Survey on User Interface Programming

Brad A. Myers and Mary Beth Rosson*

February 1992
CMU-CS-92-113

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

To appear in *Proceedings SIGCHI'92: Human Factors in Computing Systems*.
Monterrey, CA, May 3-7, 1992.
Also reprinted as IBM Research Report RC17624.

Abstract

This paper reports on the results of a survey of user interface programming. The survey was widely distributed, and we received 74 responses. The results show that in today's applications, an average of 48% of the code is devoted to the user interface portion. The average time spent on the user interface portion is 45% during the design phase, 50% during the implementation phase, and 37% during the maintenance phase. 34% of the systems were implemented using a toolkit, 27% used a UIMS, 14% used an interface builder, and 26% used no tools. The projects using only toolkits spent the largest percentage of the time and code on the user interface (around 60%) compared to around 45% for those with no tools. This appears to be because the toolkit systems had more sophisticated user interfaces. The projects using UIMSs or interface builders spent the least percent of time and code on the user interface (around 41%) suggesting that these tools are effective. In general, people were happy with the tools they used, especially the graphical interface builders. The most common problems people reported when developing a user interface included getting users' requirements, writing help text, achieving consistency, learning how to use the tools, getting acceptable performance, and communicating among various parts of the program.

This research was sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U. S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

*User Interface Institute, IBM T.J.Watson Research Center, P.O.Box 704, Yorktown Heights, NY 10598

Keywords: User Interface Software, Surveys, Toolkits, User Interface Tools, User Interface Management Systems.

1. Introduction

We were tired of seeing references to papers from 1978 for data about how much of the time and code in applications is devoted to the user interface. Surely with modern window managers, toolkits, interface builders and UIMSs, the data have changed! Therefore, we decided to conduct a new survey to determine what user interface programming is like today. This paper reports on the results of that survey.

These results will be useful for a number of purposes. First, they will help user interface developers demonstrate to their managers that, in fact, most projects spend significant time and resources on designing and programming the user interface portion. Indeed, the numbers reported here might be used by managers to predict the type and amount of resources to be directed toward user interface development. Second, the data clearly show that most projects are using user interface development tools, and that these tools are generally effective and of significant help to the projects. Third, the results can be used to support proposals to research and develop new user interface tools and techniques, and the survey reports on some specific problems and recommendations for new tools. Some of the questions on the survey investigated how the various projects were organized, the process used to develop the user interface, and what tools were used. Therefore, the survey provides a snapshot of how user interface design and implementation is performed today.

Clearly, user interfaces for programs have increased in sophistication, with the use of direct manipulation and WYSIWYG styles, mice, window managers, etc. This, in turn, has made the programming task more difficult. However, tools to help with user interface software have also become more sophisticated and helpful. The data collected tends to suggest that interface builders and UIMSs are helping to decrease the programming task.

2. Related Work

There have been very few surveys of user interface software. The ones that people usually reference are quite outdated and inconclusive. For example, an IBM study found that the user interface portion of the code was between 29% and 88% [Sutton 78]. In artificial intelligence applications, an informal poll found it was about 50% of the code [Fox 86], which is similar to the results of one AI project which reported 40% [Mittal 86].

A recent paper discusses a number of reasons why user interface software is *inherently* more difficult to create than other kinds of software, and argues that we should not expect this problem to be ‘‘solved’’ [Myers 92]. These reasons include: that iterative design is necessary which makes using software engineering techniques more difficult, that multiprocessing is required to deal with asynchronous events from the user and window system, that the performance of the resulting interface must be fast enough to keep up with users, that there is an absolute requirement for robustness so the interface never crashes, and that the tools for developing user interface software can be very difficult to use.

3. User Interface Tools

To make user interfaces easier to program, many different kinds of tools have been created. These include window systems, toolkits, interface builders, and user interface management systems (UIMSs). Comprehensive definitions and surveys of these tools can be found in many places [Hartson 89, Myers 92].

A *window system* is a software package that divides the computer screen into different areas for different contexts. Although a more common term is *window manager*, some systems use that term only for the user interface, and use ‘‘window system’’ for the programming interface.

A *toolkit* is a collection of widgets such as menus, buttons, and scroll bars. When developing a user interface using a toolkit, the designer must be a programmer, since toolkits themselves only have a programmatic interface.

