the check box labeled "Include under all elements..." (covered up on the above display by the ratings listing).

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources and system logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
10. Show evidence of work progression.
12. Provide support resources without breaking the task context.
15. Automate tasks.
17. Allow customization.

Figure 4.

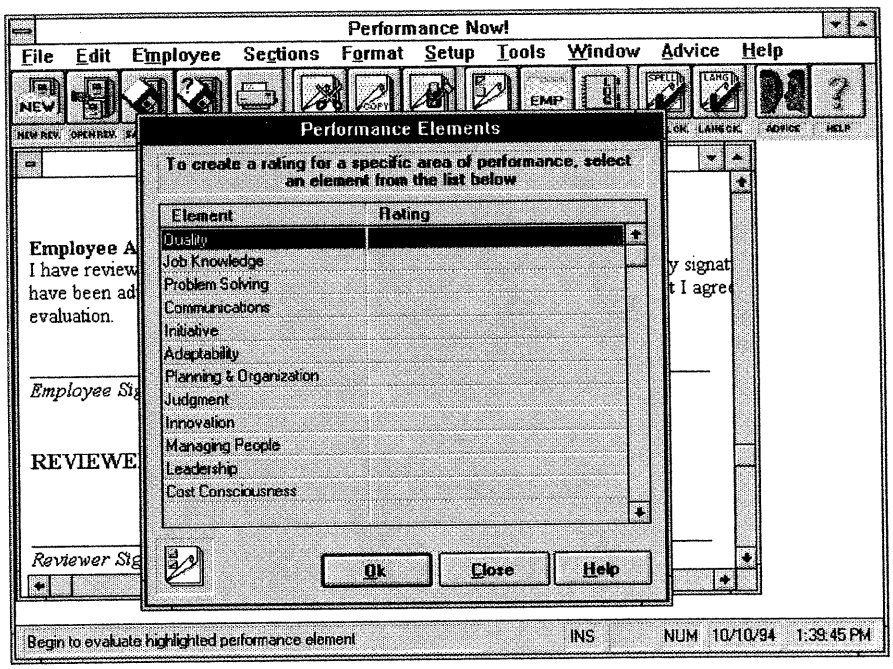


Data entry tasks are supported by forms. Data is being entered into a database that is essentially masked from the user by the more task-centered interfaces. The universe of options required or desired is presented with required fields labeled in blue (top four fields on form). When data is entered, it is checked against existing database listings to assure information is not duplicated.

Attributes Illustrated:

3. Structure work process and progression through tasks and logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
15. Automate tasks.
19. Employ consistent use of visual conventions, language, visual positioning, navigation, and other system behavior.

Figure 5.

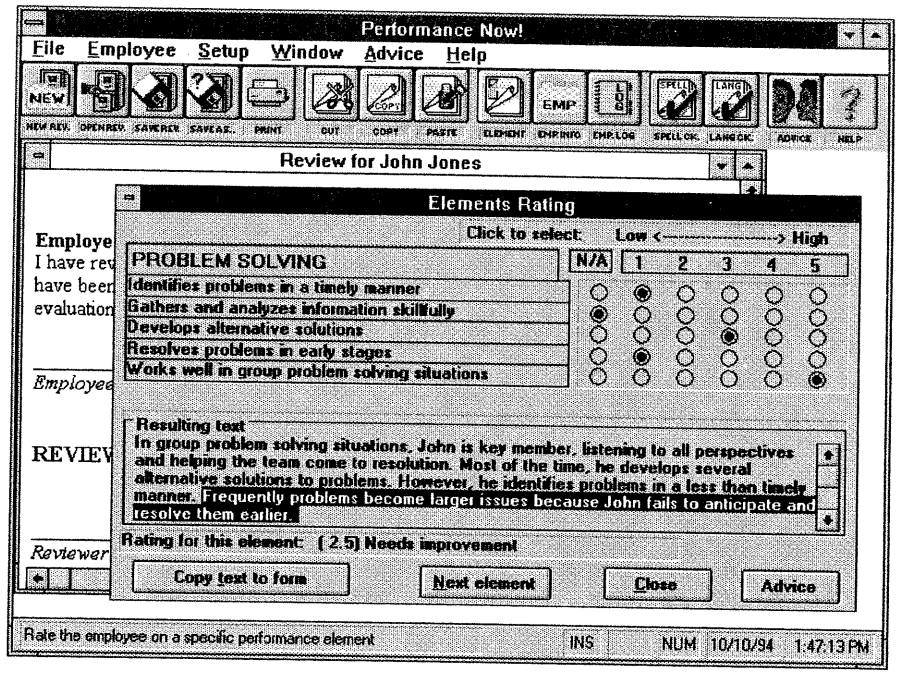


Performance Now! by Knowledge Point automates performance appraisal deliverable creation by generating a text file in a word processor that acts as a behind the scenes repository of content as it is created. The final deliverable appears under the Performance Elements window. The deliverable context is always maintained. Users are stepped through the process of selecting elements to rate. Required instructions about next steps appear on the screen.

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
7. Reflect natural work situations.
15. Automate tasks.
18. Provide obvious options, next steps, and resources.

Figure 6.

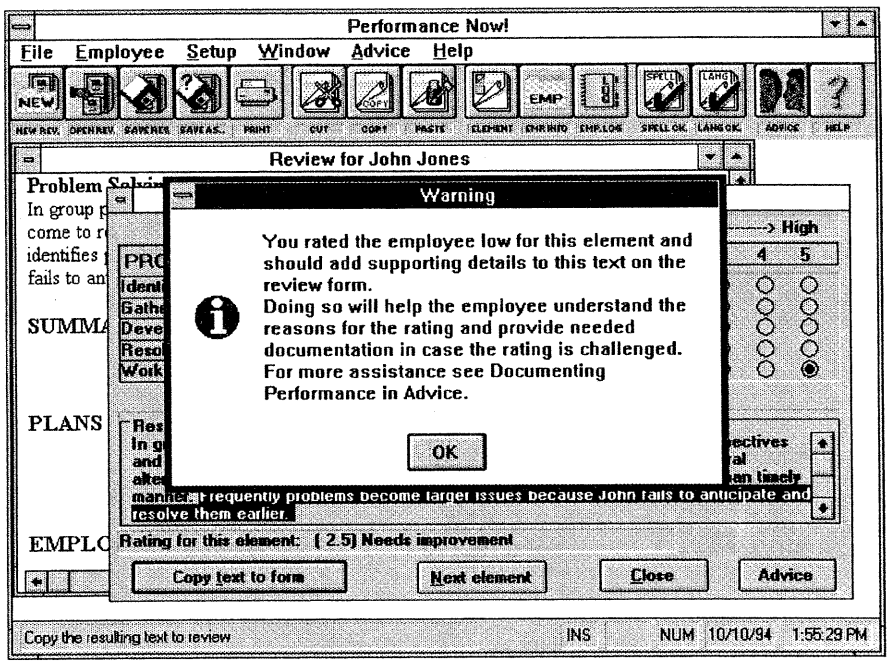


Performers are assisted in evaluating behavior associated with a given evaluation element by being presented with a rating scale. Selections are made via clicking on radio buttons. Language that will appear on the written appraisal appears below the rating scale. Data on employee name and sex are automatically generated in the narrative. "Best practice" rules are built into narrative creation. For example, more favorable information is reflect first with least favorable information last. Users can edit the standard phrases using traditional selection, replacement, cut, paste and add functions from the word processor. Each rating is weighted based on human resource research and institutionalized in the software. Ratings are automatically generated and displayed on the form.

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources, and system logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
7. Reflect natural work situations.
9. Observe and advise.
10. Show evidence of work progression,
15. Automate tasks.

Figure 7.

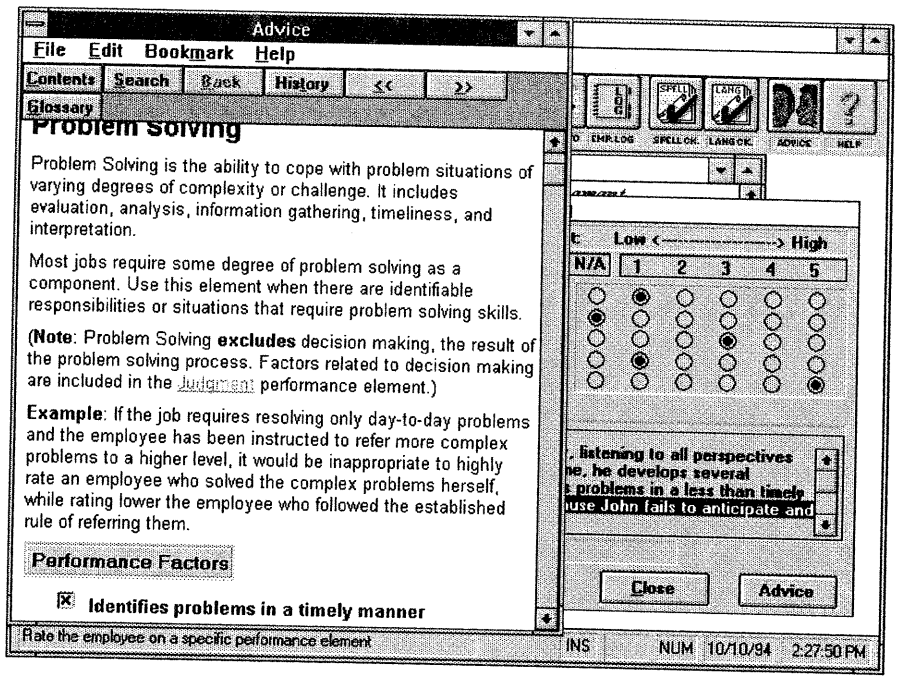


In Performance Now! by Knowledge Point, users are advised to begin the process of documentation when individuals are evaluated as demonstrating inadequate behavior. Human Resource expertise is built into the system and conditionally presented when certain conditions (such as a low rating on an Element) exist. Designers might have added a button link to the documentation environment to increase probabilities of users actually completing the recommended documentation. In addition, hyper-links to context-sensitive advice would increase the likelihood of access. While the textual cross-reference is better than nothing, hard-links permitting easy access would be significantly better.

