

SALVO: A BASIC METHOD FOR USER INTERFACE DEVELOPMENT

E. Vance Wilson
University of Wisconsin-Eau Claire

James R. Connolly
California State University, Chico

ABSTRACT

SALVO answers a need in the Information Systems undergraduate curriculum for a user interface development method that is basic, prescriptive, and suitable to situations where the topic coverage is limited. The SALVO method is a five-part, iterative process focusing on areas of user interface development that are foundational to Human-Computer Interaction pedagogy.

INTRODUCTION

User interface (UI) development is an important topic in the undergraduate Information Systems (IS) curriculum. However, IS students typically receive only a cursory introduction to the topic due to two factors. Treatment of UI development and other aspects of human-computer interaction (HCI) is spread thinly across the curriculum. The IS '97 model curriculum (Davis, Gorgone, Couger, Feinstein, & Longenecker, 1997) suggests that in-depth study of the topic should be presented in programming language and physical design curricula. But UI development is not central to either of these courses, so depth of students' learning is necessarily limited by the brief amount of classroom time that can be devoted to the topic.

Compounding the problem is the tendency for textbooks to avoid the issue of *how* to develop a UI. Instead, texts tend to provide a broad survey of UI types, including command line mainframe interfaces (CLI), graphical user interfaces (GUI), multimedia interfaces, and next-generation virtual reality interfaces. From their texts IS students may come to understand what a dialog box is and receive tips to "use command verbs clearly," but they do not learn a specific method for UI development that is as applicable in their careers as entity-relationship

diagramming and flowcharting techniques are to data modeling and programming, respectively.

Because HCI specialists are relatively rare in the IS profession and UIs are a major part of current practice in event-driven programming, it is very likely that IS students will become responsible for developing UIs at some time in their careers. Thus, there is a compelling need to mitigate the limitations of course time constraints and unfocused texts in the IS curriculum.

In this paper, we present a method for UI development that is adaptable for use as stand-alone training or as an augmentation to textbook treatments of UI development. In the following sections, we describe the method and relate it to steps of the traditional systems analysis and design process.

THE SALVO METHOD FOR UI DEVELOPMENT

salvo 1. a discharge of artillery or other firearms in regular succession, often performed as a salute.
2. a round of cheers or applause. (Webster, 1989)

IS specialists in HCI typically complete one or more courses in User Interface Design, supported by study in

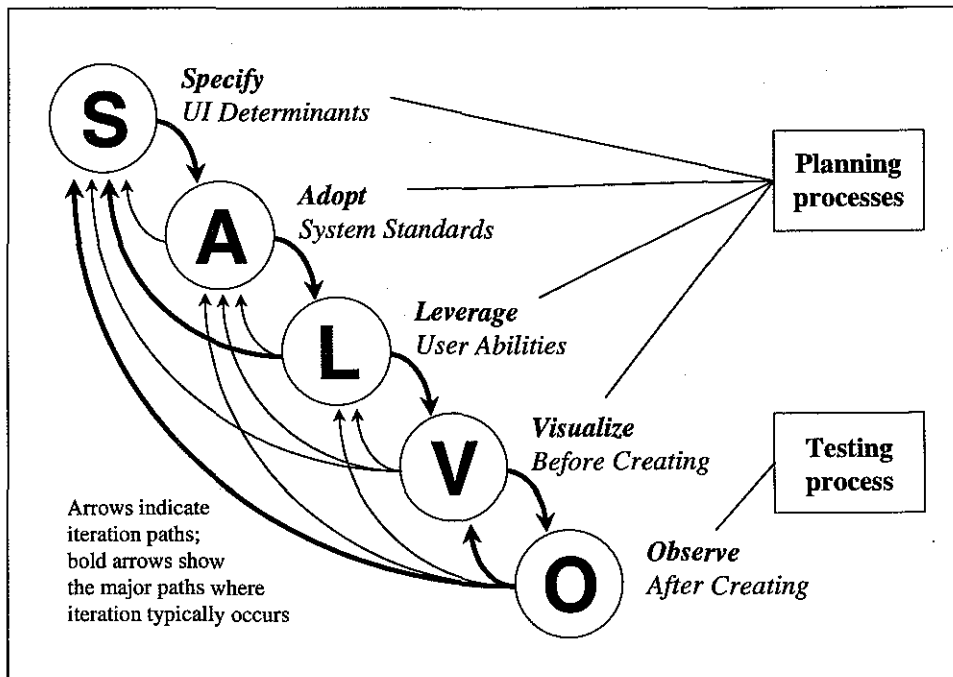
Cognitive Science, Human Factors of Information Systems, and Electives in HCI (ACM SIGCHI, 1992). We do not propose that it is desirable to incorporate this level of focused training within the mainstream IS curriculum. Instead, our approach draws key elements from HCI user- and task-centered design techniques and integrates these into a streamlined method. Our objective is to enhance the instruction of UI development within systems analysis and design or advanced programming courses without the need to alter existing course schedules.

