QUID: A Quick User-interface Design Method Using Prototyping Tools

David G. Novick

Dept. of Computer Science and Engineering
Oregon Graduate Institute
19600 N.W. Von Neumann Drive
Beaverton, OR 97006-1999

Sarah A. Douglas

Dept. of Computer and Information Science
University of Oregon
Eugene, OR 97403

Abstract

Experience with prototyping tools for user interfaces indicates that just providing tools does not solve the problem of producing useful interfaces. Rapid prototyping is a design method for user interface development that emphasizes usability. However, it a bottom-up approach and thus in inherent conflict with more traditional software engineering techniques, which are top-down and specification-driven. The solution is to integrate both approaches in a single method.

The Quick User Interface Design (QUID) method is a user-centered method that is particularly useful for producing an initial prototype so that iterations of the prototype/test loop can begin from as good a design as possible. The method is a refinement of participatory design, adapted specifically for use with interface prototyping tools.

In practice, we have observed novice designers use this method quickly to produce prototypes that are useful because the method promotes systematic attention to users.

1: Introduction

The recent methodology of user-interface development has been characterized by two complementary issues, usability and productivity. Approaches for dealing with these issues have centered around rapid prototyping and interface tool or construction kits, respectively. That is, the issue of usability has been addressed by software engineering techniques that allow greater exploration of the design space or more attentiveness to users' needs, while the issue of productivity has been addressed by software systems designed to remove the tedium of hand-coding interactor objects and/or their integration into in an interface.

In this paper, we propose a methodical approach to the development of the initial prototype that addresses both usability and productivity. The method, Quick User Interface Design (QUID) method, extends our previous work in user interface construction kits. We describe QUID, placing it in the context of prior work in prototyping and construction kits, and present our experience with using the method. The fundamental idea in QUID is to begin the iterative process of designing, evaluating and refining the user interface at a position in the design space that minimizes the effort required to attain an acceptable design.

1.1: Rapid prototyping

We begin by briefly examining the origins and current practice in the use of rapid prototyping for design and implementation of user interfaces. The key point is that prototyping techniques avoid many pitfalls of more traditional approaches but in turn lead to methodological conflicts among members of the design team.

Gould and Lewis [10] recommended three principles of design for usability of computer software: early focus on users and tasks, empirical measurement, and iterative design. These principles underlie the use of rapid prototyping as a method for development of user interfaces and stand in contrast to beliefs that lead to waterfall-model design: belief in the power of reason to construct an *ab-initio* rational design, belief that design guidelines should be sufficient because they generalize our experience with human-computer interaction, and belief that good design means "getting it right the first time."

As a development method, rapid prototyping of user interfaces has matured into a reasonably well-understood program. This involves acceleration of the process of constructing a system—and particularly for user-interface components—prototypes so that the time between design and evaluation is minimized [13]. This permits multiple iterations through the design and test loop, thus leading to better attention to the needs of the user and systems that are more learnable, usable, and functional.

Rapid prototyping of user interfaces has played a significant role in participatory approaches to design (see, e.g, [2, 3]), primarily because the user/designers can quickly see the consequences of design decisions.

Prototyping is a social process that provides a means for designers' ideas to be understood and evaluated by the design team [28].

There is an apparent conflict between this sort of bottom-up method and the top-down methods based on specification. The issue we see as critical in considering the utility of rapid prototyping for user interfaces is the conflict between the methods of development best suited to usable interfaces and those best suited to achieving economical results. These differences stem in part from the characteristics of the technology and in part from the characteristics of the members of the design team. Accordingly, a basic conflict as to method may emerge among members of the design team as they strive to meet divergent goals:

> For most applications an evolutionary, whole-system, continuous prototype is a desirable choice for the human factors developer. However, revolutionary, interface-only, intermittent prototypes are much easier for the computer scientist to provide, mainly because most programming environments require programs to be complete and correct. [13, p.60].

The differences in approach among members of the design team may stem from their inherent cognitive models of user interfaces. Gillan and Breedin [8] have shown systematic differences in the cognitive models of the human computer interface between human factors specialists and software developers. The different experts have different patterns of links between interface and system-related concepts, suggesting that the experts view the components of the an interface system in terms of the functional values most directly relevant to their own methodological concerns. For example, human factors experts tend to associate the term *UIMS* with *prototyping*, while software experts tend to associate *UIMS* with *application software*. Accordingly, we conclude that a good design method for supporting multiple-role design teams should provide systematic means for specifying and communicating information from which the design is to be developed.