Attributes Illustrated:

2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources and system logic.
9. Observe and advise.
11. Provide contextual feedback.
12. Provide support resources without breaking the task context.
18. Provide obvious options, next steps, and resources.

Figure 8.



Conceptual information, examples and advice are available to performers using Knowledge Point's Performance Now! Context-sensitive information is provided when Advice is accessed from within a given screen. Accessing Advice through the Advice button provides a hierarchically organized body of information that is structured in Windows Help and employs the WinHelp navigation and hypertext capabilities. Activating the Help button generates procedural and conceptual information about interacting with the software as distinct from conceptual information about the task.

Attributes Illustrated:

5. Contain embedded knowledge in the interface, support resources and system logic.
8. Provide alternative views of the application interface and resources.
9. Observe and advise.
12. Provide support resources without breaking the task context.
16. Provide alternative knowledge search and navigation mechanisms.

fined as the set of sequential, simultaneous, and/or conditional actions necessary to accomplish work in a given context by using metaphors, dialogs, forms, and other interface conventions.

While performers may have a general idea of what must be accomplished, (e.g., troubleshooting an equipment problem, responding to a customer's request, constructing a form), they may not be aware of variables (e.g., installation options, customer account information, data elements) that will affect the specific goals established or the proper sequencing of activities associated with task completion. In traditional data-driven systems, forms are presented or users communicate with a system by structuring a sequence of commands and entering them in proper sequence. Performers must know the task and related concepts or requirements in order to interact with the software. In performance-centered design, it is not assumed that performers are competent in the task.

The interface sets forth goal options and then incorporates "stimuli" guiding the performer's progress through a task. Goals can include such things as resolution of troubleshooting problems, design and construction of a deliverable such as a performance appraisal or brochure, or creation of a response to a customer inquiry.

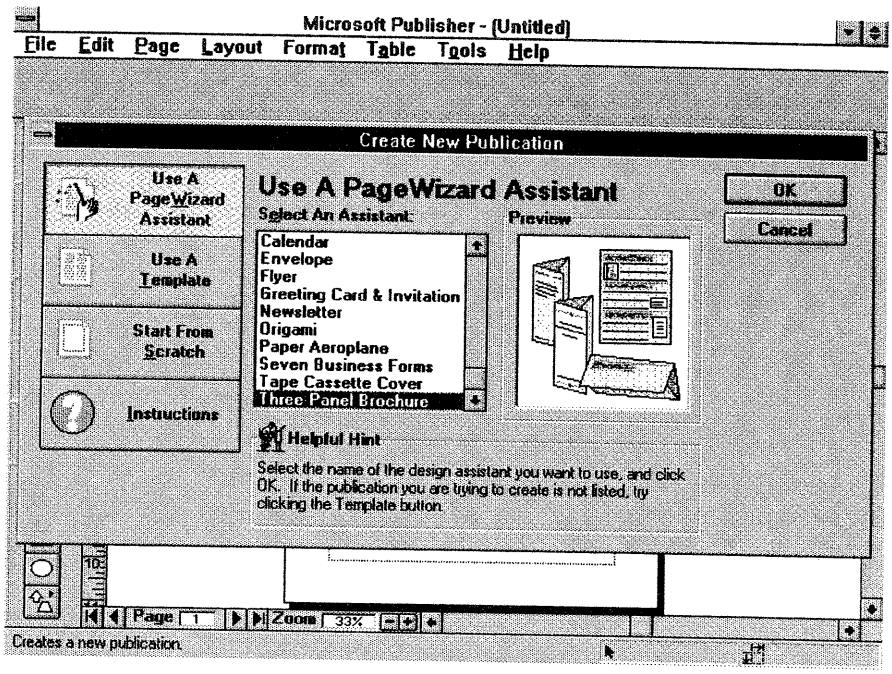
The software supports goal determination and progress toward the goal. Techniques include using dialog and conversational interviews, menus, and "tools" such as "Wizards" to summarize context or state specific options to be elected by performers. Accumulated performer inputs and selections invoke rules determining next steps. Task logic resides in the system and is apparent by what happens within the interface. Note that the system structures the task, not the performer. Performer decisions are the primary means by which the process is structured. As we'll see in the examples and discussion of other attributes, resources to permit evaluation and understanding of options, informed progress, and collateral learning are available.

Two levels of task logic are incorporated: 1) the visual representation of task progression apparent on the interface; and 2) the logic underlying the interface which interacts with data or performer inputs, senses conditions or states, executes commands, or programs or performs arithmetic, rule-based, or conditional activities. The underlying logic may be as simple as sensing mouse position, inputs, and conditions and invoking a related message box (e.g., the need to document poor performance). For other types of tasks, such as configuring complex hardware environments, the underlying logic can be complex. Sophisticated data comparisons and rule- or case-based reasoning might be required to diagnose complex conditions, configure recommendations or progress the performer to another point in the work process.

Software Illustration of Attributes 1-10: *Microsoft Publisher*

Microsoft Publisher is a desktop publishing system that incorporates most, if not all, of the attributes of performance-centered design. Inherent to any performance-centered system is the concept of task structuring. Because of the richness of the task

Figure 9.

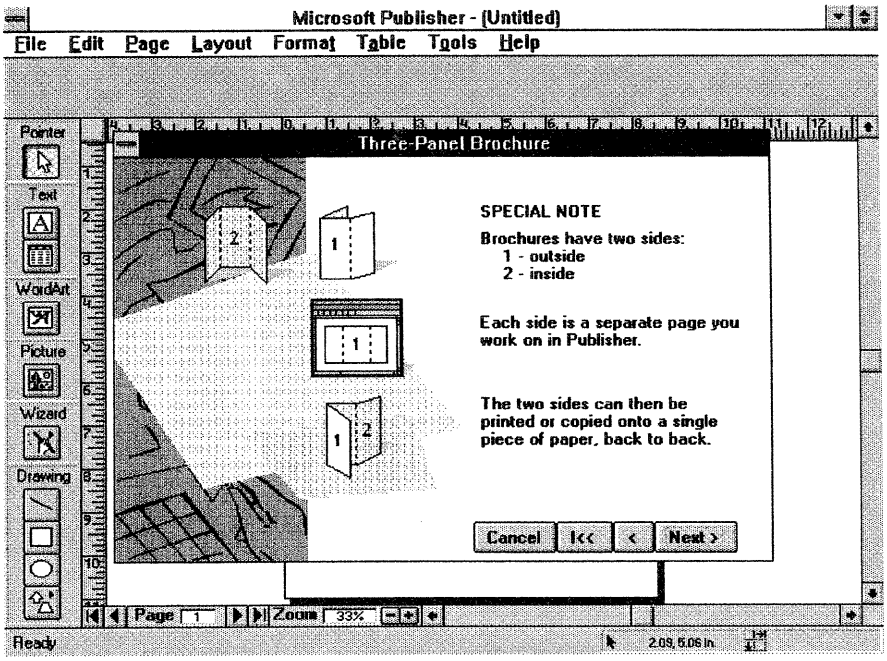


In the Microsoft Publisher desktop publishing system, users are presented with three distinct ways of interacting with the software: 1) Page Wizard Design Assistants which structure and automate the deliverable creation process from among an array of predefined options which can be mixed and matched; 2) Templates which offer predefined deliverables to be populated by the user; and 3) the Blank Page which offers all design possibilities and permits the user complete freedom in structuring their deliverable and progressing through the process.

Attributes Illustrated:

1. Establish and maintain a work context.
2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources and system logic.
8. Provide alternative views of the application interface and resources.
13. Provide layers to accommodate performer diversity.

Figure 10.

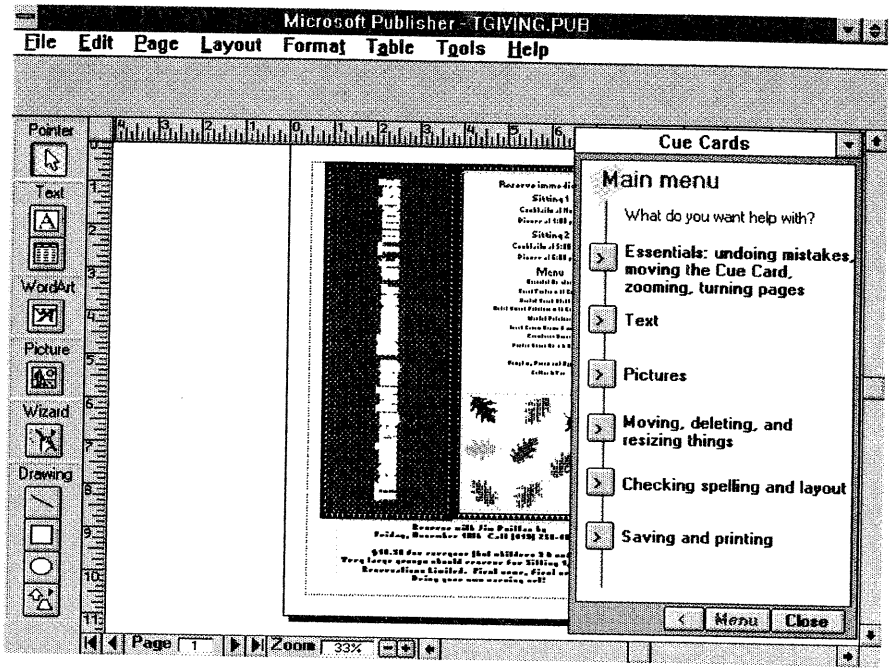


Before proceeding with deliverable design, Microsoft Publisher presents conceptual information about brochure creation. The content is embedded in the interface and presented to the performer to assure understanding of the process. In this software, there is no option to turn off such information screens, regardless of whether or not it has been internalized by the performer. In some software, the "Hide information screens" option permits a collapsed view of the interface.

Attributes Illustrated:

5. Contain embedded knowledge in the interface, support resources, and system logic.
12. Provide support resources without breaking the task context.

Figure 11.

