ED 422 923 IR 057 081

AUTHOR	Wilson, E. Vance; Connolly, James R.
TITLE	Key Procedures in User Interface Development.
PUB DATE	1997-00-00
NOTE	9p.; In: Proceedings of the International Academy for Information Management Annual Conference (12th, Atlanta, GA, December 12-14, 1997); see IR 057 067.
PUB TYPE	Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE	MF01/PC01 Plus Postage.
DESCRIPTORS	*Computer Interfaces; *Computer Software Development; *Computer System Design; Information Science Education; *Information Systems; Interaction; *Man Machine Systems; Research and Development; User Needs (Information)
IDENTIFIERS	Graphical User Interfaces

ABSTRACT

Information systems (IS) professionals are called on to produce increasingly sophisticated user interfaces as a part of software development. Although IS education includes coverage of user interface (UI) development, classroom presentation of this topic frequently is cursory and does not provide any particular methodology for the task. In this paper, a set of procedures is presented for teaching UI development with a "how to" approach, based on research and practice in the field of Human Computer Interaction (HCI). Initial application in the classroom suggests this approach is useful for focusing students' conceptual understanding of the topic without increasing required classroom time or resources when compared to traditional methods. Three tables present: key procedures in UI development, examples of UI constraints, and results of user testing in an Apple II online tutorial. (Contains 16 references.) (Author/AEF)

KEY PROCEDURES IN USER INTERFACE DEVELOPMENT

E. Vance Wilson
University of Wisconsin–Eau Claire

James R. Connolly
California State University, Chico

Information systems (IS) professionals are called on to produce increasingly sophisticated user interfaces as a part of software development. Although IS education includes coverage of user interface (UI) development, classroom presentation of this topic frequently is cursory and does not provide any particular methodology for the task. In this paper, we present a set of procedures for teaching UI development with a "how to" approach, based on research and practice in the field of Human Computer Interaction (HCI). Initial application in the classroom suggests this approach is useful for focusing students' conceptual understanding of the topic without increasing required classroom time or resources when compared to traditional methods.

INTRODUCTION

The onrush of new user interface technologies and a lack of HCI specialists in the field of IS have combined to make UI development a common activity for mainstream IS professionals. However, students in IS academic programs are not well prepared for this responsibility as they frequently receive only a cursory introduction to the topic. This is due primarily to two factors. First, the treatment of UI development and other aspects of HCI is spread thinly across the curriculum. The IS '95 curriculum proposal [7] suggests that in-depth study of the topic should be presented in both the programming language and the physical design curricula. However, UI development is not the central focus in either of these courses, and the small amount of classroom time that can be devoted to the topic in either course implicitly limits the depth of student learning.

Second, course textbooks avoid the issue of how to develop a UI. Instead, texts tend to broadly survey UI categories and their characteristics, touching on such diverse topics as command line instructions for mainframe computers, graphical user interfaces (GUIs), multimedia computing, and even virtual reality. From their texts, IS students may learn what a dialog box is and receive isolated tips such as "use command verbs clearly" and "don't display blue text on a red background." What students do not learn is a specific method for UI development that they can use in their subsequent careers in the manner that they are able to apply, for example, data flow diagramming or flowcharting techniques.

Skills in UI development will continue to be important for mainstream IS students. Since HCI specialists are relatively rare in the IS profession and UIs are a major part of current practice in event-driven programming, it is foreseeable that today's IS students will be responsible for developing UIs during their careers. Thus, there is a compelling need to overcome the limitations of course time constraints and shallow texts in teaching this topic.

This paper presents a set of procedures for UI development that we have found to be useful both in stand-alone training and as augmentation for textbook treatments of the topic. In the following sections, we discuss the background leading to the research, describe the procedures, and highlight ways they can be applied in practice.

BACKGROUND

IS programs that offer specialization in HCI require one or more courses in User Interface Design, supported by study in Cognitive Science, Human Factors of Information Systems, and additional electives in HCI [1]. We do not propose that it is desirable, or even possible, to add this volume of coursework to the mainstream IS curriculum. Instead, our approach focuses on isolating and presenting the procedures that HCI user- and task-centered research has found to be key to pedagogy and practice. 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.

Our criteria are, first, that the procedures must be sufficiently basic that students can quickly grasp the major concepts. Although basic instruction inherently lacks depth, our intention is to provide a framework to which students can add the experiential learning they presumably will gain during their careers. Second, the procedures must prescribe specific activities or ranges of activities to be performed in a systematic fashion. Third, the procedures should avoid assumption of particular technologies or other contingencies. Finally, it must be possible to present the material effectively during a single class period of 50 to 80 minutes length.