Hartson and Smith [13] identified the bottom-up nature of designs as a pitfall for user interfaces produced by rapid prototyping. They pointed out that because the details of the interface tend to be developed first, basic issues of system functionality may be ignored until too late in the development process. This causes upheaval in the progression toward an implementation and, we would add, frustration on the part of user/designers. One solution to this problem involves alternating phases in the development process. In case studies of interface design, Hartson and Smith observed alternating "waves" of bottom-up and top-down progressions through abstractions in design.

What is needed, then, is a more integrated approach that seeks to combine the benefits of both bottom-up and top-down design in the initial prototyping process. In section 2 of this paper, we propose such an approach.

1.2: Prototyping tools

User interface construction kits, including direct-manipulation systems, now support relatively rapid programming of interface functions, particularly for prototypes. However, the evidence suggests that these systems do not lead to increased productivity because of a systematic failure in their interaction with current design methods.

Gould and Lewis [10] observed that although their three design principles seem obvious and have been recommended since the 1970's, they are not usually employed in system design. One might have surmised that, from the perspective of 1985, the failure to use the prototyping method was a consequence of a lack of prototyping tools: the designs could not be prototyped rapidly enough for the method to be effective. Thus, in considering the same question in 1988, Wilson and Rosenberg [28]suggested that the stumbling block is the unavailability of prototyping tools:

> We might ask, 'If prototyping is so great, why is it so seldom used?' Neither computer programmers nor designers seem to be formally trained in the techniques of prototyping. The tools that exist are usually proprietary or require expensive workstations. Surprisingly, few prototyping tools are available for the ubiquitous 'personal computer.' [28, p.873].

In the three years since Wilson and Rosenberg assessed the situation, some parts of the answer have become clearer. More tools are available, even on personal computers; recent commercial tools include HyperCard, SuperCard, Prototyper, the NeXT Interface Builder, and HPVUE. The problem seems to arise from some factor other than tools.

Grudin [12] suggested that the failure of software developers to adopt the Gould and Lewis approach can be attributed to organizational forces. The importance of the user interface to other organizational units, such as preparers of documentation, leads to discouraging the use of an iterative approach: when an interface is produced—even the first prototype—there is strong pressure to "sign off" on it in order to avoid instability in the development process as a whole.

Our own experience with prototyping tools for user interfaces indicates that just providing tools does not solve the problem of producing useful interfaces. It is certainly true that prototyping tools relieve the programmer of the onerous burden of coding the mechanics of interactor

objects, just as interface toolkits relieved the burden of having to code interactor objects using graphics primitives. (Interactor objects are the components of the user interface that present and collect communication to and from the user; interactor objects typically include text fields, buttons and menus.) The problem remains, however, that a rapidly produced interface may place the prototype in a non-advantageous location in the design space. Addressing the issue of user interfaces specifically, Wilson and Rosenberg [28, p. 873] concluded that

> No matter how powerful the tools used for rapid prototyping, the job of designing a human-computer dialogue for a significant size project will always remain complex. A lot must be articulated between user and computer just to perform a simple task. It is important to remember that the 'tool' only speeds up the process, it does not ensure a high quality design solution.

One approach to overcoming this problem has been to change the technology. In the PICTIVE design technique [18], users manipulated physical representations of interface design objects. Sharing of design concepts and rationales is achieved via video recordings of the users' actions and interactions. The developer of PICTIVE found this approach to be particularly successful in originating new designs. The major problems of the technique that he observed basically involved the video technology: the records were too informal and too extensive. Of course, the technology of plastic and video does not create a working interface. Moreover, other research suggests the importance of providing "vertical" prototypes (i.e., usefully functional, even if only for a subset of the application domain) for user evaluation [11].

We addressed the technical aspects of prototyping with our research into the QUICK, the Quick User Interface Construction Kit [5, 6]. QUICK was a Macintosh-based system intended for non-programmers. In developing QUICK, we stressed that productivity gains from prototyping tools must be understood by considering the programming language and the programming environment as separate factors. We approached the problem of the programming language by offering a system with mid-level abstractions—interface objects with powerful, general functionality—that could be manipulated directly in the programming environment. The objects were prototype-based rather than class-based.

In using QUICK to prototype interfaces, we found:

- Although the set of application functions was limited (e.g., no arithmetic), expressive interface prototypes could be developed quickly.
- Complex objects could be rapidly created, duplicated, and modified.
- Interfaces had to planned; creation of interfaces using a "free-form" approach was time-consuming and frustrating.