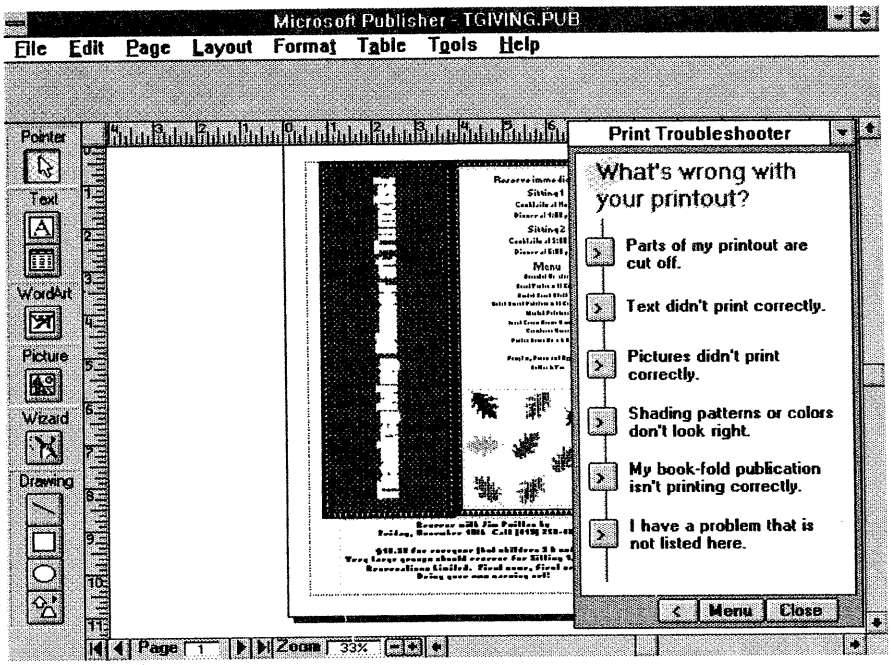


Cue Cards are Extrinsic support structures within Microsoft Publisher which present task support options to determine user needs. Decision tree structures are employed to drill down to increasingly specific needs. When common ground between the system and user has been determined, users are told to read, perform the task and press next to continue.

Attributes Illustrated:

5. Contain embedded knowledge in the interface, support resources, and system logic.
8. Provide alternative views of the application interface and resources.
12. Provide support resources without breaking the task context.
13. Layered to accommodate performer diversity.

Figure 12.



In Microsoft Publisher, the Cue Card structure is employed again, but in the task of supporting users through diagnosing their printing problem.

Attributes Illustrated:

2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources and system logic.
7. Reflect natural work situations.
12. Provide support resources without breaking the task context.
13. Provide layers to accommodate performer diversity.

structuring in *Publisher*, it will be discussed in addition to other important attributes such as Embed Knowledge, Observe and Advise, Provide Alternative Views of the Application, and so on.

Review Figures 9-12 in preparation for the discussion of Attributes 5-10. Explanations of some of the system's structures will be helpful in understanding the attributes.

Wizards: Also known as Assistants or Helpers, provide task structuring support that steps performers through processes associated with tasks such as designing a brochure or completing a Mail Merge (within *Microsoft Word*) or constructing a chart from data within a spreadsheet (Chart Wizards in *Microsoft Excel*). Wizards contain embedded knowledge. Wizards collect performer choices and automate task completion. Wizards appear as options whenever a new publication is created, or can be accessed from the Button Bar. Wizards are metaphors for assistants or magicians in the real world. Wizards provide an alternative interface to the traditional blank page with tools and buttons (the native mode of most desktop publishing software).

Cue Cards: Cue Cards are metaphorical coaches that structure progression through tasks while explaining the overall process, procedural steps, and related conceptual information. The Cue Card interface is reused across multiple contexts, including in the Print Troubleshooter (explained below) and when accessing explanations of conditions diagnosed by other support structures such as the Layout Checker.

Print Troubleshooter: The Print Troubleshooter (accessed from the Tools Menu) employs a dialog to determine the performer's problem and works the performer through the diagnostic, analytical, and problem-solving activities associated with resolving printing problems. Possible conditions are presented. Based on responses, conditional logic branches the performer through "next questions" or "next steps." Explanations about related procedures or concepts are accessible from the Troubleshooter. (The Troubleshooter is essentially a Cue Card structure employing underlying conditional logic.)

Checkers: The Layout Checker (not illustrated) evaluates design layout against a rule base and informs the performer of real or potential problem conditions. Advice on how to correct conditions is included. Links to Explanations available through Cue Card structures are provided. Checkers are metaphorical representations of the logic typically found in the head of a Quality Control expert.

Attribute

5 - Contain embedded knowledge in the interface, support resources, and system logic.

6 - Use metaphors, language, and direct manipulation of variables to capitalize on prior learning and physical reality.

7 - Reflect Natural Work Situations.

8 - Provide alternative views of the application, including:

- — interface,
- — logic (including task structuring support),
- — content,
- — monitoring, feedback, and recommendations.

9 - Observe and advise.

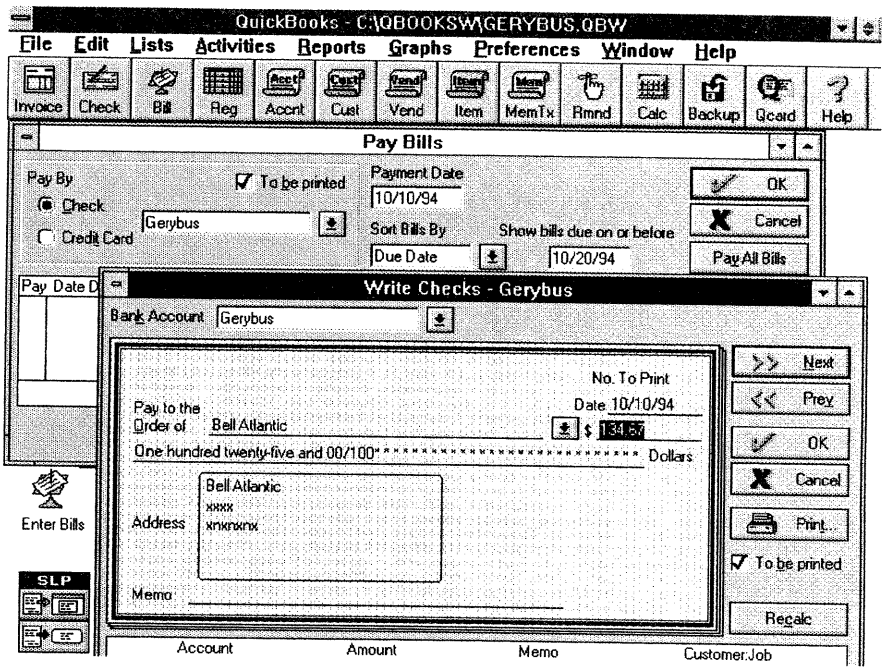
10 - Show evidence of work progression.

Embedded Knowledge: Knowledge exists in many forms—as explanations, demonstrations, examples, facts, and logic—which can be represented in tables, rule statements, decision trees, and so forth. In traditional software, the primary knowledge and logic associated with task performance focuses on data and data manipulation and transformation or functionalities associated with task performance such as cutting and pasting, formatting, and so on. The logic is largely hidden from users and is represented in the system's programming code. Task logic that is evident to the user includes sequences of data-entry fields and screen progressions within a transaction system or in any system messages that inform the user of options, incorrect actions and the like. In Microsoft Publisher, knowledge is embedded and tightly integrated through the application. Knowledge is contained in primary interfaces; logic and best practice is institutionalized in Wizards. Cue Cards and Quick Demos explain and direct. Message boxes inform and advise based on conditions and user interaction.

In traditional work environments, knowledge about task performance, work context, concepts, consequences, and so on resides almost exclusively in the job holder's head. Such knowledge and logic must be developed prior to using software and the performer must be inherently competent at calling relevant knowledge to the fore and structuring task progression. Performers must know what must be done, how to do it, and all knowledge related to that task. For example, when working in spreadsheet software, the user must understand what a spreadsheet is, what spreadsheets are used for and how to construct one; knowledge about software structure, procedures, and options such as graphing and embedding charts in spreadsheets is also required. Proper formulas must be constructed appropriate to the required data calculations and display. Knowledge of data organization and representation in relationship to a goal is necessary for proper spreadsheet design. Finally, understanding of *best method* for graphically depicting specific types and volumes of information in relationship to a goal is necessary for *effectiveness.* Successful task completion, it is obvious, requires far more than simply knowing how to interact with the software itself.

In performance-centered systems, knowledge related to *both* system utilization *and* work performance are incorporated in the underlying system logic, the interface itself, *and* within all support resources. This knowledge can exist in many forms, structures, and media types based on the task and performer analysis or task risks. For example, some knowledge may be embedded within a primary display or message box and be presented *every time* a task is performed or a condition exists (e.g., inquiry to the user about whether they want to save changed data). In primitive interface and communication environments such as DOS, the system presumes performers know what they want to do, what they've already done, the consequences of their actions, and so on. Responding *yes or no* to the "Are you sure?" message has probably generated more ulcers than any other condition in recent memory. I'm either sure and wrong,

Figure 13.

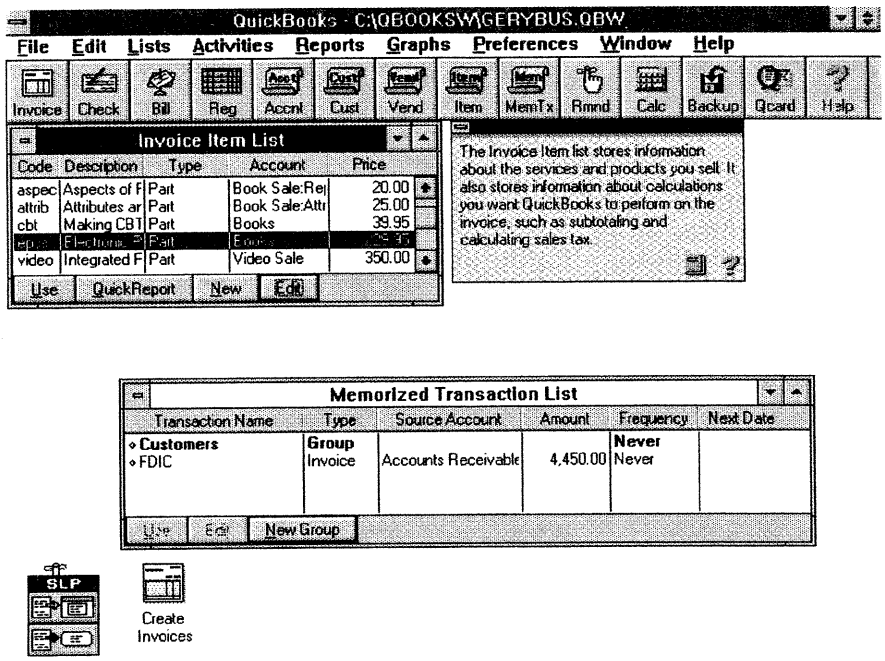


In Intuit's Quickbooks Small Business Accounting software, tasks such as Paying Bills and Writing Checks are performed through metaphorical objects such as Checkbooks. Alternatively, performers may work in a Checkbook Register and simply write checks through the Register itself.