There are three major criteria for this method. First, the method must be sufficiently basic that students can quickly understand its major concepts. Although a basic method inherently lacks depth, our intention is to provide a skeletal framework which students can flesh out with experiential learning gained during their careers. Second, the method must prescribe specific activities or ranges of activities to be performed in a systematic fashion. At the

same time, the method must avoid relying on particular technologies or other contingencies that would restrict flexibility in its application. Third, it must be possible to present the method effectively during a single class period of 50 to 80 minutes length.

The method we created is SALVO, an acronym for the five-step process illustrated in Figure 1. The name also serves as a mnemonic aid to help students recall the steps involved in the method as well as to reinforce the method's iterative nature. The SALVO method implements a streamlined set of planning and testing processes for UI development. In our own courses, we find that SALVO is adaptable to a wide range of programming environments and UI formats, including CLI and GUI. In the following sections, the individual steps in the SALVO method are described, and suggestions are offered for integrating the method into standard systems analysis and design procedures. The Appendix demonstrates the results of applying SALVO in an actual student UI development assignment.

FIGURE 1
THE SALVO METHOD



S: SPECIFY UI DETERMINANTS

Specify the factors that are key in deciding what form the UI should take and what functions it should provide; these are the UI determinants. The process of listing UI determinants is a planning activity in which developers pinpoint their initial views of "what's important" in the UI and then refine the list by interviewing system users. The list that results from the specification stage will be instrumental in guiding subsequent UI design and testing. Each specification must be sufficiently detailed to avoid abstractions that may otherwise mask UI problems. For example, focusing on the average age in a department of data entry workers could mask the fact that a sizable portion of the workers are over 45 years old, and that these specific workers have great difficulty reading small font sizes on a computer monitor.

Prototype Determinants Document

For UI development in organizational settings, three categories of factors should be examined:

- the users of the UI—*who will use it?*
- the tasks that users will perform with the UI—*what will it be used for?*
- the environments in which the users operate the UI—*where and how will it be used?*

Developers typically have preconceived ideas as to what will be important UI determinants, but their preconceptions often are viewed with suspicion by HCI experts:

The biggest problem we have is one of attitude—the arrogance of ... programmers who make arbitrary decisions about how the user shall use our products....They tend to see the world as composed of

- (a) people who are like them, and
- (b) people who really can't cope, and can't be taken seriously. (Tognazzini, 1992, p. 218)

Despite these concerns, we propose that developers' preconceptions can be used productively. First, ideas should be collected and listed in a prototype determinants document (see Table 1). Making individual beliefs explicit allows them to be considered and discussed by all members in the project, and it is logical that experience gained from past projects in an organization will make a good starting point for future projects. Second, the prototype determinants document guides subsequent interviews during which specifications of users, tasks, and environments are augmented and refined, or discarded and corrected. Having a prototype in hand focuses interviews and speeds up the overall planning process.

TABLE 1
PROTOTYPE DETERMINANTS DOCUMENT WITH SAMPLE ENTRY HEADINGS

User	Task	Environment
Name	Scenario	Technology
Job title	Simple Tasks	Security Requirements
Computer proficiency	Complex Tasks	Management Structure
English language proficiency		Physical Facilities
Data entry speed		Marketplace Competition
Visual acuity		Industry Practices
Computing system		Legal and Regulatory
Use for new system		Socio-cultural

Specify Users

Early computer interfaces were difficult for people to use, limiting the situations where the systems could be deployed. Early researchers in HCI developed user-centered design as a way to improve system performance through understanding the important characteristics of system users. Norman writes, "As we expand the base of the user population, we must attend more and more to the needs and abilities of a variety of users" (Norman, 1984, p. 11). Initially, user characteristics were studied with the idea of configuring system features for each particular user need or style. However, research indicates that it is more practical to build systems that are sufficiently flexible to accommodate the anticipated range of use (for discussion of this issue see Huber, 1983).