Our results with QUICK indicated that there was a strong need for planning in interface design and development. It's not simply that the designer needs a clear notion of the functions of the interactor objects; we explored examples of interface creation in cases where a complete, working system had already been developed on another platform. Rather, it was the process of creation that needed the planning. For example, the ease of creation and modification of interface interactor objects was both exhilarating and frustrating. Typically, the designer would create some basic object, perform modifications that specialized the objects, and then would realize that the basic objects should have been different in some way. This meant either changing each of the objects or abandoning the specialization. Garnet [19] partially addresses the specific problem of delayed modification through default inheritance of values of the prototype. In Garnet, objects are copies of a prototype rather than instances of a class; however the copies will reflect subsequent changes to the prototype. Nevertheless, the general problem of a lack of a systematic approach to the development of the initial prototype remains unsolved.

In short, what is needed is a design method that accounts for and facilitates effective use of user interface prototyping tools. Perlman [21] contrasts user interface development tools *versus* methods. A better approach, we believe, is tools *plus* methods. To reach the goal of a synthesis of tools and methods, we developed a process of rapid prototyping designed to take advantage of the features of user interface construction kits such as QUICK and HyperCard.

2: An introduction to QUID

The Quick User Interface Design method (QUID) is a user-centered method that is particularly useful for producing an initial prototype so that iterations of the prototype/test loop can begin from as good a design as possible. The method is a refinement of participatory design, adapted specifically for use with interface prototyping tools. The central idea is to combine the advantages of prototyping as a method with the characteristics of user-interface construction kits so that both usability and productivity are enhanced.

In developing QUID, we took a pragmatic approach to defining usability. That is, usability is an empirically determinable quality, the attributes of which can be assessed via observation of indices of satisfaction by users. Thus, for example, one could assess usability by looking at

these measures—some objective, others subjective—of a system's user interface:

a. Overall functionality
b. Learnability (i.e., "intuitive" ease of use)
c. Robustness (i.e., protection against and recovery from failures)
d. Quality of output
e. Time to perform tasks
f. Enjoyableness
g. Overall acceptability

Accordingly, productivity could be measured in terms of the number of person-hours spent by the design team in reaching an acceptable level of usability. To attain these goals, then, we refined QUID from the earlier prototyping methods, taking particular account of the characteristics of currently available prototyping tools.

QUID's contribution to the methodology of user-interface development is to integrate (1) prototyping for usability and (2) specification for productivity in a way that addresses the conflict between top-down and bottom-up methods.

2.1 Task/Action analysis in QUID

To integrate the results of functional analysis into a prototype user interface, one needs a systematic approach to providing users with the means to express their actions in the domain of the program. While guidelines and standards have been traditionally used in interface development to associate actions with meaning, recent research suggests that guidelines are not very useful [25], particularly for design teams with few HCI experts [14]. In QUID, then, designers apply a task/action analysis to make explicit what the user does and how they do it.

The task/action analysis in QUID is a simplification and refinement of traditional task analysis; our intention is to make the process a feasible one in order to overcome the blocks to the use of prototyping observed by Lewis and Gould and of prototyping tools observed by Wilson and Rosenberg. Task/action analysis involves a kind of action language as perceived by the user; that is, how users can express their goals through the interactive system [20].

There are a variety of task analytic techniques because this approach has application in a wide number of fields. In the general case, a task is a meaningful unit of work performance. According to Phillips, et al. [22], the focus of users as members of the design team is on explaining the requirements and their interactions; task analysis is an appropriate method for doing this. As a user interface design method for conceptual design, they propose defining (a) the display objects, (b) their associated properties, and (c) the operations on the objects, their properties and relationships. In screen design, the point of task analysis is to determine what items of information the user needs at every step in the task, and what items the user needs to provide to the system [27]. The system of task/action analysis used in QUID derives from the GOMS model developed by Card, Moran and Newell [4]. QUID's model is like that of Siochi & Hartson's User Action Notation for representation of tasks in user interfaces [23]. However, QUID's model was simplified and generalized to include graphical user interfaces in addition to text-based interfaces.

2.2: Overview of the method

In QUID, the user interface design team follows modern models of team-based approaches to software development. Much of the process is akin to knowledge engineering: the knowledge of the tasks to be performed is a kind of expertise to be elicited from the users. Accordingly, we base our the makeup of our design team on successful systems for knowledge elicitation (see, e.g., [1]). The cast of characters should include representatives of the parties interested in the outcome of the development process:

- Client: The person (or a representative of the organization) who has authority for the project.
- User: Actual users of the current and proposed systems. If the product is a mass-market product, representative users should be identified.
- Interface designer: Typically a human factors specialist or computer scientist with particular expertise in human-computer interaction.
- Programmer: Software engineers who will be building and maintaining the system.