Attributes Illustrated:

1. Establish and maintain a work context.
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
7. Reflect natural work situations.
8. Provide alternative views of the application interface and resources.
15. Automate tasks.

Figure 14.

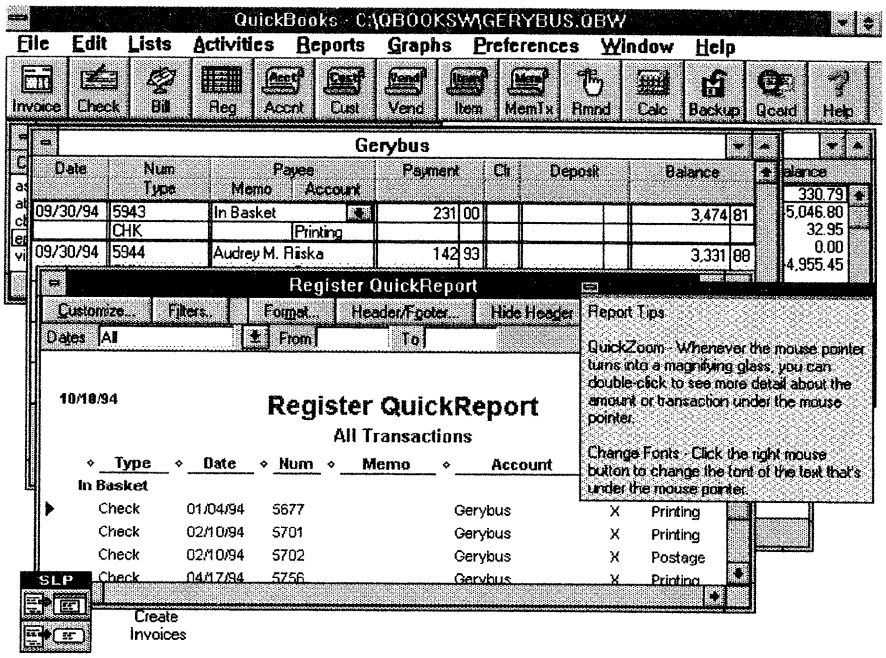


In Intuit's Quickbooks, users can Memorize frequently used transactions such as sending invoices, writing checks via the Memorize Transaction activity. Such task automation saves repetitive work. Memorized Transactions are available through the MemTX button. Qcards (to the right of the Invoice Item window) within Quickbooks provide on screen information about the activity at hand. Qcards can be turned on and off by task or as a default setting.

Attributes Illustrated:

3. Structure work process and progression through tasks and logic.
12. Provide support resources without breaking the task context.
15. Automate tasks.

Figure 15.



In Intuit's Quickbooks, users working in the Checkbook Register (Gerybus window) can generate a QuickReport of all payments to a given vendor (Register QuickReport window). Once in this window, access to underlying transactions supporting report items is available through the QuickZoom feature described on the Qcard to the right of the window. This integration of data, transactions, reporting is structured within the software based on an in-depth understanding of performer needs while working in or conducting analysis of financial information. The task automation makes previously cumbersome activities simple.

Attributes illustrated:

8. Provide alternative views of the application interface and resources.
13. Provide layers to accommodate performer diversity.
15. Automate tasks.
16. Provide alternative knowledge search and navigation mechanisms.

thereby ruining my work, or I'm not sure and can't proceed because, in fact, I am *not sure!* Clearly, embedded knowledge and advisory messages are superior resources.

Metaphors: Metaphors reflecting reality are used to represent space, objects, tasks, forms, and so on. Metaphors such as checkbook registers, physical objects such as gauges and sliders and forms are programmed into the software itself. Developing powerful metaphors to represent data, objects, tasks, and concepts is a high-leverage element of performance-centered design. In Microsoft Publisher, metaphors include Cue Cards and Wizards or Assistants. Structured visual metaphors are not as suited to the graphic design task as they are to others, but subtle metaphors, such as the conversational dialog, are evident. Figures 13-15 from Intuit's *QuickBooks* small business financial accounting software illustrates the use of metaphors supporting data entry and recordkeeping. *QuickBooks* employs a Checkbook metaphor and uses all of the familiar conventions such as registers, checks, and deposit slips to capitalize on broad-based cultural knowledge about working in such environments.

Natural and Realistic Language: Language employed in displays and available resources such as help, instruction and reference should always be biased toward customers and novices, not experts or systems developers.

Example:

In a credit card customer service situation, customers disputing charges would state that they had returned an item but were billed anyway. Representatives were required to categorize the dispute by selecting descriptions from a listing on a display. The proper selection for the returned item condition was "Credit posted as a Debit." The mismatch between customer and system language turned a simple matching task into a complex reflective task requiring in-depth knowledge of double-entry bookkeeping. This technical knowledge was irrelevant to customer service tasks and outcomes and simply added complexity and increased probabilities of error.

Achieving clear, non-technical language is always difficult for those immersed in a field, function, or organization with its own terms and abbreviations that are not obvious to novices or users with a different perspective. Designers must continually involve novices in language review so their requirements and points of view are incorporated into the interface and resources.

Alternate Views of the Application: Alternate views are achieved employing:

- Display attributes
- Interaction mechanisms such as:
  - commands
  - menus
  - buttons, dialog boxes, list boxes, and other GUI conventions
  - interviews and dialogs
- Interaction approaches and styles including:
  - direct manipulation of objects (such as drag and drop)
  - interaction with metaphors (such as gauges and switches)
  - forms

- command lines
- instructions

- Representation of content and data including multiple media such as:
  - text
  - graphics
  - animation
  - sound
  - video
  - virtual reality
- Customization capability

Provision of alternate views is complex to define and describe; it is best understood by example. It is the cumulative effect of many of the attributes described in this document and represented in the examples and software. In much the same way that a query language can be used to construct different combinations of data to produce a unique report to accommodate the specific needs of a requester, logic and interaction mechanisms can be used to construct various views of a system to accommodate the specific needs and preferences of a performer. Performers can select, or the system can present, combinations of structures, representations of content, and display attributes to meet specific needs. The term "Alternate Views" is intended to provide a defined (not infinite) range of alternatives. The goal is to accommodate diversity in knowledge, skill, and learning style, as well as preferred ways of interacting. For example, some people prefer to work with lists, and others with forms; some prefer command-level interaction, and others prefer working with buttons and mouse selection. The number and kind of alternative views must be defined in relationship to the expected benefits, and characteristics and requirements of the performer/user populations. *Technology for its own sake* (e.g., non-value-added multimedia) is unnecessary, expensive, and disruptive.

Several premises underlie the concept of alternative views:

1) Displays are dynamic and configured based on conditions, logic, data, and performer preference. *They are not static screens.* Message boxes, dialog boxes, overlay devices, and possibly additional windows are opened as conditions, performance, or data warrant. Note that this is in direct contrast to traditional mainframe approaches of working within a single data screen to perform a task.

2) Graphical User Interface conventions and attributes such as buttons, pull down menus, dimming and bolding, etc. are used to construct displays and permit alternate views. A performance-centered system is far more than a GUI interface, although many systems developed to date employ GUI conventions.

3. Multimedia capacity can enrich a performance-entered system by permitting more robust representations of content and logic and more different means for interaction. Multimedia representation of content does not *per se* define a performance-centered system.

4. To the extent that multimedia interactions (e.g. voice or pen-based input) or representations of content (e.g. a voice narrated demonstration of a physical process) accelerate the rate at which a performer can learn to interact with the application or understand a concept or procedure, multimedia can be viewed as a critical element of performance-centered design.

Observe and Advise: When performer actions, a relationship among data, the passage of time, or other factors result in a specific condition, the system presents visualizations of the data, information, options, suggestions, warnings or consequences to the performer using message boxes and dialog boxes and appropriate choice conventions such as radio buttons, list boxes, or check boxes.

The Socratic Dialog as Primary Interface Model: Wizards, Assistants and Helpers as Examples

One of the primary models being employed within successful consumer software is the Socratic Dialog. This model of an increasingly specific, dynamic question/response/conditional question or explanation human interaction is easily structured and produces a comfortable task context. The good news is that for most tasks, examples such as apprentice/master, consultant, and advisor/client abound. Commonly, control over the direction of human dialog shifts between parties. Likewise in this interface. Sometimes the experts ask a question and sometimes they provide information before asking questions or providing direction; sometimes the client asks a question in response to an expert question—and sometimes the client simply makes a choice.