User specification is initiated by interviewing all system users or a sampling of individuals who are representative of the *range* of system users. The interviews typically are conducted as a part of user requirements elicitation accompanying systems analysis. Users should be specified by name and job description. Knowing who will use the system and what they do in their jobs is helpful in deciding which characteristics are pertinent to the UI, is key to understanding the range of variation that can be expected for each characteristic, and can be useful in directing follow-up questions and scheduling subsequent user testing. The objectives of user specification are, first, to refine the prototype determinants document by surfacing additional important determinants and eliminating factors that are not important determinants and, second, to collect information from users about their characteristics, tasks, and environments.

Specify Tasks

As a discipline, HCI focuses on users and their relationship to the system. However, HCI research also highlights the task as an important determinant for UI development. Lewis and Rieman present a task-centered design method that "focuses on real, complete, representative tasks ... [vs.] abstract, partial task elements" (Lewis & Rieman, 1993, chap. 2). In task-centered design, UI developers interview representative users to develop task specifications through the following activities:

- collect actual tasks that the UI will be used to accomplish, ranging from simple to complex in action;

- ask *what* the user wants to do, not *how* to do it, thus avoiding the tendency to define the task too narrowly within perceived technical constraints;
- develop scenarios that encompass individual tasks as well as interactions among tasks;
- use task scenarios to highlight *what* UI features will be necessary to accomplish the tasks and to infer *how* these features will be used.

We recommend that this task-centered approach for the process of specifying tasks be conducted during user requirements elicitation. As it will rarely be possible in practice to develop and fully document more than a small number of sample tasks and scenarios, we suggest that developers attempt to be brief in documenting tasks and scenarios. The primary objective of task specification is to establish unambiguous criteria for subsequent review and testing of the UI. Secondly, written specifications help developers to understand how users intend to apply the UI in their work activities.

Specify Environments

User- and task-centered design are valuable aids to UI development in IS projects, but additional factors are important for systems that are to be used in organizational contexts. We call these general factors *environments* in recognition that they surround the user and task.

Environments and environmental considerations should be specified as UI determinants only if they place important constraints on the user, the task, or the form that the system can take. It is not productive to include environments that simply provide context for the system. For example, technology is an obvious environmental UI determinant if an organization has a legacy computer system that limits the potential range of UI designs. In this case, the computer model, operating system, and mode of user interaction should be named as environmental UI determinants.

During requirements analysis the UI developer should initially review the overall range of environmental factors that have potential to constrain. The important factors identified in this review are combined into a listing of environment-related UI determinants. It is difficult to develop a complete listing during the initial review as many factors come to light only during later stages of

systems analysis and design, often as a result of feasibility analysis. Therefore, it is important to be ready to add environment entries to the UI determinant list throughout the UI development process and to be prepared to iterate as necessary through the SALVO steps.

Use and Documentation of UI Determinants

UI determinants provide development guidelines representing the users, their tasks, and their environments. Specifications must be documented adequately to support subsequent testing of the UI against an unambiguous set of determinants. At the same time, the specification process should be sufficiently streamlined so that UI developers find it to be an interesting and useful aid in understanding system constraints and focusing their subsequent development efforts, rather than a new form of tedious paperwork (see Appendix). Documentation of UI determinants should be reviewed during the ensuing steps of SALVO and revised when the determinants change.

A: ADOPT SYSTEM STANDARDS

Adopt operating system standards for UI design. In UI development, it is imperative that developers make a concerted effort to *fail to invent* new UI designs and interaction methods, insofar as possible. In terms of productivity, this is extremely important both for developers, who must take time away from other activities to invent and implement the new feature, and for users, who must learn how to make the new UI designs work. Brown states, "Consistency is one of the most obvious human-computer interface design goals, but one that requires perhaps the most discipline in the design process" (Brown, 1988, p. 9). This statement is borne out in the vast range of UI designs that students produce in their programming projects. Although we tell ourselves that our students' penchant for garish magenta and lime green designs will diminish over time, there is always some tendency for UI developers to experiment unproductively unless their designs are anchored in documented standards.