QUID stresses participatory design. Recent research suggests that the interface designer should also be a part of the team rather than a consultant [16]. Certainly the users on the design team should be actual users rather than supervisors or other team members who are role-playing.

QUID includes the essential of elements of Gould and Lewis's method, but adds a critical top-down component as a way of providing a more systematic means of creating the initial design, thus positioning the initial design in a good location in the design space. Accordingly, the four key themes in QUID are

1. Focus on the user
2. Top-down analysis
3. Early and continuous testing
4. Iterative process

As QUID's principal focus—and its chief difference from earlier methods—is development of the initial prototype, in this paper we concentrate on this phase of the method. In QUID, then, the process of design includes a

functional specification, an interface specification, and a prototype implementation. We now discuss each of these steps in turn.

2.3: Functional specification

The functional specification of the interface involves characterizing the user population, studying the existing system, and defining the primary functions of the new system (seen as including both machines and humans).

2.3.1: Characterizing the user population: Typically, the client will have identified particular users or a class of users for whom the new system is intended. If for particular users, the process of characterization is a straightforward one; the design team can directly apply the kinds of user-focusing techniques described by Gould [9]. In the general case, the design team should address the following questions:

Who are the users?

Are they going to become true experts or just casual users or one-time only users?

What familiarity do they already have with answering machines and telephones?

What kinds of things are they likely to want the system to do?

How much time will they have to learn the system?

How much time are they willing to spend looking things up in a manual?

How much time do they have to use the system?

With an explicit profile of the users, the designers can judge the suitability of the design during the initial stages of development rather than simply hoping that the initial design will be useful.

2.3.2: Study the existing system: The second phase of the functional specification is to study the existing system. Often the existing system is not computer-based; nevertheless, the functions that the existing system provides to its users can be identified and analyzed. In this process, it is the actual system rather than a description of the system that should be analyzed.

The analysis of the existing system should address these questions:

What are the most useful functions of the existing system and why?

What functions, if any, seem unneeded and why?

What problems do you find in the existing design?

What is good in the design of the existing system?

2.3.3: Define the primary functions of the system: The third phase of QUID's functional specification is to define the system's basic functionality.

We have found that it is helpful to focus on the most central, characteristic functions rather than to try to define peripheral functions. Functions that will be merely helpful can and should be added in later iterations of the design. Indeed, it may turn out that a clear and direct system produced by basic functional specification does not need the ancillary functions that the designers had anticipated. If these functions are in fact needed, early usability tests will disclose this. Functions relating to on-line assistance should not be considered ancillary however. Postponing development of on-line assistance until later iterations is one of the chief causes of inadequate help systems; assistance should be implemented as a central function of the system [7]. Thus, the key questions to ask are

What are the most necessary functions? Why?

What other functions would be nice to have? Why?

What functions should an on-line help system support?

In answering these questions, the design team should take into account the following factors:

- What does the user want the system going to do? The system will be a process, so an appropriate way to think about its characteristics is in terms of its dynamic functions.
- What "products" will it produce? What the user expects from the process is some useful result, so the system should be defined functionally in terms of its outputs, most broadly considered.
- What are the constraints? The system consists of both computational and human elements, and each is subject to its peculiar constraints. For the computational elements, the constraints include the complexity of the application's computation and the characteristics of available input and output devices. For the human users, the constraints include psycho-motor attributes, cognitive capacity, and extent of knowledge.

2.4: Interface specification

With the functional specification completed, the next major step in QUID is the interface specification, in which the system's domain functions are expressed as computational functions. Thus the interface specification includes refining the system's functions into core functions for implementation, translating the functions into a task-based design of the interaction, and then defining the presentation of the interactor objects that constitute the interface. The key point of this process involves a task/action analysis of the system: for each primitive task, the interface developer identifies the function the user seeks to execute, the user's

action needed to achieve this, and the response produced by the interface's display. These factors help to determine systematically the presentation and responses of interface interactor objects.

2.4.1: Define core functions: The process of interface specification begins with refinement and reformulation of the system's high-level primary functions into lower-level core functions for implementation. This process has three components: reconciliation of constraints, decomposition, and allocation of responsibilities.

In reconciling constraints, the designers determine which of the system's possible functions can be achieved. Factors to be considered include the constraints identified in the functional specification, the particulars of the target platform and environment of use, and the resources available to the design team.

The second step is a top-down decomposition of the achievable functions. The basic idea is to decompose each high-level function into functional primitives available in the prototyping environment.