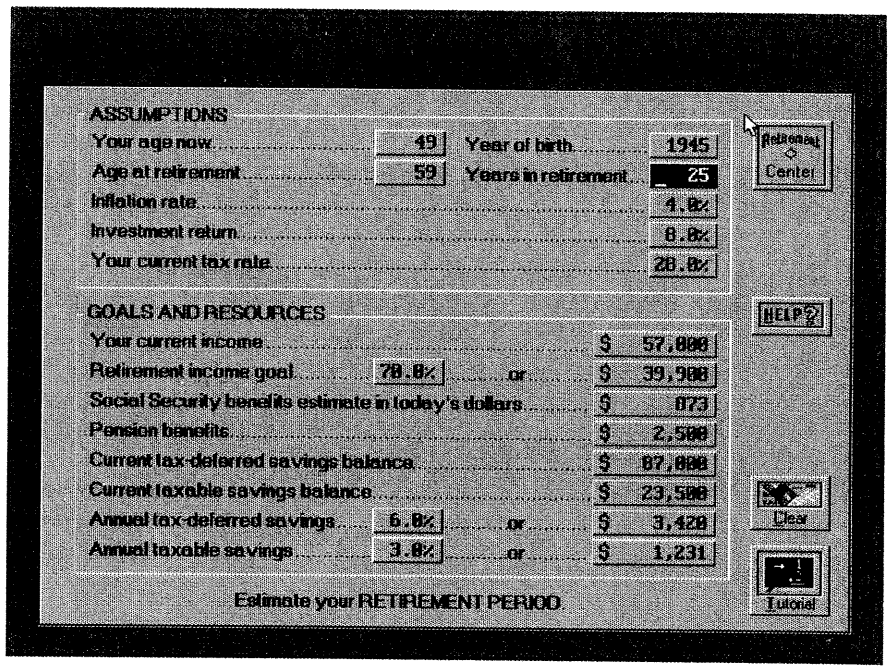
Microsoft Publisher's Wizards emulate the Socratic dialog by employing informal, sentence-like dialog such as:

- What do you want to do?
- Which of the following do you prefer?
- Enter the information (data) below...
- What are you experiencing?

In each case, options follow. Knowledge is tightly coupled with the task structuring. Sometimes consequences are explained or illustrated next to the option and sometimes they appear within a message box following the selection or input. For example, the consequences of choices within *Microsoft Publisher* are visually represented next to the Top Fold or Side Fold choices (See Figure 10.) Creating a digital equivalent of the experience of working with a knowledgeable person who both anticipates needs and responds conditionally requires thorough understanding of the task and related information and in-depth understanding of the frames of reference and needs of the performer or client.

In the *Microsoft Publisher* Cue Card support, the dialog and choice points are employed to progress performers through tasks. Anticipated additional information needs are available through "Why" or "How" buttons (not illustrated) embedded in the display. Activating the buttons links performers to more in-depth explanations or demonstrations. While not illustrated in the screen exhibits shown here, some software includes very robust content—such as "Show Me" buttons that activate voice-narrated demonstrations of procedures or processes or "Tell me about..." buttons that activate rich multimedia explanations of content. When value-added multimedia are employed, the time to understanding of a concept or process can be greatly reduced. But much can be achieved with limited text and graphical media. The key is

Figure 16.

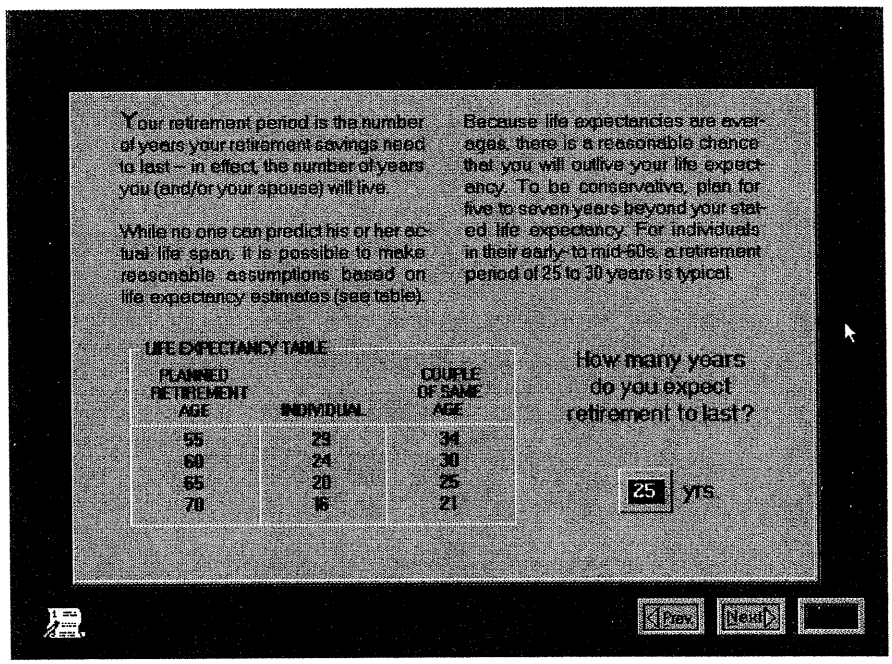


The worksheet mode of data entry in the Vanguard Retirement Planner permits those who understand the data required for retirement planning to interact with the software by filling in metaphorical forms. Context-sensitive Tutorial and Help resources are available to the user by clicking on labeled buttons. The Tutorial presents a guided walk through the fields and screen options with definitions and brief procedural explanations when appropriate. Accessing Help, in this case, accesses an alternative interface which is layered to permit more structured and more guided interaction -- not about how to enter data, but on what to think about when deciding such things as how long retirement will last.

Attributes Illustrated:

3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
8. Provide alternative views of the application interface and resources.
13. Provide layers to accommodate performer diversity.
15. Automate tasks.
17. Allow customization.
18. Provide obvious options, next steps and resources.

Figure 17.

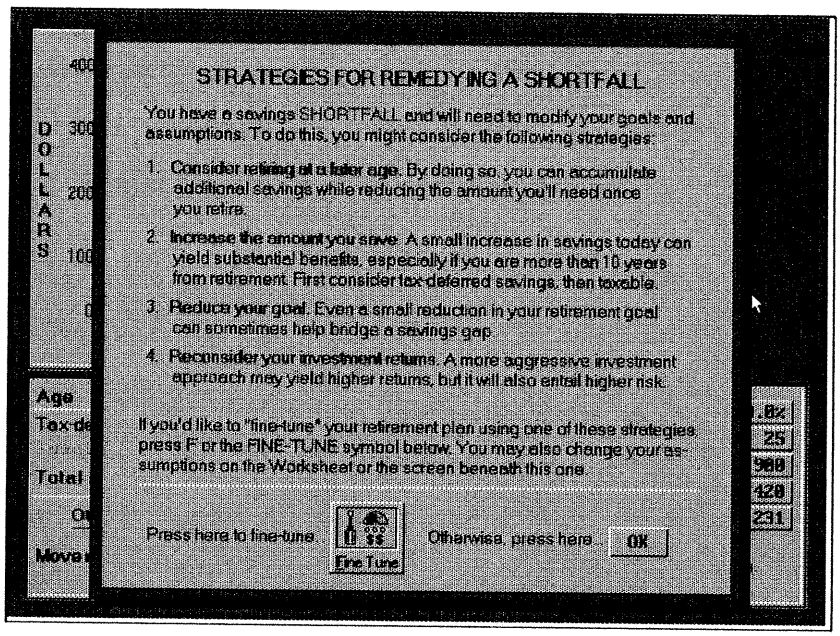


Deciding what values to enter into data fields often requires significant cognitive processing. In determining length of retirement, for example, users must consider their current age, health, family history, and other known factors. But often they lack sufficient detailed information about life expectancy in the broader population. This alternative interface in the Vanguard Retirement Planner provides the usual data entry field (yr. field on bottom right), but surrounds the data entry field with definitions, procedural information and advisory explanations. A life expectancy table is included. A universe of content is embedded within the data entry display. In this case the information is not layered, so more detail is not available, but that is a design choice. Developers can layer and link content as much as need, time, money and technology alternatives permit.

Attributes Illustrated:

3. Structured work process and progression through tasks and logic.
4. Institutionalize best approach.
6. Use metaphors, language and direct manipulation of variables.
7. Provide alternative views of the application interface and resources.
12. Contain embedded knowledge in the interface, support resources and system logic.
13. Provide layers to accommodate performer diversity.
15. Automate tasks.
17. Allow customization.
18. Provide obvious options, next steps and resources.

Figure 18.

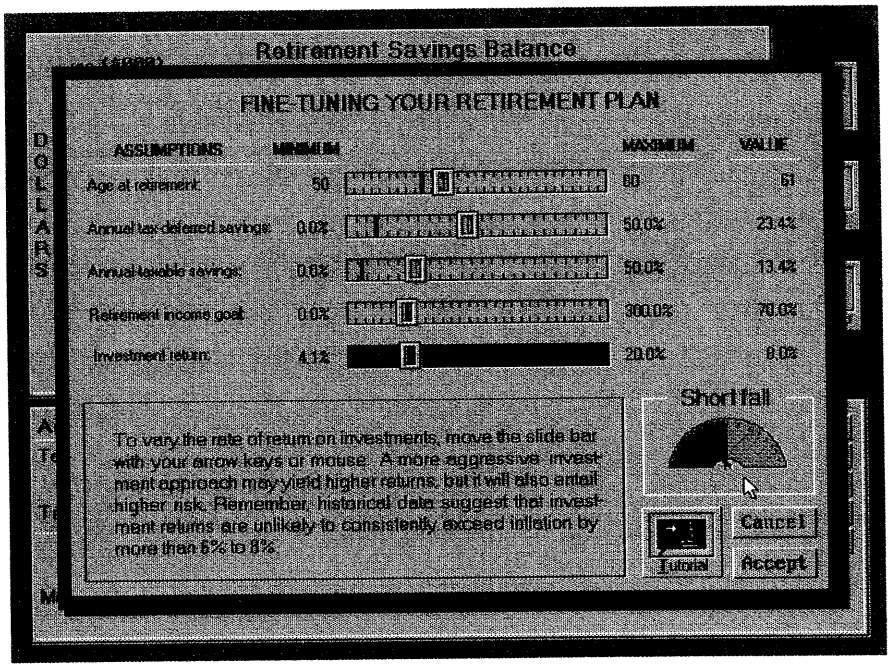


The Vanguard Retirement Planner presents users with evaluative information about their overall retirement resources once all data has been entered. The system automatically calculates values and, based on goals, savings, etc. summarizes the financial condition as adequate, surplus or, in this case, a shortfall. The four options to remedy the financial shortfall are presented. The user is informed of the mechanisms by which to implement those options: Worksheet screen, Overview screen (below the message box) and by accessing the Fine Tune button. Language is simple and clear.

Attributes Illustrated:

2. Aid goal establishment
3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources, and system logic.
6. Use metaphors language and direct manipulation of variables to capitalize on prior learning and physical reality.
8. Provide alternative views of the application interface and resources.
9. Observe and advise.
11. Provide contextual feedback.
13. Provide layers to accommodate performer diversity.
15. Automate tasks.
17. Allow customization.
18. Provide obvious options, next steps and resources.