An *interface builder* is a graphical tool that helps the programmer create dialog boxes, menus and other controls for applications. It provides a palette showing the widgets available in some toolkit, and allows the designer to select and position the desired widgets with the mouse. Other properties can then be set. Interface builders are limited to only laying out the static parts of the interface that can be created out of widgets, however. They cannot handle the parts of the user interface that involve graphical objects moving around.

By our definition, a *User Interface Management System* (UIMS) is a more comprehensive tool than an interface builder. A UIMS covers more of the application’s user interface than just dialog boxes and menus. Typically, it will provide help with creating and managing the insides of application windows.

Some tool makers have reported significant gains in productivity by users of their tools. For example, the MacApp tool from Apple has been reported to reduce development time by a factor of four or five [Schmucker 86]. As another example, designers were able to create new, custom widgets about 15 times faster with the experimental Peridot system than by coding the widget using conventional techniques [Myers 88].

4. Survey Methodology

A draft of the survey was circulated on the SIGCHI electronic mailing list, and a number of useful comments were incorporated. The final survey was published in the *SIGCHI Bulletin* [Myers 91a] and *SIGPLAN Notices* [Myers 91b]. Also, it was distributed on several electronic bulletin boards and sent explicitly to a number of people. The responses were all received between April, 1991 and November, 1991.

We should emphasize that although some of the respondents were recruited directly, the majority were self-selected. However, given the breadth of the response (as shown in Figure 1), we feel the results will be useful in a variety of personal computer and workstation contexts.

An important goal of the survey was to differentiate the time and code spent on the ‘‘user interface portion'' of the application from the rest. Unfortunately, previous surveys have shown that many people have difficulty separately identifying these two parts [Rosson 87]. Therefore, at the beginning of the survey, we included the following paragraphs:

> The term ''user interface'' is notoriously difficult to define. In this survey, we intend it to mean the software component of an application that *translates a user action into one or more requests for application functionality*, and that provides to the user *feedback about the consequences of his or her action*. This software component (or components) would be distinguished from the underlying computation that goes on in support of the application functionality. Also, we are *not* including the part of the application that generates hardcopy output (e.g., for printing) in the user interface component.
>
> If you are not happy with our definition, please describe why. However, in answering the remaining questions, please try to apply this definition as best as you can.

No one reported any difficulty with our definition, or entered a different one.

5. Survey Results

We received responses from 74 individuals representing a variety of countries and types of organizations (see Figure 1). 70% came from the US, 15% were from Europe, 8% were from Canada, and 7% were from other places. Most respondents are part of the software industry, either in software development companies (44%) or software research labs (29%), with the remainder from universities (27%). Thus it is not surprising that most of the applications described were developed as commercial, internal or military products (75%). Although the data include a reasonable number of research systems (25%), most of the respondents indicated that they intended these systems to be used by others; virtually none of the systems are ''throw-aways.''