Third, the system's component functions should be allocated between user and machine in a way that takes best advantage of the characteristics. Factors to be considered include:

- *Repetitive elements*. If some functions are repetitive or ministerial, they ought generally to be assigned to the machine. For example, searching text for a string would be a good function to assign to the machine.
- *Judgment calls*. If other functions require the exercise of judgment, they are likely candidates for assignment to the user. For example, determining if a sentence has a clear meaning would be a good function to assign to the human.
- *Errors and error recovery*. The characteristics of the anticipated errors should suggest a strategy for allocation.
- *Speed*. Again, the particulars of the function and the abilities of the user and the machine need to be evaluated on an individual basis. For example, arithmetic division would likely be performed by the machine and identification of pictures of faces would likely be performed by the human.

2.4.2: Translate functions into design specification: After the core functions have been identified and allocated, the functions assigned to the computer need to be translated into a specification for the design of the user interface. The basic idea is to express functions in terms of actions that users can take, thus the key question for the design team to answer is "How will the users use the system's functions?"

Using QUID's task/action analysis scheme, the specification is expressed as a comparatively simple triple of function, action, and display response. Thus to define task specification, the design team defines the user task method for each core function. The function is the expressed as a unit task, the action as an expression by the user via the interface, and the display response as feedback from the system to the user. The display response normally includes the consequences of the application's processing; the design team needs to bear in mind that absent a display response, the effect to the user of even the most effective computation is that nothing has occurred. We express this analysis in the following formula:

Goal + Action on Object -> Display Response

The objects that compose the interface can both exhibit responses and collect input from the user. The effects of the user's actions on the objects constitute an *interactor object specification*. This specification should be directly implementable in the prototyping environment available to the design team. In other words, it is highly unlikely that there is a primitive operation in the prototyping environment that expresses any of the high-level functions the system is intended to perform. If one of these component functions can be realized directly by a primitive operation available in the prototyping environment, the decomposition is complete for that function. Otherwise, the process continues recursively on the remaining complex, intermediate functions until all functions are expressed as design primitives of the prototyping environment.

The palette of user actions provided by the prototyping environment constitutes a bound on the design. Thus if one is prototyping in QUICK, a possible user action might involve dragging an object on the screen; if one is prototyping in HyperCard, such an action would not be possible. If the design team is using HyperCard, the possible user actions would be

mouse up
    button
    card

type
    field

scroll
    field

mouse in XY
    picture

Possible display responses in HyperCard would include:

show / hide
    picture
    field
    button

play
    sound

visual effects
    (on go to card)

Expressing the interface specification using the task/action method thus involves generation of triples of functional goals, means for users to achieve the goals, and the system's feedback to the user.

2.4.3: Define presentation of display objects: The final step of the interface specification process involves defining the layout and graphical characteristics of the interactor objects. Up to this point, the screens and interactor objects have existed as abstractions; the design team is now ready to make them more concrete because the key functional aspects of the interface have already been made explicit. The alternative, bottom-up approach, lets the determination of key functions of the initial design fall to the more haphazard consequences of choice of layout and graphical characteristics of the interface objects. This difference explains why QUID's approach is likely to lead to a better starting point in the design space for the iterative process of prototype development.

The spatial layout of the interactor objects again proceeds via top-down refinement. For example, overall contexts are assigned to windows and sub-contexts assigned to sub-windows. In making these choices, performance issues become important: the physical ability of the platform to express the design must be taken into account. A similar balance of factors must be considered in choosing the graphic characteristics of the interactor objects. Here, human factors research provides assistance in deciding on attributes such as shape and colors (see, e.g.,[24]). Metaphor and analogy also provide useful design-organizing principles [17].

2.5: Interface implementation and evaluation

With the specification, the initial prototype can be implemented in a reasonably straightforward manner using systems such as Garnet, QUICK, HyperCard, or HP-VUE. Testing of the initial prototype can then be conducted.

Prototype implementation need not be complete, or even computer-based. While QUID emphasizes systematic development of an initial prototype for implementation with computer-based rapid prototyping tools, other researchers in the field of iterative, participatory design have achieved success with low-tech methods such as mock-ups and storyboarding (see, e.g., [15]). There is, in fact, a spectrum of prototypes extending from plastic shapes on a table [18], through storyboarding, mock-ups, Wizard-of-Oz techniques, and functional software prototypes.