The result of our research is the list of six procedures profiled in Table 1. The procedures may be combined to portray a skeletal method of UI development that is adaptable to a wide range of programming environments and UI formats. In the following sections we describe the individual procedures and suggest ways of integrating them into standard systems analysis and design procedures.

Determine Factors That Constrain the UI

Determine the factors that are most important in constraining the form and function of the UI; these are the UI constraints. We propose three categories of factors to be central to the success of UI development within the organizational contexts that are typical of IS practice. These relate to:

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

The determination 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 would likely 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. The determination must be adequately documented so that standards to which the UI is designed and tested are unambiguous. At the same time, the documentation process should be streamlined so that UI developers can use it as an interesting and helpful means to understand system constraints and focus their subsequent development efforts, rather than just another form of tedious paperwork.

Determine Users. Early computer interfaces were difficult for most people to use, limiting the situations where the systems could be deployed. 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 [computer] user population, we must attend more and more to the needs and abilities of a variety of users" [11 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 than to try to match the system to each individual special characteristics (for discussion of this issue, see Huber, 1983).

Users should be specified by name and job description, and a set of pertinent user characteristics should be developed from these specifications (see Table 2). Knowing who will use the system and what they do is helpful in deciding which characteristics are pertinent to

TABLE 1

KEY PROCEDURES IN USER INTERFACE DEVELOPMENT

Procedure	Stage in Process	Description
Determine factors that constrain the UI	Early	Determine and list the most important factors relating to the users of the UI, the tasks that will be performed with it, and the environments where it will operate
Leverage users' skills	Early	Review listing of users' skills to identify those that will be most important to leveraging the usability of the UI
Adopt operating system standards	Early	Use the OS manufacturer's guidelines as the primary standard for UI development
Visualize prior to coding the product	Middle	Visualize the finished design in two separate stages using cocktail napkin and mock-up techniques
Observe prior to releasing the product	Middle to late	Perform user testing of mock-ups and working project components using a simple think-aloud technique
Conduct planned iteration among the procedures	Throughout	Plan ahead and budget for iterative development, especially for cycling between the visualize and observe procedures

TABLE 2

EXAMPLE LISTING OF UI CONSTRAINTS

User	Task	Environment
Demographic factors	Simple task (in spreadsheet)	Technology
Age	Enter formula to sum range A4:B7	Computers: Mix of Intel 386, 486,
Gender	Copy cell A5 to cell D8	and Pentium PCs
Height	Complex task	O/S: MSDOS ver. 5.0
Able to telecommute?	Create histogram bar chart of range A1:A28	Other environments
Salaried or hourly?	Make formula and cell	Management structure
	entries to calculate varying rates, periods,	Physical facilities
	and loan amounts	Market competition
		Industry practices
Skills		Legal and regulatory
Computer spreadsheet		Socio-cultural
programming skill		
English language		
fluency		
Data entry speed		
Personal	Scenario	
Learning style	John is creating a check register in Excel.	
Cognitive complexity	He is a proficient Lotus 1-2-3 user but has	
Visual acquity	never used Excel. He can create the register	
Color vision	by performing tasks A, B, and C.	

the UI and is key to understanding the range that can be expected for each characteristic. Once the user-related UI constraints are decided upon and documented in a simple listing, information about them can be collected as a part of user requirements elicitation accompanying systems analysis. However, if it is not possible to contact all system users, it is important to find users who are representative of the identified target group, especially in their technology skills and interests.

Determine Tasks. HCI research also suggests that task has important consequences for UI development. Lewis and Rieman present a task-centered design method that "focuses on real, complete, representative tasks ... [vs.] abstract, partial task elements" [6, chap. 2]. In task-centered design, UI developers interview representative users to develop task specifications in the following steps:

1. collect actual tasks that the UI will be used to accomplish, ranging from simple to complex in action;
2. ask what the user wants to do, not how to do it, thus avoiding the tendency to define the task too narrowly within preconceived technical constraints;
3. develop scenarios that encompass individual tasks as well as interactions among tasks; and
4. use task scenarios to highlight what UI features will be necessary to accomplish the tasks and to infer how these features will be applied in practice.

The described task-centered approach should be used for UI constraint determination as far as is practical. However, there rarely will be time in practice to both develop and fully document large numbers of sample tasks and scenarios. Thus, we suggest that developers focus their efforts on collecting a representative variety of tasks and scenarios and limit the documentation of these to simple descriptions of the type shown in Table 2. Task-related UI constraints should be collected during user requirements elicitation.