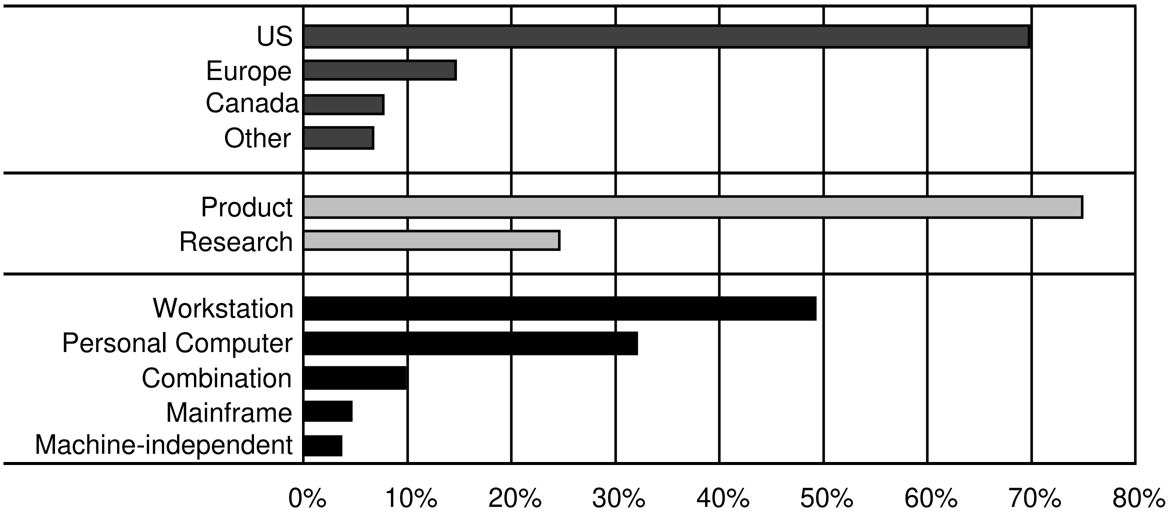

Figure 1: Distribution of survey responses across countries, type of project, and host computer. The ''Combination'' systems used multiple types of computers at the same time. The ''Machine-independent'' systems were designed to run on different kinds of computers.

5.1 Systems

We asked that the answers to the survey questions be based on a single recently-developed application. Application domain was quite varied, including programs intended for sophisticated users (e.g., operating system services and diagnostics, window managers), programming aids (e.g., structured editors and browsers, visual languages), process control and military systems, office applications of many sorts (e.g., database and accounting systems, word processing, data analysis), simulation and CAD systems, educational software, and even a few games. These includes a number of major, well-known commercial products.

As can be seen in Figure 1, the most common hosts for these applications were either workstations (49%) or personal computer systems (32%). The workstations include 8 systems for Sun, 4 for DECStations, 4 for HP, 2 for Silicon Graphics, and one each for RS6000, Intel, Apollo, and Tandem. 14 workstation systems did not specify which platform was used. The personal computer category includes 12 programs for IBM PCs or compatibles, 9 for Macintosh, 2 for PS/2, and one for an Amiga. There were also 4 systems for mainframes, and 3 systems designed to be portable across multiple machines. One of the interesting results is that a significant number of the systems (7, which is 10%) involved a user interface on a smaller computer which was in communication with a bigger computer. These are labeled ‘‘Combination’’ in Figure 1.

A majority of systems (51, which is 69%) used the C programming language. Other languages used included Assembly language (9 systems), Fortran (7), C++ (8), UIL (the OSF Motif description language: 5), Hypertalk (3), Pascal (2), Objective-C (3), Ada (3), Yacc and Lex (2), Lisp (2), and one system each for Basic, Visual Basic, Cobol, Visual Cobol, PL/1, Enfintalk, Smalltalk, Modula-2, Object Pascal, Bliss, Forth, and Self. All the Ada applications were military. A very interesting result is that 58% of the systems were written using *multiple* languages, which is reflected in the counts above. Often, this was a higher-level language and assembly language, or C++ and C, but other times, a special-purpose user interface language was used along with a ‘‘regular’’ programming language.

We asked whether the applications required the end users to have any special training to use the system (assuming they already had knowledge of the application area), and 48% said ‘‘none.’’ 24% reported that the system could be learned with just a few minutes of demonstration or exploration, 17% indicated it would take a few hours, and 12% reported that more substantial training (over a period of days) would be needed. Hopefully, this reflects a growing ease of use of the applications being written, rather than an unrealistically optimistic view of the user interfaces. 15 respondents did not supply any data on this question.

5.2 Developers

Most of the people who filled out the survey were experienced programmers. The median years of experience was 7, and the median number of applications developed was 5. Most of the projects (72%) involved multiple persons, although only 7% of the development groups had greater than 10 individuals. The largest project reported 200 developers, but some large projects did not report the number. For the multi-person projects, the respondent was usually the manager or the person in charge of the user interface. In some cases, domain experts or future users were part of the development team, and a few projects used consultants to help in designing the user interface.

5.3 Size of Applications

There was an enormous range in the size of applications: from 400 lines of code up to 5,000,000 (see Figure 2); the average was 132,000 lines and the median was 19,000. In terms of number of man-years for the entire project, the range was 0.01 man years (about 1 week), up to ‘‘several hundred’’ man years.[1] The median was 2 man years.