Figure 19.

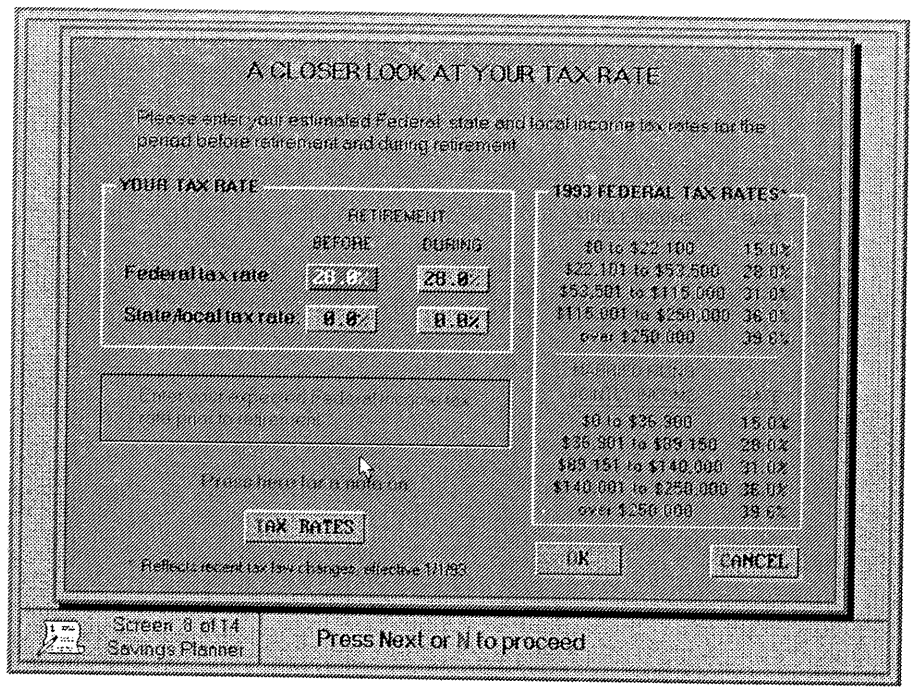


This interface within the Vanguard Retirement Planner provides a rich analysis and variable manipulation environment. Interaction instructions are embedded on the lower left of the display. In addition to procedural content, users are presented with things to consider in the situation. Users can view their own data values (right column and the vertical line on each slider bar) in relationship to the four options and the range of minimum to maximum values accepted by the software. Those values can be manipulated by using the metaphorical indicators (square blocks) on the slider bars. Associated values are changed on the display in response to manipulation. The metaphor of a gauge (half-moon object) is employed to summarize all of the elements in a single interpreted value indicating shortfall or surplus. The gauge moves directly as users manipulate variables and permits users to focus precisely on their financial condition in relationship to their retirement goal. Access to more detailed navigation and explanation tutorials is available through the Tutorial button.

Attributes Illustrated:

2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
5. Contain embedded knowledge in the interface, support resources, and system logic.
6. Use metaphors, language and permit direct manipulation of variables to capitalize on prior learning and physical reality.
8. Provide alternative views of the application interface and resources.
9. Observe and advise.
11. Provide contextual feedback.
15. Automate tasks.
18. Provide obvious options, next steps and resources.

Figure 20.

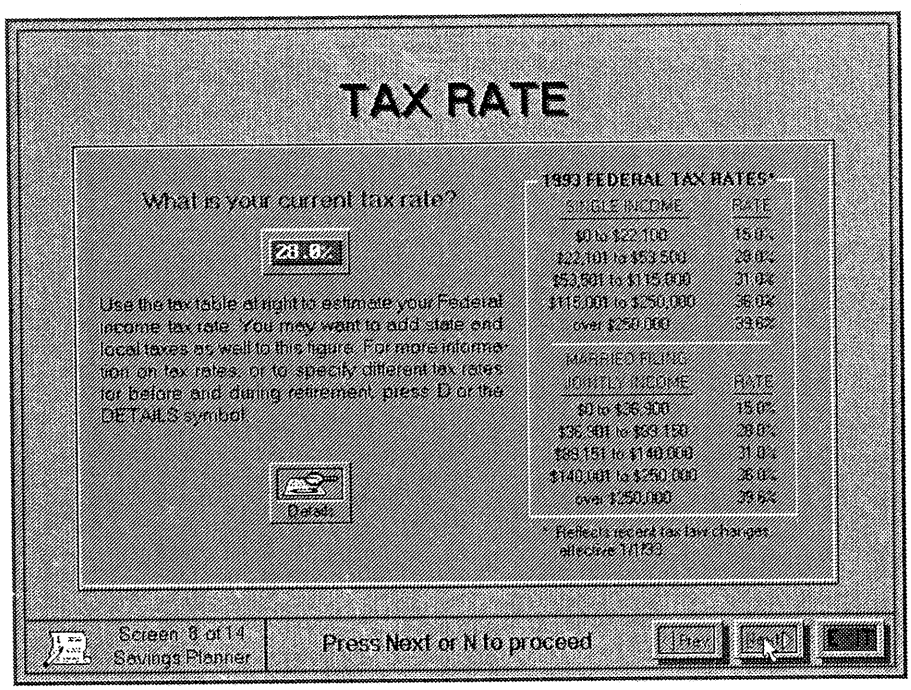


Once the Details are requested, more precise financial information can be entered in the Vanguard Retirement Planner (see Figure 19).

Attributes Illustrated:

3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources, and system logic.
8. Provide alternative views of the application interface and resources.
12. Provide support resources without breaking the task context.
13. Provide layers to accommodate performer diversity.

Figure 21.

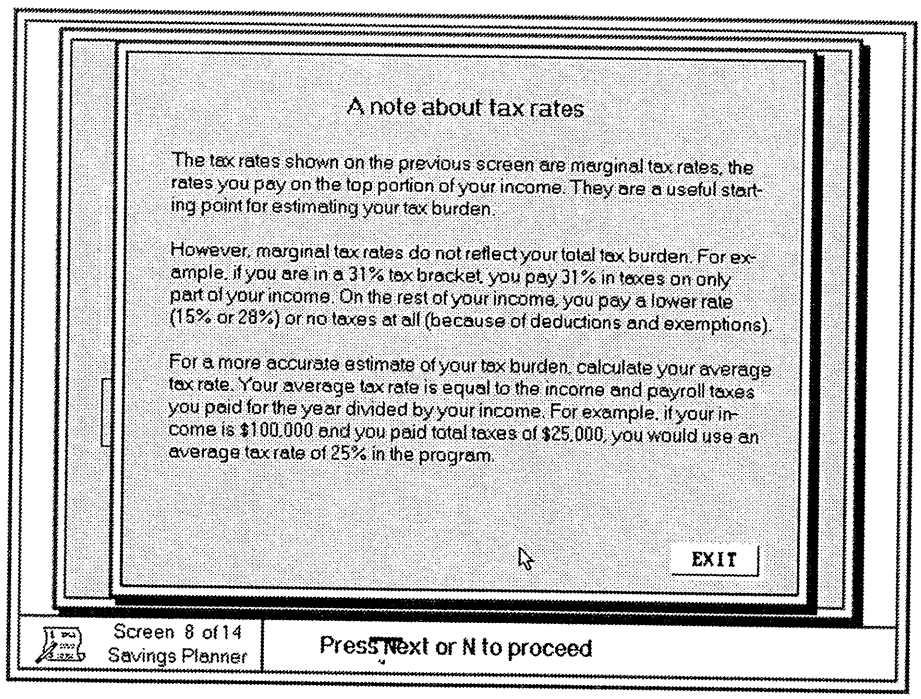


In the Vanguard Retirement Planner, different motivations and goals of users are accomodated by layered displays. If a relatively superficial retirement planning review is the objective, users can enter in high-level information such as gross taxation rates. Younger people or people conducting initial analysis may find this sufficient. But as planning becomes more detailed or retirement is more imminent, more precise information is necessary. By clicking on the Details button, users can access a display structuring more detailed and precise data input. Note that the tax tables are embedded in the display along with other procedural information.

Attributes Illustrated:

3. Structure work process and progression through tasks and logic.
4. Institutionalize business strategy and best approach.
5. Contain embedded knowledge in the interface, support resources, and system logic.
8. Provide alternative views of the application interface and resources.
12. Provide support resources without breaking the task context.
13. Provide layers to accommodate performer diversity.

Figure 22.



Invoking the "A Note About Tax Rates" button results in this granular but sufficient explanation of the reasoning behind such concepts as marginal income tax rates. The "A Note About..." label is more seductive than a label reading "Tax Lesson." This just-in-time and just enough resource exemplifies integrated and embedded knowledge.

Attributes Illustrated:

5. Contain embedded knowledge in the interface, support resources, and system logic.
12. Provide support resources without breaking the task context.
13. Provide layers to accommodate performer diversity.

to understand the needs of the performer constituencies.

More Detail...

Interview or dialog formats present alternative goals for selection by performers. Supporting information helps performers understand and compare options. In an initial display, the system might ask "What do you want to do?" and will then present options via GUI objects such as buttons, list boxes, check boxes, pick lists, etc.

Only available options are presented to performers to help narrow possibilities and alternatives within a given context. Menus have available options in bold and unavailable options either dimmed or not presented at all.

Available and appropriate tools are presented at the moment of need as defined by context, system state, input values, and other predefined conditions. Tools such as layout checkers, troubleshooters, and Wizards may be system or performer controlled. These tools aid the performer in determining whether or not they want to perform an activity that experts judge to be appropriate at that point.

Knowledge is sometimes embedded within the software, but independent of the primary displays or interactions. In these cases, the knowledge is represented in extrinsic structures such as Cue Cards, Searchable Reference, Tips, and so on. This knowledge can be organized in many alternative ways: 1) context-sensitive or not, 2) searchable or not, 3) layered or not, 4) in one or more representations such as text, graphic, animation, video, sound; and 5) continuously turned on or available only when invoked by the performer.

Extrinsic Support Structures: Mechanisms for Embedding Knowledge and Structuring Performance

Discussion of types and uses of extrinsic support structures is an article in and of itself. For purposes of this discussion, it is sufficient to cover the requirements for embedded knowledge and the associated structural possibilities. (See Table 2, Extrinsic Support Structures and Their Uses.)