Determine 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, as is typical in IS. We call these general factors environments in recognition that they surround the user and task. Some environments that are potential UI constraints are management structure, technology, and industry practices (see Table 2). Although a given project is likely to have many related environmental factors, it is incumbent on the developer during the determination process to focus on only those that place important constraints on the UI. In many cases these environments will become evident during requirements analysis. However, important factors frequently surface as additional users, managers, and other stakeholders become acquainted with the system. Thus, UI developers should be open to the possibility of emergent environmental determinants and be prepared to evaluate and document these as they arise.

Leverage Users' Skills

Apply users' existing skills to leverage the usability of the UI. Regardless of background, users bring a great number of skills to their interaction with a system. Leveraging applies users' existing skills to minimize difficulties in dealing with the 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 form they have been using. Baecker and Buxton [2, 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; and
- use feedback to effectively reinforce similar contexts and distinguish ones that are dissimilar.

In leveraging, it is useful to consider user skills from two distinct viewpoints. First, group norms are important for directing the baseline UI features; these can be thought of as the central target of the system's flexibility to accommodate users' needs 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"). The list of user-related constraints that is developed during specification should inform the leveraging process. The skills in this list that are most important to leveraging should be identified, and the constraints list should be updated to include any additional user skills that emerge during this review.

Adopt Operating 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. This is important to productivity 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 use the feature. 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" [4, p. 9]. There is always some tendency for UI developers to experiment unproductively unless their designs are anchored in documented standards. We propose that system-level UI guides, e.g., [8], 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., [4, 13, 14]), however, generalized information is not an adequate substitute for system-level documentation.

Visualize Prior to Coding the Product

Use two types of visualization techniques to explore UI designs. We recommend using two distinct visualization techniques to quickly create and refine UI designs. Cocktail napkin visualization is performed early in the design stage, with the developer 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 overrely on mental models by moving the design into a visible medium. Norman [10] 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 serves to minimize the developer's investment in the particular version and the tendency to commit to it. Thus, fast production 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 as perfectly as possible.

When the developer is satisfied with the cocktail napkin design, the final version 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, since it is quite likely that subsequent observation will suggest important changes to the design.

Where it is difficult to provide built-in functionality in mock-ups, e.g., ability to print the document shown on the screen, this can be supported by *Wizard of Oz* techniques [14, p. 101]. 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 out a part of the system by describing what actions the system performs when the user interacts with the interface.

In large systems, it may be impractical to simultaneously mock-up the UI for the entire system. For these cases, Nielsen [9] recommends developing horizontal and vertical mock-ups based on task-scenarios such as those specified in the UI constraints. Horizontal mock-ups show the broad appearance of the application, e.g., menus, dialogs, and windows. 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.

Observe Prior to Releasing the Product

Observe users of the UI with the mock-ups you have created. Observation should be planned with the goal of decreasing the overall expense of the project. For user observation, we recommend UI developers employ a think-aloud technique that is simple to apply and analyze. In this technique, the UI developer finds representative users to "try out" the system under development. Ideally, these would be individuals who represent the intended user group and 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 [6]:

1. Place the user in position to access the system and describe the task scenario he or she is to work through. 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 fault and not theirs.
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 user's 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. Incorporating early cycles of observation can highlight unexpected problems while it is still inexpensive to fix them. In fact, the relative cost to fix errors discovered in the early design phases has been reported to be less than one-hundredth the cost of fixing the same problems after the system is implemented [3]. Plan to correct observed problems, where possible, through redesigning the UI rather than alternatives, such as changing the training methods or simply expecting users to work around problems on their own.

Conduct Planned Iteration Among Procedures

Plan ahead to iterate and work through UI development in a cyclical fashion. The idea of purely sequential, or *waterfall*, development is not

TABLE 3

USER TESTING IN AN APPLE II ON-LINE TUTORIAL (TOGNAZZINI, 1992)

User Test	Screen Display	Failure Rate	Explanation for Failures
1	Color graphic is displayed *Prompt:* "Is the picture above in color?"	Overall: 25%	Users didn't know whether monitor was black and white or the color was turned off.
2	Color graphic of words GREEN, BLUE, ORANGE, MAGENTA is displayed with each word in the named color *Prompt:* "Are the words above in color?"	Color monitor 0% B/W monitor: 0% Green monitor: 100%	Users of green-screen monitors saw a green colored graphic.
3	Same graphic as User Test 2 *Prompt:* "Are the words above in more than one color?"	Color monitor: 0% B/W monitor: 20% Green monitor: 100%	Users of non-color monitors interpreted the graphic as two colors (black and white or black and green)
4	Same graphic as User Test 2 *Prompt:* "Are the words above in several different colors?"	Color monitor: 0% B/W monitor: 20% Green monitor: 25%	Users misread question as "Are the words above ... several different colors?"
5	Same graphic as User Test 2 *Prompt:* "Do the words above appear in several different colors?"	Overall: 0%	No failures.