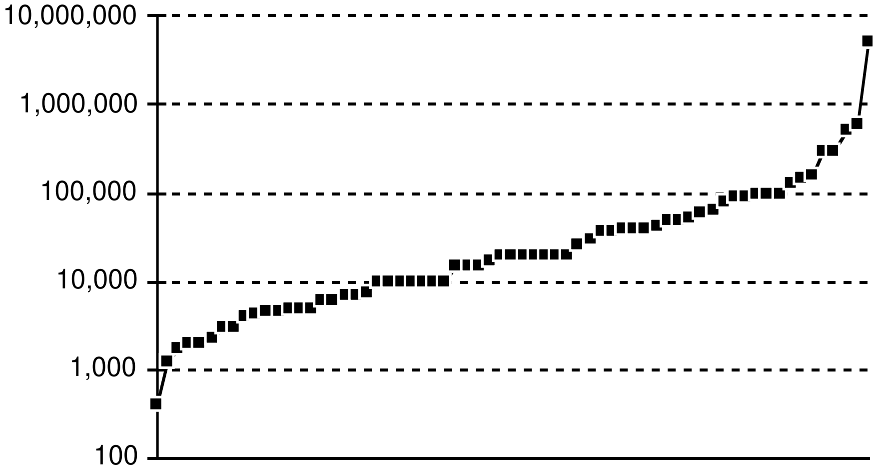

Figure 2: Total number of lines of code on a log scale (for the 68 systems that reported a value).

5.4 Breakdown of Development Time

We asked what percent of the time was spent on each of the phases of the development. 40 projects provided full answers to this question. For these, the results were an average of 20.3% of the time spent on design, 49.5% of the time spent on implementation, and 30.3% of the time spent on maintenance. 20 projects were not sufficiently finished to have values for the maintenance phase (or at least they did not provide a value). For these, the average times were 34.8% for design and 65.2% for implementation.

5.5 User Interface of Applications

In an effort to characterize the user interfaces of the projects described, we offered respondents several check-lists of interface characteristics, covering input (e.g., mouse, keyboard, tablet), output (e.g., bitmap, alpha-numeric, audio), interaction techniques (e.g., menus, commands, buttons, dialog boxes), and presentation techniques (e.g., charts, drawings, images).

[1]To put the upper bound into perspective, it was reported that by the time the WordPerfect word processor program for Microsoft Windows is shipped, an estimated 120 man-years will have been poured into the project by programmers and in-house testers [WordPerfect 91].

Most (82%) of the systems used a mouse. Only one system reported using an exotic input device, and it was a scanner to read text. None reported using a DataGlove, touch tablet, video camera, etc. Similarly, few used unusual output devices: 70% supported only bitmap screens, 16% supported only character terminals, and 13% supported both. 72% of the systems supported color. Only 6 systems reported using audio output for anything other than beeping. These included digitized audio in multi-media presentations, audible ticks as feedback during tracing, synthetic speech for blind users, and simple voice messages.

78% of the applications ran under a window system. The most popular were X/11 (40%), Macintosh (16%) and Microsoft Windows (5%); others mentioned were Amiga, Gem, DECWindows, HP-VUE, Next, Presentation Manager, Silicon Graphics, SunView, Symbolics, Vermant Views and Zinc. Six systems used internally developed window packages, and one system supported multiple windowing systems. Of those using X/11, 52% used OSF Motif, 13% used OpenLook, and 35% used a different X/11 window manager, such as uwm or twm. These results are consistent with the distribution of machine types shown in Figure 1.

Independent of whether a window system was used, the survey asked whether multiple windows were used as part of the system's user interface. This is relevant, since a program not on a window system might implement windows internally, and a program on top of a window system may only use a single window. 73% of the applications used multiple windows in their interface. Of these, 57% used only overlapping windows, 20% used only tiled windows, and 22% used both kinds. It is interesting to note that 14% of the applications that were implemented on top of a window system did *not* use windows in their user interface, and 33% of the systems that were not implemented on top of a window system still *did* use windows (presumably, implemented internally in their application). Of the last group, about half were tiled and half were overlapping. Most of these were on a PC; one was on a mainframe. We speculate that they might have built their own window systems because the projects were started before appropriate window systems were available on those platforms.

84% of the applications used some kind of menu. Menus were popular even with applications not using a mouse, with over half of the non-mouse systems having menus. Property sheets (also called forms or dialog boxes), were also very popular, and were used by 89% of the systems. Direct manipulation graphical objects (where graphical objects or icons can be selected and manipulated using a mouse) were used by 55% of the applications.