An important part of planning is choosing the standards that will be used in development. We propose that system-level UI guides, e.g., (Microsoft Corp., 1995), provide the best standards for programmers who are not HCI specialists, and we recommend that UI developers obtain and review the appropriate guide *prior to beginning design work*. System-level UI guides enhance programming productivity by presenting current

standards, being specific to the intended computing environment, and providing relevant coding examples and guidelines. Our advice is not intended to discourage interest in general books on UI design, e.g., (Brown, 1988; Shneiderman, 1987; Thimbleby, 1990). However, generalized information is not an adequate substitute for system-level documentation at this stage of design.

L: LEVERAGE USER ABILITIES

Apply users' existing abilities to leverage the usability of the UI. Regardless of background, users bring a great number of abilities to their interaction with a system. It is critical during planning to reflect on ways these may be incorporated to increase productivity relating to the UI. *Leveraging* applies users' existing abilities and frameworks to minimize difficulties in dealing with a new situation. For example, it will be easier initially for users to fill out an on-line replacement for a paper form if it has the same layout and requires the same entries as the paper form they have experience with. Alternatively viewed, leveraging also attempts to avoid implementing features in which users have inabilities, i.e., are lacking skills. Baecker and Buxton (1987, p. 212) recommend four procedures to aid leveraging:

- build upon the users' existing set of skills
- keep the set of skills required by the system to a minimum
- use the same skill wherever possible in similar circumstances
- use feedback to effectively reinforce similar contexts and distinguish ones that are dissimilar

It is useful to consider user abilities from two distinct viewpoints. First, group norms are important for directing the baseline UI features; these can be thought of as the center of the system's flexibility to accommodate users' inabilities and special skills. Second, consideration of individual users or subgroups who vary from the norm in one way or another (e.g., novices or "power users") will provide a gauge to the amount and type of flexibility that should be incorporated into the UI (e.g., whether to incorporate "balloon help"). During creation of systems analysis documentation a leverage review should be conducted during which each determinant in the user specifications should be examined for its potential to leverage the UI. If leveraging is conducted effectively a major iteration cycle may be expected to occur between the Leverage stage and the Specify stage. The abilities

and inabilities in the determinants list that are most important to leveraging (in the developer's view) should be identified, and the determinants list should be updated to include additional factors that emerge during review.

V: VISUALIZE BEFORE CREATING

Use visualization techniques to explore UI designs without investing much time. This final planning stage promotes efficient production by using two separate visualization techniques to quickly create and refine UI designs.

Cocktail napkin visualization is performed early in the design stage, producing small, rough drawings with little attention paid to clarity or detail. These are drawn rapidly one after the other, typically in a private setting, until the developer is satisfied with the emergent design. Cocktail napkin visualization has several important characteristics:

- Drawing overcomes a natural tendency to over-rely on mental models by moving the design into a visible medium. Norman (1983) points out that mental models are frequently incomplete, they are unstable, and it is hard for people to visualize dynamic actions in their mental models. Drawing exposes problems to view.
- Fast production of the drawing on a small scale minimize both the developer's investment in the particular version and the tendency to "commit" to it. This avoids freezing the design prematurely.
- The act of producing designs in private promotes creativity by relieving the developer of concerns about external review or criticism. Although cocktail napkin visualization can be conducted in groups, many people are embarrassed about their drawing skills and will spend excessive amounts of time trying to complete a single design perfectly.

When the developer is satisfied with the cocktail napkin designs, these should be carefully annotated with any supporting information that might be forgotten with the passage of time, e.g., titles of command buttons, dynamic actions on screen, and description of graphics. The annotated designs may then be shared with, and reviewed by, other members of the development team.

Mock-up visualization is the detailed portrayal of the actual system or parts of the system. Depending on the tools at the developer's disposal, this may be created either off-line or on-line. Off-line mock-ups typically are

created entirely on paper, with commands, buttons, graphics, etc. shown in position. Each UI screen is represented by one sheet of paper for a static display, or more sheets if the screen displays dynamic information or animation. On-line mock-ups are created using a development environment suitable for rapid application development (RAD), e.g., Visual Basic, Delphi, HyperCard, or a prototyping tool. In either case, development of the mock-ups should be completed quickly and without much functionality, since it is quite likely that observation will suggest important changes to the design.