Briefly, design considerations and alternatives include:

- Presentation of options that are based on the situation, with user control as to whether to pursue those options (such as related instructional sequences).
- Presentation of comments, suggestions, or tips with the user option of continuing to permit this comment or turn off system checks, comments, and communications. Cue cards, balloon help, checkers, and tips are examples.
- Presentation of requirements, such as needing to change config. sys. files, with user option to have it done automatically.
- Presentation of the option to view relevant instructional or reference content through dialog or message boxes, buttons on primary displays, or menus. This permits the performer to determine whether, when, and how often to access and/or complete it.

Many of these attributes are closely interrelated, but each is slightly different from the others. The

common outcome of these attributes is maintaining performer orientation to the situation, creating understanding of the consequences of actions at the earliest possible time to help the user avoid repetitive loops through the task cycle. For example, in Microsoft Publisher the visual representation of the accumulation of choices between brochure style and Fold options (i.e., Top Fold vs. Side Fold) permits performers to see how their documents will look before printing—enabling them to identify the specific choice that led to the consequence condition.

Software Illustration of Attributes 11-13 and 15: *Vanguard Retirement Planner*

Attribute

11 - Provide contextual feedback.

12 - Provide support resources without breaking the task context.

13 - Provide layers to accommodate performer diversity.

15 - Automate tasks.

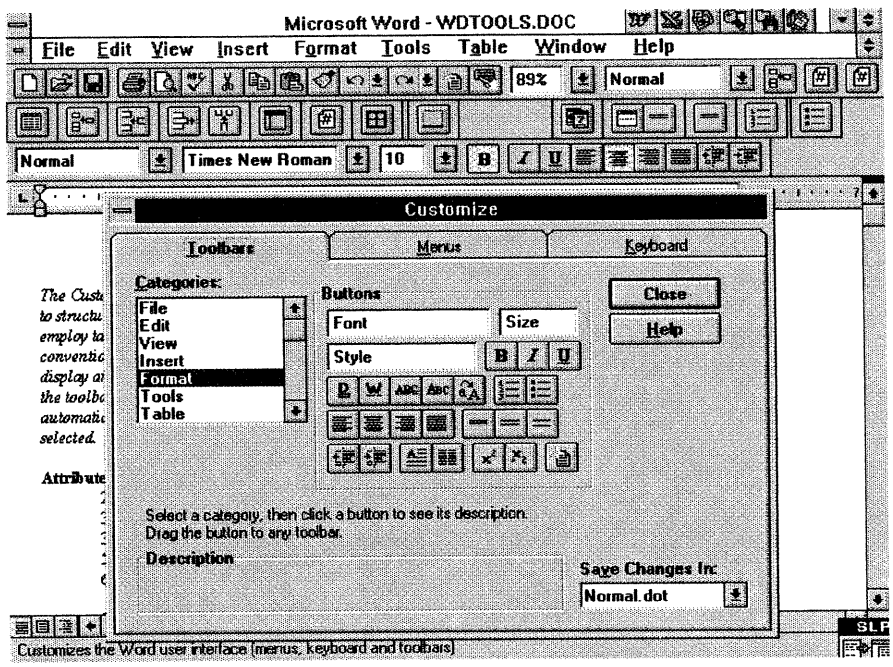
The *Vanguard Retirement Planner* is a DOS tool supporting individuals in evaluating their financial condition and goals and planning for their retirement. This software tool is made available by Vanguard Mutual Funds to assist customers in this aspect of financial planning and selection of appropriate investment vehicles. Informed customers are bigger customers and better customers, and they take less non-sales time of telephone customer service representatives. In addition, by linking Vanguard product information to the client's financial information and organizing it by investment risks and kinds of investments, customers are able to independently evaluate their alternatives and make product selections most suitable to their needs, goals, time frames, and risk tolerance.

The *Retirement Planner* evidences considerable performance-centered design. Mention will be made here of previously discussed attributes, such as task structuring and alternative interface modes, but the emphasis will be on illustrating Layering and Task Automation.

Contextual Feedback: Contextual feedback is simply providing relevant, context-sensitive information about states, conditions, judgments, what is going on, and so on that enables the performer to understand what is happening while interacting with the software. For example, in the *Vanguard Retirement Planner* (Figures 16-22), each time data is entered that will have an impact on Social Security benefits, a message box appears stating "Social Security benefits are being recalculated." The performer understands the delay and more importantly—begins to understand the relationship between the variables in a situation.

To the extent possible, the intervention maintains visual, positional, data, and/or other contextual mechanisms and does not appear to put the performer into a "space" that is visually different from the primary workspace. If possible, performers should not have to radically change contexts into a support space and then return to the primary space. This is most easily accomplished when visual objects convey meaning, relationships, or status, or when common interface conventions such as dialog boxes are used to communicate

Figure 23.

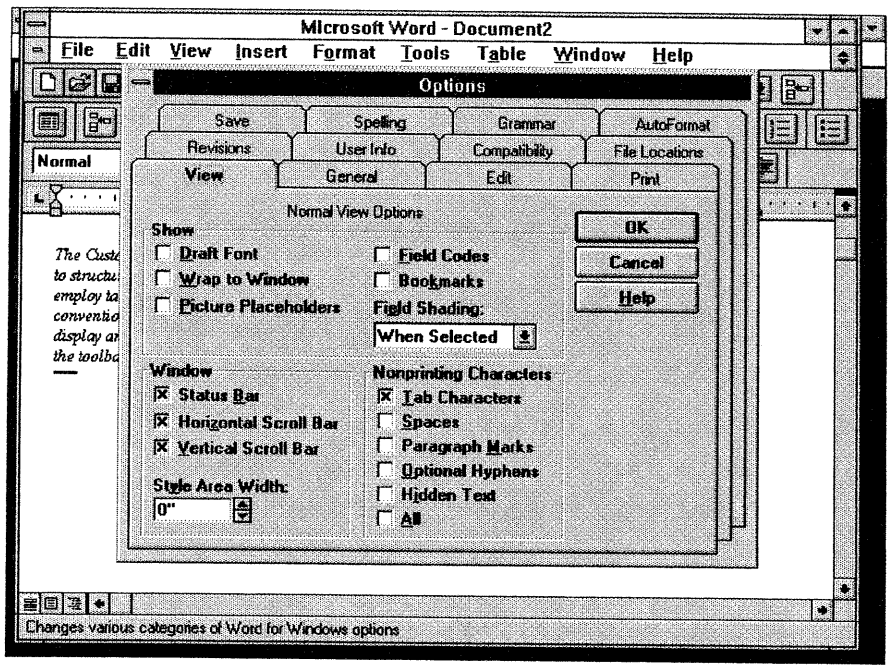


The Customize option selected from the Tools menu in Microsoft Word 6.0 for Windows permits users to structure their tool bars to accommodate personal working styles. The metaphorical file folders employ tabs for selecting Toolbar, Menu or Keyboard customization options. Employing the conventional list box to select categories of customization, users then follow directions embedded in the display and can click on options to obtain descriptions of the item. They then drag their selection to the toolbar, menu, etc. In total, hundreds of options are available to customize their workspace. Word automatically executes the logic associated with the options and interface changes as options are selected.

Attributes Illustrated:

2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
5. Contain embedded knowledge in the interface, support resources and system logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
8. Provide alternative views of the application interface and resources.
10. Show evidence of work progression.
12. Provide support resources without breaking the task context.
15. Automate tasks.
17. Allow customization.
18. Provide obvious options, next steps, and resources.
19. Employ consistent use of visual conventions, language, visual positioning, navigation and other system behavior.

Figure 24.



In Microsoft Word 6.0 for Windows, users can view all options around software displays, functionality and behavior in a single universe. Employing the file folder metaphor and traditional list box, check box, radio button and button conventions, over a hundred options can be viewed. Given the number of alternatives and settings, users require a simple space within which to view all previous and possible options in an integrated space.

Attributes Illustrated:

2. Aid goal establishment.
3. Structure work process and progression through tasks and logic.
5. Contain embedded knowledge in the interface, support resources and system logic.
6. Use metaphors, language and direct manipulation of variables to capitalize on prior learning and physical reality.
8. Provide alternative views of the application interface and resources.
10. Show evidence of work progression
12. Provide support resources without breaking the task context.
15. Automate tasks.
17. Allow customization.
18. Provide obvious options, next steps, and resources.
19. Employ consistent use of visual conventions, language, visual positioning, navigation mechanisms and other system behavior.

and when system behavior is consistent. In addition, minimalist resources such as Cue Cards can be overlaid on displays, or the system can provide content via message boxes generated by conditions, data, or preset preferences. The key is to ground resources in the task context (see the discussion of Attribute 1). The ultimate design goal is fluid and minimally disruptive movement between the task performance and the resources presented. It is best accomplished by providing intrinsic support.

Currently, many software developers are providing textual and graphical resources to performers via context-sensitive help features using Microsoft Windows Help or equivalents such as ROBO Help, Microsoft Multimedia Viewer, or the OS/2 Information Presentation Facility. The tradeoffs in this practice include:

- ease of development,
- technological simplicity, and
- consistency in help system behavior across applications,

versus...

- disrupting performers by breaking context with the native application (i.e., help is definitely a separate software space), and
- a tendency to default to wall-to-wall text rather than overlaying the most powerful support resources (e.g., a narrated demonstration, a "Try-it" exercise, a graphical representation of a concept, or an instructional sequence).

Evidence of progression through the task or task status is presented throughout the process. This evidence is a significant form of contextual feedback. The representation of task progression can be by graphic, text, sound, or image. Performers should be provided with continuous feedback to orient them to where they are within a process or activity. Examples of feedback styles include

- bars indicating percentage of task complete,
- visual evidence of the deliverable in progress,
- progression through a fishbone diagram, flowchart, process map, etc.,
- listing of steps or requirements with check marks,
- color cue of completed steps or processes, and/or
- audio status sounds.