Most user interfaces incorporated graphical presentation techniques to some extent, with 70% of the applications using 2-D graphics, and 14% using 3-D graphics. Over half of the applications (55%) indicated that they had developed specialized graphical representations of application data (maps, charts, gauges, plots); 23% employed wireframe or rendered drawings.

5.6 UI Development Process

We asked respondents to describe the process they followed in developing the user interface. Many (42%) indicated that the work had been very evolutionary in nature, with design and implementation of the user interface proceeding in parallel (intertwined). Almost all (89%) described some effort aimed at gathering and responding to user input, consistent with the iterative development methodology promoted by user interface specialists [Carroll 85, Gould 85]. 43% reported some level of formal testing with end users prior to release, with only two respondents indicating that the testing had little or no effect. Of the seven respondents not describing any interactions with users, two indicated that the user interface had been based on some other already tested system.

The most common user interface development process (46%) was to build one or more prototypes, which were often offered to users for comments or testing. In a few cases, these prototypes became the released product, but more frequently they were part of earlier design activities. One project complained that the actual implementation team ignored the user interface team's carefully constructed prototype, but most reported that the prototype guided the final design. Other projects (17%) carried out evaluations of paper designs.

Other techniques for considering the needs of end users were also described. In some cases (11%), this involved participatory design in which end users contributed directly to the design of the user interface; in others, the design team interviewed users or observed them at work. 12% of the respondents claimed to have developed user scenarios as part of their design process.

Two projects reported developing a style guide as part of the systems' development. Most of the Motif projects reported using the OSF Motif Style Guide, and most of the Macintosh projects relied on the Apple Human Interface Guidelines. One project reported following the IBM CUA style guide, and one received guidance from several user interface textbooks.

5.7 Tools Used

Most of the projects (74%) used tools of some sort in developing the code for their user interfaces. For many projects (34%) this consisted of a toolkit providing a library of user interface components (e.g., menus, buttons). As for the case with window managers, the most common toolkits were those for X11 systems (e.g., Motif, OpenLook) and for the Macintosh. Other toolkits mentioned included the Amiga, Athena Widget Set, DecWindows, Interviews, Objective C library, Silicon Graphics, SunView, and Vermont Views.

Other projects used more sophisticated tools, often in concert with a supporting toolkit. So, for example, 20 projects (27%) reported the use of a UIMS. Five of these used Hypercard; other UIMSs included Designer Forms, Domain Dialog, Easel, Enfin, Garnet, Lex/Yacc, Menlo Vista Forms, MetaWindows/Plus, Visual Basic and Visual Cobol. Two projects used internally-developed UIMSs. Ten projects (14%) used interface builders; these included DevGuide, HP-UIMX, MacFlow, Next Interface Builder, TAE+, VUIT, and WindowsMaker.

5.8 User Interface Programming

A major goal of the survey was to assess the code and effort spent on developing the user interfaces of applications. Thus we asked respondents to estimate the percent of code devoted to the user interface, as well as the percent of time spent designing, implementing and maintaining the interface. The code percentage estimates ranged from 1% to 100%, with an average of 47.6% (see Figure 3). Respondents spent an average of 44.8% of design time on the user interface, 50.1% of implementation time, and 37.0% of maintenance time (Figure 4). These estimates did not differ significantly as a function of the type of application described, the country in which the work was done, or the host computer system.

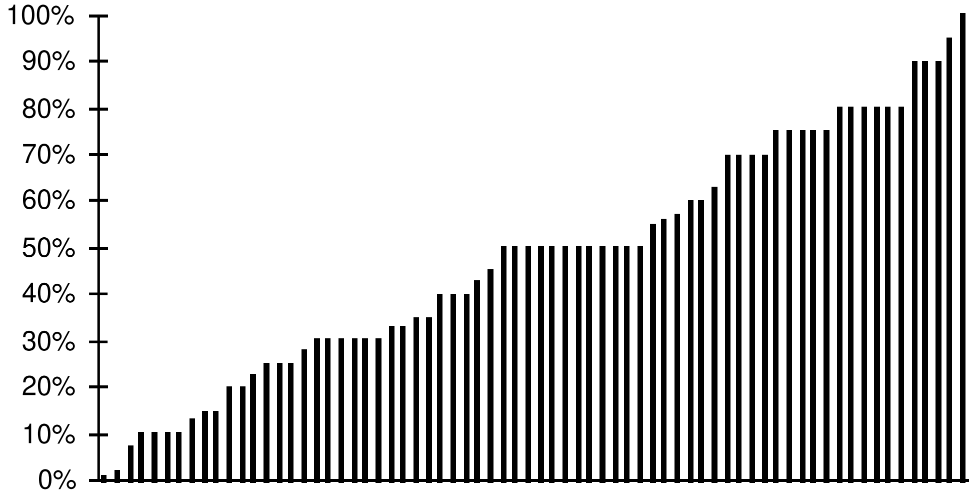

Figure 3: The percent of the code devoted to the user interface (for the 71 systems that reported a value).