Wizard of Oz techniques (Thimbleby, 1990, p. 101) can provide support in areas where it is difficult to generate the system's built-in functionality in mock-ups (e.g., ability to print the document shown on the screen). These techniques, based on the *faux* wizard in the story of the same name, are used to augment mock-ups with a human intermediary who acts as a part of the system by describing what actions the system performs when the user interacts with the interface.

In large projects, it may be impractical to simultaneously mock-up the UI for the entire system. For these cases, Nielsen (1989) recommends developing *horizontal* and *vertical* mock-ups based on task-scenarios, such as those specified in the UI determinants. The horizontal mock-ups show the broad appearance of the application, e.g., menus, dialogs, and windows. The vertical mock-ups are designed to provide sufficient depth and detail of a particular part of the system to show how it will react in specific scenarios.

O: OBSERVE AFTER CREATING

Observe users of the UI with a think-aloud technique. The Observe stage is the test process for the SALVO method. Visualization and observation form a natural iteration cycle that has potential to decrease overall costs and completion time for the project. To gain these benefits, it is crucial to begin to observe users early during the systems design process to iteratively modify specifications and avoid nonproductive back-tracking. This is illustrated by the following report of user testing for an on-line tutorial. The UI developers found that the trouble spots they had anticipated were easily remedied, but an unforeseen problem in configuring software to run with the users' monitors required numerous user tests to correct:

No matter how many engineers we had crowded into a room to discuss what areas users were or

were not going to have trouble, we would never have hit upon this as the major problem in the application. Had we not tested, we would have had a disaster on our hands.... My experience with this and other applications and systems have proven to me beyond a shadow of a doubt that testing can save time, rather than cost time because I don't have to work on things that aren't broken. (Tognazzini, 1992, p. 89)

Early cycles of observation can highlight unexpected problems while it is still inexpensive to fix them. The relative cost to fix problems after the system is implemented has been reported to be in excess of 100 times the cost to fix the same errors discovered in the early design phases (Boehm, 1981).

We recommend UI developers employ a *think-aloud* technique for user observation that is simple to apply and analyze. In this technique, the UI developer finds representative users to try out the system under development. Typically, these would be first-time users whose actions are not biased by experiences with a previous mock-up. The following procedures for the think-aloud technique are summarized from Lewis and Rieman (1993):

1. Place the user in position to access the system and describe the task scenarios he or she is to work through. Scenarios and tasks are drawn from the task specifications incorporated into the determinants list. Description of task scenarios should focus on *what* the user is expected to accomplish, rather than *how* it should be done.
2. Explain how to think aloud: "You should say aloud what you are thinking about, concerning the system, as you work. If you quit talking I'll remind you to speak up. Go ahead and begin." Make it clear that if users have trouble it is the system's problem and not their fault.
3. As the user works, avoid volunteering information to explain the system's operation, as your purpose is to see how the user will progress *without* a human guide. Either explain in advance that help cannot be given during testing or plan ways to avoid leading the user when giving help. If the user quits talking, give a prompt to "Tell me what you're thinking."
4. Pay special attention to any areas where the user is *blocked* (can't progress without help), *backtracks*

(retraces steps due to uncertainty of how to proceed), *misappropriates* (incorrectly uses commands or tools), or accesses on-line help functions. Keep notes primarily in writing, but also consider making unobtrusive audio or video recordings with the users' permission.

Observation sessions should not be lengthy, as both users and developers tire quickly when concentrating on their roles. After each observation session is completed, notes from the session should be reviewed and summarized, paying special attention to unexpected user actions. The designer must bear in mind that the intention is to test the interface, not to validate it. Avoid the temptation during review to "gloss over" or rationalize user problems. Plan to correct observed problems, where possible, through redesigning the UI rather than changing the training methods or simply expecting users to work around problems on their own.

FIRE ANOTHER SALVO

When circumstances dictate, return to the appropriate previous step and work through the processes again. Iteration is expected to occur in the SALVO method, especially between the observation and visualization stages where iterative user testing should be applied to refine the system prior to complete construction. Coupling a streamlined methodology such as SALVO with modern RAD programming environments can take much of the onus out of iteration and quickly improve final products.

Moving from a sequential to iterative process design can make it difficult to know when pre-release development is completed. We suggest the following guidelines to assess completion. First, review documentation of UI determinants and ensure that each has been addressed satisfactorily. Second, review the UI design to ensure that it conforms to the UI development standards documentation for the operating system. Finally, address problems observed in testing a representative sample of users. The number of users to be tested should be weighed against the difficulty and expense of revising the system after full release.