At this point, the design team's main task is to build the display objects and their responses. The prototype should include the design's spatial layout, graphics, text, and other details. Implementors can stub system functions where necessary but should try to give the user an indication of what function would have been executed. In HyperCard, showing a field or going to a card (and then hiding or returning) are typical ways of doing this. For display responses that involve on-line assistance to the user, the actual text of the specified system should be displayed to the extent possible. The initial prototype should emphasize coverage of the most significant features of the design. Storyboarding, metaphors, and RTN specifications can be used in order to cover display and control aspects of the design.

In testing the initial prototype developed via QUID, we suggest using protocol analysis of target users performing real tasks. A complete review of evaluation techniques is beyond the scope of this paper. We would like, though, to stress the factors that we believe are key to successful use of the techniques in the context of the QUID method. In producing the protocol, we prefer the "constructive interaction" approach, which involves a problem-solving conversation between two users, rather than the "think-aloud" approach, which may distort cognitive loads and lead to misleading introspection. The tasks presented to the users should encompass the key functions, present a range of actions, and vary in difficulty. The interaction should be videotaped for later analysis.

In analyzing the users' interaction with the prototype, QUID again uses the task/action analysis as a central part of the process. The design team reviews the testing protocol, identifying and describing in detail incidents of interaction difficulties in the following categories:

(a) Interpreting the meanings and functions of the interactor objects

(b) Formulating tasks (matching actions to goals)

(c) Knowing what to do next

(d) Misunderstanding feedback

(e) Getting help

(f) Other

For each problem the users encountered in performing the tasks, the design team recommends changes to make the system more useful or learnable. Indeed, the same basic analytic methods applied to the existing system can now be brought to bear on the new system: What are the most useful functions of the system and why? What functions, if any, seem unneeded and why? What is good in the design of the system? Given answers to these questions, the cycle of analyze-specify-implement-and-test can begin its second iteration, moving closer to the best possible design.

3: Example

To make clear the use of QUID, we illustrate key points of the method using the design of computer-based telephone answering machine as an example.

3.1: Functional specification

In studying the existing system, the design team should see what an actual answering machine really does, as opposed to reading the machine's instructions or manual. In the absence of experience with the real system, judgments of utility of functions will be significantly less accurate. When the team then defines the primary functions of the system, in the case of the telephone answering machine the core functions would likely include dialing and answering telephone calls, and might include setting a message that incoming callers would hear. The system's products turn out mainly to be actions: outgoing calls should be placed and incoming calls should be signaled and answered.

3.2: Interface specification

In the interface specification, the system's domain functions are expressed as computational functions. A significant aspect of this process is identifying what aspects of the task are computational and what are best-suited to humans. Thus, in allocating responsibility for functions between human and machine, for the telephone answering machine, looking up phone numbers in a directory would thus be an appropriate function for the computer, because it calls for repetitive, speedy work. Conversely, choosing which number to call should normally be the responsibility of the user because it requires the exercise of judgment. Similarly, identifying a null outgoing message would be appropriate for the machine; identifying that the wrong number had been reached would be appropriate for the user. Clearly, generation of the dialing tones would be a good function for allocation to the computer.

Having identified and allocated functions, the designers of the telephone answering system would then apply task/action analysis to develop actions for the user that express their task goals. For example, one of the task/action triples might express the task, action, and display response in the case of the human's using the system to dial a telephone number:

Function: dial a telephone number

Action: mouse up PhoneButton

Display response: change PhoneButton icon, play PhoneTones

Defining the presentation of the display objects for the telephone answering system would then involve graphic representation and placement of the interactor objects. A typical display might include a text field for a number to be dialed.

3.3: Implementation and evaluation

Having implemented a prototype of the specified answering machine system in QUICK or HyperCard, evaluation of the prototype should be based on a range of tasks typical to the application. For example, one might ask the users of the telephone answering machine system to perform the following tasks:

1. Listen to new messages
2. Record an outgoing message
3. Call a relative
4. Look up a friend's telephone number

These tasks were chosen because they represent classes of goals at different levels of specificity and function. They are intended both to assure usability and to challenge the designers to be imaginative.

4: Discussion

Having described QUID, we now look at our experience with the method in practice and the implications of using QUID for developing usable and efficiently produced user interfaces.

4.1: Empirical evaluation of QUID

In practice, we have observed novice designers use the QUID method quickly to produce prototypes that are useful. We tested QUID using a method similar to that of Thakkar, et al. [26]. Our design teams consisted of members of a class in interface design and evaluation. There were 15 designers working in teams of two or three. The designers were typically software professionals with considerable experience with graphical user interfaces. The teams were given an actual telephone answering machine and asked to build a user interface for a similar computer-based system using HyperCard.