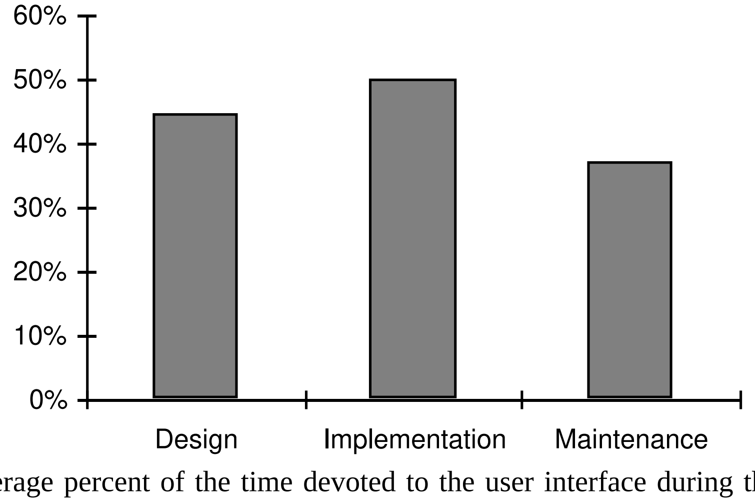

Figure 4: The average percent of the time devoted to the user interface during the various phases of the system development (N = 63 for design, N = 63 for implementation and N = 42 for maintenance) .

These estimates do seem to be related to the kinds of tools the projects used in building their user interfaces. We grouped projects according to the tool use they reported: in Figures 5 and 6, 'No Tools' refers to respondents who reported the project used no special user interface programming tools; 'Toolkit' refers to those reporting use of a toolkit only; and 'UIMS or Builder' refers to those reporting use of a UIMS or of an interface builder (whether or not they also reported using a toolkit). The code percent for the 'No Tools' group was 45.2%, for the 'Toolkit' group it was 57.0%, and for the 'UIMS or Builder' group, 40.6%. Comparable figures for the implementation time estimates were 44.0%, 64.9%, and 41.2%. These data suggest that the projects reporting use of toolkits devoted more code and spent more time implementing their user interfaces (the trend is marginally significant for the code percent measure, Kruskal-Wallis Chi-Square (2) = 5.54, $p < .07$; Kruskal-Wallis Chi-Square (2) = 10.34, $p < .01$, for implementation time).

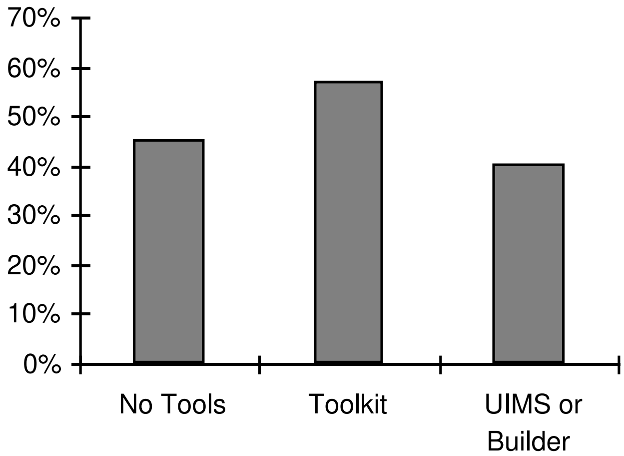

Figure 5: Comparison of the average percent of the *code* devoted to the user interface for projects with different levels of tool use (N = 18 for No Tools, N = 25 for Toolkit and N = 27 for UIMS or Builder.