DISCUSSION

In the authors' experience, students quickly acquire proficiency where the SALVO method is used to complete hands-on projects. Out-of-class practice in the method is essential, and we cannot whole-heartedly recommend SALVO, or any other UI method for that

matter, for courses that teach only concepts.

Our students have had the greatest success when SALVO accompanies projects that incorporate complete analysis, design, and implementation stages of the systems development life cycle (SDLC). Analysis naturally emphasizes the Specify, Adopt, and Leverage planning processes; design provides a reason to Visualize; and implementation creates products for users to interact with while developers Observe them. In addition, observation frequently gives developers cause for iterating their UI designs to remove unanticipated flaws. We have also had success using SALVO in less extensive projects. For example, students can learn to Visualize and Observe effectively even in implementation projects where functional development is guided by instructor-supplied requirements and students design the UI only.

LIMITATIONS

The SALVO method has several limitations. First, it is a new method that has been tested in the classroom in a limited manner to date and may benefit from continued refinement. Students find some parts, such as cocktail napkin visualization and the think-aloud method of observation, to be straightforward and immediately beneficial. Other parts have proved less simple for all students to grasp. For example, students sometimes confuse Adoption (a process of standardizing system controls) with Leveraging (a process of supporting users' existing abilities) and many have trouble giving up a "code first, plan later" mentality.

Second, the method by nature is informal and lacks statistical rigor, particularly in the O (Observe) phase. Further, the time constraints under which SALVO must be presented limits it to discussing only a small subset of

the tools that are used regularly by HCI specialists. Thus, SALVO should not be considered to substitute for specialized training in HCI in situations like commercial software development where the highest level of UI development is a critical success factor. However, as we have argued, a basic methodology for UI design is better than the *status quo* situation.

Finally, certain aspects of the method must be weighed against external factors. For example, advice to leverage users' abilities may be subordinated by the organizational goal of reengineering business processes for improved efficiency. These and similar interactions among factors are difficult to explore in a classroom context when time is limited. We propose that these limitations are offset by practical value provided to IS students who must draw initially upon their classroom training when called upon to develop UIs.

CONCLUSIONS

A substantial body of research has emerged in the study of user interface development, but the volume, diversity, and multidisciplinary nature of this literature has made it hard to apply the findings to IS pedagogy. SALVO presents a memorable, logical, and systematic method for developing UIs that can help eliminate the mystery and superstitions that often shadow this aspect of systems development. As students gain experience, it should not be necessary for them to replace the SALVO method with a different UI development methodology. Since SALVO's skeletal framework incorporates key aspects of HCI research and pedagogy, the method is suitable for fleshing-out with knowledge gained through experience and further study.

REFERENCES

- ACM SIGCHI, 1992. *ACM SIGCHI curricula for human-computer interaction*. New York: ACM.
- Baecker, R. M., & Buxton, W. A. S., 1987. *Readings in human-computer interaction: A multidisciplinary approach*. San Mateo, CA: Morgan Kaufmann Publishers.
- Boehm, B. W., 1981. *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall.
- Brown, C. M., 1988. *Human-computer interface design guidelines*. Norwood, NJ: Ablex.

- Davis, G. B., Gorgone, J. T., Couger, J. D., Feinstein, D. L., & Longenecker, H. E., Jr., 1997. *IS '97: Model curriculum and guidelines for undergraduate degree programs in information systems*. Association of Information Technology Professionals.
- Huber, G. P., 1983. Cognitive style as a basis for MIS and DSS designs: Much ado about nothing? *Management Science*, 29 (5), 567-582.
- Lewis, C., & Rieman, J., 1993. *Task-centered user interface design*. Electronic text available through anonymous FTP to ftp.cs.colorado.edu in /pub/distribs/clewis/HCI-Design-Book.
- Microsoft Corp., 1995. *The Windows® interface guidelines for software design*. Redmond, WA: Microsoft Press.
- Nielsen, J., 1989. Usability engineering at a discount. In *Proceedings of the Third International Conference on Human-Computer Interaction*, Boston, MA.
- Norman, D. A., 1983. Some observations on mental models. In Gentner, D., & Stevens, A. L. (Eds.), *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum.
- Norman, D. A., 1984. Cognitive engineering principles in the design of human-computer interfaces. In G. Salvendy (Ed.), *Human-Computer Interaction*, pp. 11-16. New York: Elsevier.
- Norman, D. A., 1986. Cognitive engineering. In D. A. Norman & S. W. Draper (Eds.), *User centered system design*, pp. 31-61. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Shneiderman, B., 1987. *Designing the user interface: Strategies for effective human-computer interaction*. Reading, MA: Addison-Wesley.
- Thimbleby, H., 1990. *User interface design*. New York: ACM Press.
- Tognazzini, B., 1992. *Tog on interface*. Reading, MA: Addison-Wesley.
- Webster, 1989. *Webster's new universal unabridged dictionary*. New York: Barnes & Noble.