All six teams produced goal-oriented interfaces, rather than the function-oriented interfaces typified by VCR controls. They were able to do so despite little or no prior knowledge of HyperCard. In three days, the teams achieved two iterations of their prototypes, including task/action analysis, specification, implementation and evaluation.

The prototypes varied mainly in terms of completeness. Not surprisingly, prototypes developed by three-member teams appeared more complete and polished than those

developed by two-member teams. All prototypes had appropriate levels of system display responses, as determined by the evaluations and our post-course judgment. There was general agreement among the prototypes as to key tasks but user actions for these tasks varied from team to team. Accordingly, using the criteria of Thakkar, et al., we conclude that the QUID method met its goals for both usability and productivity.

There were significant differences, however, in the circumstances, methods, and results of the QUID experience and that of Thakkar, et. al, in their evaluation of the NeXT Interface Builder (NeXT IB). In the NeXT IB evaluation, the designers had the underlying program functionality already prepared for them; all they had to do was to build the user interface. In the QUID evaluation, no prior answering machine application program was available to the designers, so they had to program whatever functions they needed in HyperTalk. The method of evaluation differed as well. In the NeXT IB evaluation, the designers rated their own interfaces. In the QUID evaluation, the rating of interfaces was conducted by non-team designers. In both cases, subjective measures of usability were collected, but comparisons between these numbers is not meaningful. In the QUID evaluation, use of the QUID method included operational testing by persons who were not members of the design team, but the results of this kind of usability testing are of the "here's something to fix" variety rather than a numerical value somehow indicating goodness.

A comparison of the interfaces produced using the NeXT IB alone and using HyperCard with QUID is difficult because of the different circumstances of the evaluations. However, review of the graphics of the designs produced by the design teams suggests the following preliminary conclusions. First, the interfaces produced with the NeXT IB *look* more polished, largely because the range of design expression is limited by the construction environment to interaction objects with a consistent "NeXT look." The HyperCard interfaces have a much greater variety in their expression, reflecting the plasticity of HyperCard as a tool. Second, the NeXT IB interfaces, based on an underlying application and without using user-focused, top-down methods, appear to express the computational functions of the underlying application quite directly; the buttons and fields correspond to functions and fields of the program. In the systems developed through QUID, the interfaces appear to express task-based functional goals, and then provide means to accomplish these. In making these comments, though, we want to make clear that:

(1) We could only evaluate the Next IB interfaces on the basis of the pictures and descriptions of the interfaces published by Thakkar, et al.

(2) The study performed by Thakkar, et al., was sound and needed research; we believe that their comments about the NeXT IB are well-founded.

(3) The NeXT IB is not a prototyping environment; it is basically a construction kit particularly suited to certain kinds of applications. However, we believe that use of QUID in conjunction with the NeXT IB could produce significantly more usable interfaces.

We have replicated our results with QUID in additional classes of undergraduate and graduate students in computer science.

4.2: QUID in the development context

In discussing obstacles to participatory design in large product development organizations, Grudin [12] identified several challenges to productive involvement of users in the design process. In some cases, the use of the QUID method should ameliorate some of the systematic problems Grudin observed. Foremost among these is the late involvement of users and HCI experts. In traditional software engineering methods, a relatively sharp line is drawn between the application and its interface; this leads to defining the tasks with too little consideration of user expression. And at the same time, as Grudin stressed, organizational forces cause too-early stabilization of the interface because organizational units dealing with issues like documentation and training depend on it. These forces, taken together, tend to squeeze interface development into a very small window.

The use of QUID addresses this problem by requiring that systems be analyzed from the beginning in terms of users' requirements and expression. That is, what the system is to do depends on analysis of user tasks in terms of actions within the interface. Moreover, we expect that the use of QUID would help to overcome challenges to benefiting from user contact and challenges to obtaining feedback from users. Grudin noted that:

- Developers typically either generalize from a limited number of contacts or get bogged down with disagreements among users;
- Design recommendations may not be incorporated in the product;
- Feedback from users is usually gathered haphazardly; little actually gets to developers, who are rarely aware of users' pain; and
- Developers neglect on-line help systems.

Participation by users in the design process, use of iterative prototyping, early and continuous testing, and incorporation of on-line assistance as a central function of the system would be systematic responses to these problems. Most important, the problem of the shrinking window for interface development can be addressed via the use of a

top-down, task-analytic specification technique, specifying the "application" and the "interface" as a single system, and incorporating user involvement from the earliest stages of the design process. These are precisely the significant features of the QUID method for user interface development. We thus expect that use of QUID in large software development organizations should reflect a positive outcome similar to that in our trial evaluations.