appropriate for UIs [6]. Regardless of the amount of planning that may go into requirements specification, unexpected problems are typical in UI development. This is illustrated by the case of Apple UI developers who, in testing an on-line tutorial, found that their anticipated trouble spots were easily remedied, but a different, unforeseen problem in configuring the software to run with the users' monitors required a surprising number of observation cycles to remedy (see Table 3). The importance of planned iteration to their project is apparent in this quote:

> 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. [15, p. 89]

Iteration should be anticipated in UI development, especially between the observation and visualization stages where iterative cycles can refine the system prior to complete construction. Coupling a streamlined documentation process, fast visualization, and inexpensive observation with modern RAD programming environments can take much of the onus out of iteration and can significantly improve final products.

Moving from a sequential to an 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 constraints and ensure that each has been addressed satisfactorily. Second, review the UI design to ensure that it conforms to the operating system standards for UI. Finally, address problems observed in testing a representative sample of users. The number of users that are observed should be weighed against the cost and difficulty of correcting problems that are discovered after the project is released.

DISCUSSION

Our presentation in the present paper has several limitations. First, the procedures have been tested in the classroom in only a limited number of occasions to date. Potentially, they will be refined and improved through future testing. Second, the presentation is informal and lacks statistical rigor, particularly in the observe procedure, where a substantial literature exists. However, the time constraints under which UI development must be covered in IS classes limits instructors to discussing only a small subset of the tools and techniques that are used regularly by HCI specialists. Thus, our presentation is not intended as a substitute for specialized training or to replace sophisticated HCI practices in situations where the highest level of UI development is a critical success factor, e.g., commercial software development. Finally, the importance of the procedures we have described must be weighed against external factors. For example, advice to leverage users' skills may be subordinated by the organizational goal of reengineering business processes to improve efficiency.

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 later in their careers. Our presentation describes a well-grounded set of procedures that can be applied to UI development in a systematic process. As students gain experience, it should not be necessary for them to replace these procedures with completely different methods. Since the procedures implement key aspects of HCI research and pedagogy, students with an interest can build upon them with knowledge that they gain through experience and further study.

REFERENCES

[1] ACM SIGCHI. *ACM SIGCHI curricula for human-computer interaction*. New York: ACM, 1992.

[2] Baecker, R. M., & Buxton, W. A. S. *Readings in human-computer interaction: A multidisciplinary approach*. San Mateo, CA: Morgan Kaufmann Publishers, 1987.

[3] Boehm, B. W. *Software engineering economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981.

[4] Brown, C. M. *Human-computer interface design guidelines*. Norwood, NJ: Ablex, 1988.

[5] Huber, G. P. "Cognitive style as a basis for IS and DSS designs: Much ado about nothing?" *Management Science*, 29 (5), 1983, 567-582.

[6] Lewis, C., & Rieman, J. *Task-centered user interface design*. Electronic text available through anonymous FTP to ftp.cs.colorado.edu in /pubs/distrib/clewis/HCI-Design-Book, 1993.

[7] Longenecker, H. E., Jr., Feinstein, D. L., Gorgone, J. T., Davis, G. B., & Couger, J. D., *IS '95: Model curriculum and guidelines for undergraduate degree programs in information systems* (draft report). School of CIS, University of South Alabama, Mobile, AL, 1995.

[8] Microsoft Corp., *The Windows® interface guidelines for software design*. Redmond, WA: Microsoft Press, 1995.

[9] Nielsen, J. "Usability engineering at a discount." In *Proceedings of the Third International Conference on Human-Computer Interaction*, Boston, MA, 1989.

[10]Norman, D. A. "Some observations on mental models." In Gentner, D., & Stevens, A. L. (Eds.), *Mental Models*. Hillsdale, NJ: Lawrence Erlbaum, 1983.

[11]Norman, D. A. "Cognitive engineering principles in the design of human-computer interfaces." In G. Salvendy (Ed.), *Human-Computer Interaction*, pp. 11-16. New York: Elsevier, 1984.

[12]Norman, D. A. "Cognitive engineering." In D. A. Norman & S. W. Draper (Eds.), *User centered system design*, pp. 31-61. Hillsdale, NJ: Lawrence Erlbaum Associates, 1986.

[13]Shneiderman, B. *Designing the user interface: Strategies for effective human-computer interaction*. Reading, MA: Addison-Wesley, 1987.

[14 Thimbleby, H. *User interface design*. New York: ACM Press, 1990.

[15]Tognazzini, B. *Tog on interface*. Reading, MA: Addison-Wesley, 1992.

[16]Webster. *Webster's new universal unabridged dictionary*. New York: Barnes & Noble, 1989.