Unobtrusive Support Resources: This attribute is closely related to embedded knowledge in that information and links to related resources are part of the interface. There are other support resources or tools which may be extrinsic to the system—but they are presented in highly integrated ways. For example, in *Microsoft Publisher*, a message box appears after a document is printed. It reads "Are you satisfied with the way your document printed? If not, access the Print Troubleshooter from the Help menu for assistance with your problem." An even better alternative would be to permit button access to the Print Troubleshooter through the same message box. This, however, is a matter of elegance and integration. The important point is that all of these resources and possibilities ebb and flow in a seamless dialog between the performer and the system. Instructional content in mes-

sage boxes summarizing conditions, explaining alternatives and linking to relevant resources is far less intrusive than booting someone into a modular, computer-based training sequence that disrupts the task and the performer.

Layering: Layering is a powerful design alternative that permits accommodation to a broad range of performer goals, interests, capabilities, knowledge and skill, or time available. Layering can be applied to the interface itself, related knowledge resources, data representation, and other system elements.

Layered or Alternate Interfaces. With this type of layering, performers can select more or less structure to be provided by the interface based on their competence, confidence, time available, preference and goals. This is best understood by a comparison with the feature of expanding or collapsing amounts of detail when working in outlining mode in presentation software such as Microsoft PowerPoint, Aldus Persuasion, or Symantec's More. Based on interest or focal point of attention, users can look at 1, 2, 3, or more levels of the outline for either the entire outline or for a sub-section. Similarly, when an interface or content is *layered* or has alternative views, the user can request more or less *task structuring, embedded knowledge, or communication from the system.* For example, using Page Wizards within Microsoft Publisher provides a more detailed structure for task performance than the blank page.

In the *Vanguard Retirement Planner*, layering is extensively employed. Figures 16 and 17 illustrate *layered interfaces.* Users can interact with the software in a worksheet or forms mode by entering data into labeled fields. When uncertain of how to complete information such as Length of Retirement, pressing HELP invokes a layered interface which presents significantly more detail about definitions, considerations, and resources such as the Life Expectancy Table to assist in informed decision making.

Figures 20, 21, and 22 illustrate *layered content* in the Vanguard product. When entering tax rate information, simple data-entry fields are an option. Additional conceptual and factual information is available by clicking on the "A Note About Taxes" button. This explanation or tutorial is available only when needed and desired. Similarly, customer motivation for more or less detailed analysis can be accommodated by permitting interaction at varying levels of detail.

Layered Knowledge or Content. Knowledge or content about concepts, procedures, definitions and so on can be organized for progression through various levels of detail based on performer interests, needs, goals and time available. Content layering is accomplished using hypertext links, access through buttons labeled *How, Why, Tell me about... Show me...* and so on.

Layering can occur within the primary interface itself—or in extrinsic resources such as Cue Cards or Searchable Reference employing hypertext capabilities. When storage limitations exist, sometimes layering is accomplished by cross-referencing to external resources such as manuals or named expert resources.

Automation: Performance-centered systems automate as much of the following as possible:

- navigation
- data extraction, manipulation, representation and visualization
- deliverable creation
- editing
- evaluating and checking
- application of rules to conditions or data
- extraction, integration and presentation of content or knowledge

For example, presenting only the data that is appropriate to a context or customer minimizes the requirements for analysis and filtration by the performer. Rather than be required to generate the commands, syntax, and sequence associated with a task such as Mail Merge, the system can automate the process to permit the performer to focus on what needs to be done, rather than on how to do it.

Automated checkers, which compare input or conditions, contrast it against rules, and present alternatives for the performer to evaluate and decide upon, are moving beyond limited tasks such as spell checking to more rule-based or context-specific tasks such as grammar or language checking, layout checking, or evaluation against rules of a discipline such as troubleshooting, diagnosis, risk evaluation, and so on. In *Microsoft Publisher*, the Layout Checker evaluates against rules, presents conditions, and recommends alternatives for solving layout problems that might not have been detected by the performer (e.g., object in a non-printing area). In *Performance Now!* (a performance evaluation system), the Language Checker evaluates for words that might be inflammatory and create legal or employee relations problems. In various proprietary sales systems, sales representatives employ automated configurators to identify products appropriate to customer requirements, parts necessary for installation, and comparisons with competitor products.

In the *Vanguard Retirement Planner,* whenever data changes, Social Security benefits are recalculated and related data is automatically changed throughout the software. User data is represented on a display summarizing retirement data, pre- and after-tax savings and retirement goals. By employing the metaphorical slider bars (see Figure 19), users can manipulate their variables (e.g., increase their retirement age). The gauge indicating total shortfall or surplus moves directly in relationship to the variable manipulation, thereby automating expressing the complex mathematical relationships among and between variables in the visual display. This automatic activity permits users to focus on their goals, rather than on performing tedious calculations.

The design goal is to identify tasks subject to automation employing macros, calculations, or additional programming and to automate them whenever possible to free the performer up for their primary work and to permit them to focus on tasks currently requiring human thought or action.

Software Illustration of Attribute 17: *Microsoft Word 6.0*

Microsoft Word 6.0 for Windows is a powerful word processor with huge amounts of functionality and options. Not every user wants or needs every function. Some functions are critical for some tasks and not others. And so Microsoft created a standard envi-

ronment that could be customized by each user.

Attribute 17 Allow Customization.

Performance-centered systems provide various kinds of customization options which include

- **Display Layout and Features** Displays can be configured in terms of what, where, and how things are displayed. Buttons can appear or not, with or without text labels on the top, left or right side, or bottom of the display.
- **Type of Interface** (See the detailed discussion of Layering under Attribute 13.)
- **System Settings** These include color options, on and off buttons for various features, and other optional system behaviors.

Optional system configuration by users is increasingly observed in consumer software products, but is less prevalent in custom software designed by internal software developers in organizations. This may relate to perceived needs to control performance—or the view that "developers know better than performers what performers need." The nature and degree of customization options requires conscious thought and should be based on detailed analysis of the range of performers using the software, consequences associated with flexibility, and costs.

Customization options, while less common, are emerging in increased choices for establishing preferences and selecting options. Figures 23 and 24 illustrate the broad range of customization options within *Microsoft Word.* Each folder tab contains choices that are selected employing radio buttons, check boxes, or lists. Over 100 choices are possible. It's clear to see that one person's Word is not another's! And so it will go. Individuals defining their computer-mediated work space to suit their needs, tastes, objectives, knowledge, and skill.

Attributes Not Illustrated by Software Examples

The remaining attributes of performance-centered systems will be only briefly discussed. Some are self evident, such as being consistent in displays and system behavior. Others require more discussion. Some, such as Attribute 15 *Provide alternate search and navigation mechanisms,* are commonly observed in current consumer product HELP systems which permit hierarchical views, relational views, alphabetical views, and contextual views of information. Search methods include browsing, hypertext button access, key word search, and button access which generates context-sensitive information.

Attribute 14 Provide access to underlying logic.

In order to develop mental models and understanding of rules and relationships, performers need to understand the underlying basis for conclusions or next steps. While this is not a commonly available feature at present, it will become increasingly useful for performers to understand how and why the system came to its recommendations. This will become increasingly important as configurators, troubleshooters, and support for other complex decision making tasks are developed. Access to that logic could be made available

through a "Why" button — or the system could present its thinking in narrative form.

Attribute 16 Provide alternative search and navigation mechanisms for knowledge or data.

Performance-centered systems permit numerous ways to access information. Methods include access through alternative structures such as:

- alphabetical listings
- query-specified searches using words, phrases, and combinations, exclusions or delimiters
- indices
- context-sensitive access
- comparisons with known alternatives (such as "sounds like")
- jumps to pre-linked screens, content, data

The system effectively treats content as a database, even if it is not technically organized as such. Access to the content can be made available through:

- content embedded in the display,
- help from the general Help menu or within displays,
- hypertext link access from displays or within Help,
- access through glossaries,
- message boxes presented by the system,
- buttons within displays,
- button bars,
- Find commands or search options, including key word, tiled, or tree-structured displays,
- tables of contents, and/or indices.

Alternate access methods are growing as more powerful technologies become widely available. The design goal is to accommodate performer capabilities, limits, frames of reference, and thought process, and permit the range of options necessary to achieve required content access.

Attribute 19 Consistent use of visual conventions, language, visual positioning, navigation and other system behavior.

Achieving this attribute is the result of consistent application of standards for a given piece of software. The standards should, of course, be grounded in ergonomic, usability, and performance research. It is sufficient to understand the user expectations for system look, feel and behavior should be consistently met to avoid performance deterioration in primary tasks.

Summary

The goal of performance-centered software design is 1) to integrate the knowledge, data, and tools required to be successful in performing a task, and 2) to provide task structuring support to help performers create required deliverables. The focus is on creating a computer interface that provides intrinsic support for the task within the interface itself, and contextually linked extrinsic resources that will generate *day-one* performance in all classes of performers. Achievement of this goal is eminently possible when the right mix of people have performance-centered design as a shared goal and when the measurements, incentives, and reward systems are in alignment.

The task of the performance technologist is to influence the design based on an in-depth understanding of the context, task requirements, deliverables, and performer popula-

tion. Designing performance-centered, computer-mediated environments will be the work of the next twenty years. The attributes and behaviors described above are an initial attempt to provide definition to our goals—and guidance to our efforts.

References

Microsoft Publisher. Microsoft Corporation, One Microsoft Way, Redmond CA 98052.

Performance Now! Knowledge Point, 1129 Industrial Avenue, Petaluma CA 94952.

Quickbooks. Intuit, Box 50630, Palo Alto CA 04303.

Vanguard Retirement Planner. The Vanguard Group, PO Box 2600, Valley Forge PA 19482.

WORD for Windows 6.0, Microsoft Corporation, One Microsoft Way, Redmond CA 98052.

GLORIA GERY is an independent consultant specializing in interactive learning and performance support systems. She consults primarily in Fortune 500 organizations aiding Training, Documentation, Systems and Business Management in reframing thinking about how performance and learning can be accelerated by exploiting both available and emerging technologies. Her clients include American Express, Hewlett Packard, and AT&T. She is the author of *Electronic Performance Support Systems* (1991) and *Making CBT Happen* (1987), both published by Ziff Institute, Cambridge, MA, each in its fourth printing. Books must be ordered through the author. *Mailing address:* 108 South Trail, The Tunxis Club, Tolland, MA 01034. *Phone: (413) 258-4693.*