Acknowledgments

The authors wish to thank one of the anonymous reviewers, who provided extensive, helpful comments and suggestions for improving this paper.

References

[1] Bell, J., & Hardiman, R. (1989). The third role—the naturalistic knowledge engineer. In Diaper, D. (ed.), Knowledge elicitation. Chichester: Ellis Horwood.

[2] Bannon, L. (1991). From human factors to human actors: the role of psychology and human-computer interaction studies in system design. In Greenbaum, J., & Kyng, M. (eds.), Design at work: cooperative design of computer systems. Hillsdale, N.J.: Erlbaum, 25-44.

[3] Bodker, K. (1989). Analysis and design of computer systems supporting complex administrative work processes. Office: Technology & People, 4(1), 75-89.

[4] Card, S., Moran, T., & Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum.

[5] Douglas, S., Doerry, E., & Novick, D. (1990). QUICK: A user-interface design kit for non-programmers. Proceedings of the Third Annual Symposium on User Interface Software and Technology (UIST90), Snowbird, UT, October, 1990.

[6] Douglas, S., Doerry, E., & Novick, D. (1991). Splitting the difference: Exploring the middle ground in user interface design. Proceedings of the 24th Annual Hawaii International Conference on System Sciences, January, 1991, 2, 468-477.

[7] Elkerton, J. (1991). Online aiding for human computer interfaces. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 345-364.

[8] Gillan, D., & Breedin, S. (1990). Designers' models of the human-computer interface. Human Factors in Computing Systems (CHI '90), 391-398.

[9] Gould, J. (1991). How to design usable systems. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 757-789.

[10] Gould, J., & Lewis, C. (1985). Designing for usability: key principles and what designers think. CACM, 28(3), 300-311.

[11] Gronbaek, K. (1989). Rapid prototyping with fourth generation systems—an empirical study. Office: Technology and People, 5(2), 105-125.

[12] Grudin, J. (1990). Obstacles to participatory design in large product development organizations. CPSR Conference on Participatory Design, Seattle, WA,

[13] Hartson, H. R, & Smith, E. C. (1991). Rapid prototyping in human-computer interface development. Interacting with Computers 3(1), 51-91.

[14] Jeffries, R., Miller, J., Wharton, C., Uyeda, K (1991). User interface evaluation in the real world: A comparison of four techniques. Human Factors in Computing Systems (CHI '91), 119-124.

[15] Kyng, M. (1989). Designing for a dollar a day. Office: Technology and people, 4(2), 157-170.

[16] Lundell, J., & Notess, M. (1991). Human factors in software development: Models, techniques, and outcomes. Human Factors in Computing Systems (CHI '91), 145-151.

[17] MacLean, A., Bellotti, V., Young, R., & Moran, T. (1991). Reaching through analogy: A design rationale perspective on roles of analogy. Human Factors in Computing Systems (CHI '91), 167-172.

[18] Muller, M. (1991). PICTIVE, an exploration in participatory design. Human Factors in Computing Systems (CHI '91), 225-231.

[19] Myers, B., Guise, D., Dannenberg, R., Vander Zanden, B., Kosbie, D., Pervin, S., Mickish, A., & Marchal, P. (1990). Comprehensive Support for Graphical, Highly-Interactive User Interfaces: The Garnet User Interface Development Environment. IEEE Computer, 23(11), 71-85.

[20] Payne, S., & Green, T. (1986). Task-action grammars: A model of the mental representation of task languages. Human-Computer Interaction, 2(2), 93-133.

[21] Perlman, G. (1991). Software tools for interface development. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 819-833.

[22] Phillips, M., Bashinski, H., Ammerman, H., Fligg, C. Jr. (1991). A task analytic approach to dialogue design. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 835-857.

[23] Siochi, A., & Hartson, H. (1989). Task-oriented representation of asynchronous user interfaces. Human Factors in Computing Systems (CHI '89), 183-188.

[24] Snyder, H. (1991). Image quality. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 437-474.

[25] Tetzlaff, L. & Schwartz, D. (1991). The user of guidelines in interface design. Human Factors in Computing Systems (CHI '91), 329-333.

[26] Thakkar, U., Perlman, G., & Miller, D. (1990). Evaluation of the NeXT Interface Builder for prototyping a smart telephone." SIGCHI Bulletin, 21(3), 80-85.

[27] Tullis, T. (1991). Screen design. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 377-411.

[28] Wilson, J., & Rosenberg, D. Rapid prototyping for user interface design. In Helander, M. (ed.), Handbook of Human-Computer Interaction (2d edition). Amsterdam: Elsevier Science Publishers, 859-875.