APPENDIX
APPLICATION OF SALVO IN A STUDENT UI DEVELOPMENT ASSIGNMENT

Assignment Instructions

Develop a training log for an athlete. The athlete should be involved in a sport which requires constant conditioning and continual training.

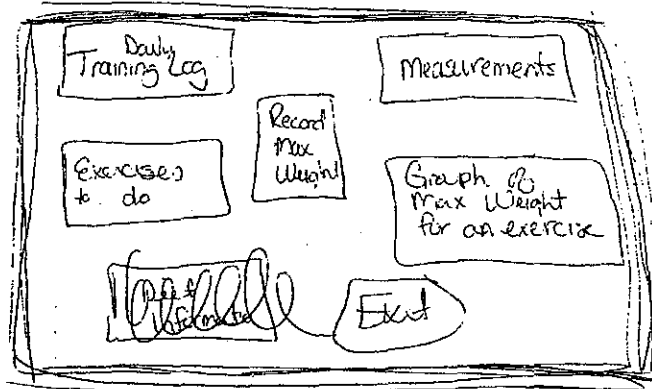
SALVO Documentation Example—Training Log for a Weightlifter

Step	Documentation
Specify Determinants	<p>User</p> <p>User Name: A. U.</p> <p>Age: 32</p> <p>Gender: Female</p> <p>Sports Activity: Weightlifting</p> <p>Computer Skills: a) currently tracks training using a computer spreadsheet; b) minimal proficiency with computer applications</p>
Task	<p>Enter new user data</p> <p>Enter lift-weight and muscle growth goals</p> <p>View daily lifting routine</p> <p>Record daily lifting data</p> <p>View time-series graph of maximum weight lifted by specific exercise</p>
Environment	<p>Technology: PC Pentium laptop computer; Windows 95 O/S; Visual Basic development environment</p>
Adopt System Standards	<p>User for reference: <i>The Windows Interface Guidelines for Software Design</i>, Microsoft Press, 1995</p>
Leverage User Abilities	<p>Application should support users with limited computer skills with easy entry form training sheet or direct entry onto laptop during training; incorporate balloon help into application</p> <p>Layout of main entry/viewing screen should give a spreadsheet view</p> <p>Weightlifting terminology should be incorporated where appropriate</p>

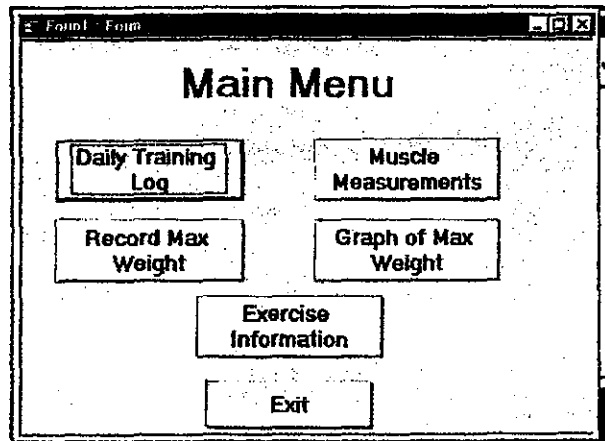
APPENDIX
(continued)

Visualization

Cocktail napkin
(sample design)



Mock-up
(sample screen)



Observation

After creating the mock-up, I had the user complete a set of tasks from the task specifications while "thinking aloud." She pointed out that it might be hard to remember the user ID and asked if she could enter a name instead. She asked if the system could hold information about exercises and stretching techniques. She also wanted to be able to search through the data rather than simply browsing.