The differences in these estimates between the 'UIMS or Builder' group and the 'Toolkit' group are what one would expect: UIMSs and interface builders are intended to provide high-level programming support and management of the kinds of user interface components provided by toolkits, and thus should reduce the time and code devoted to user interface development. However, we were surprised to see that the estimates for projects using no tools at all were also less than those for the groups using toolkits. One possibility is that the developers in the 'No Tools' group were attempting less in terms of user interface, either because they knew they did not have the appropriate tools, or because their applications had simpler user interface needs.

In an effort to examine this issue, we used respondents' reports of interface techniques as a rough measure of the complexity of the user interface, summing together the number of interface features they had checked from our lists of input, output, interaction, and presentation characteristics. Although the actual numbers have little meaning, the comparison across the three levels of tool use was as expected, with the fewest techniques reported by projects using no tool support (Kruskal Wallis Chi-Square (2) =

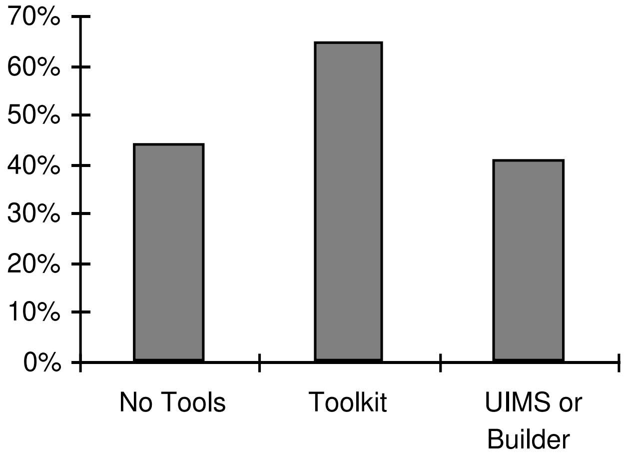

Figure 6: Comparison of the average percent of the *implementation time* devoted to the user interface for projects with different levels of tool use. (N = 15 for No Tools, N= 22 for Toolkit and N = 26 for UIMS or Builder).

9.88, $p < .01$). The greatest number of techniques were reported by the projects using toolkits only. Toolkits are sometimes promoted over UIMSs or builders because they offer greater flexibility to the application programmer [Lee 91]. These survey results are consistent with this claim, in that projects relying on toolkits incorporated a larger number of features into their user interfaces, but at greater cost with respect to implementation time and code.

We were also curious about the relative impact of different user interface characteristics, so we did a series of analyses contrasting projects who did or did not incorporate a given feature. Given the post hoc nature of these analyses, the findings must be interpreted with caution. This caveat aside, we found that the strongest predictor of design time was the use of menus (systems using menus devoted 49.1% of design time to user interface vs. 24.6% for those not using menus; Kruskal-Wallis Chi-Square (1) = 6.78, $p < .01$). This could be due simply to the fact that applications with more complex functionality are more likely to need menus; it could also be that menu organization, terminology and interaction are seen as an important usability concern and thus are likely to increase the relative time spent on user interface design. The factor most likely to increase both implementation time and percent of code was the use of a bitmap display (bitmap applications devoted 55.6% of implementation time to user interface vs. 32.6% for non-bitmap, Kruskal-Wallis Chi-Square (1) = 8.58, $p < .01$; and 51.9% vs. 32.0% of the code, Kruskal-Wallis Chi-Square (1) = 7.18, $p < .01$). This seems likely to be due to the enablement of more sophisticated graphical interfaces with bitmap displays, but again at greater implementation cost.

5.9 Modifications

51% of the respondents reported that they had been able to re-use part of older code when creating this system. We asked if the system was modularized well enough so that the user interface could be modified without changing the application code. Not surprisingly, most (76%) said yes. However, it is interesting to note that some of the users of modern toolkits, like Motif, said that their code was tightly coupled to the particular toolkit, and therefore was not well modularized.

18% of the systems claimed to support different natural languages (such as English and French). For those that did not, 28 respondents estimated how long would take to convert to another language, with an average of 1.8 months. The most common technique suggested for separating the user interface from the rest of the application was to put all the text strings into a separate file.

5.10 Evaluation of the Tools

In general, the respondents were quite pleased with the tools they used. When available, interface builders were especially appreciated, and were mostly thought to be easy to use. Another important feature mentioned more than once was the ability to execute the interface (including application functions) while still inside the interactive tools. When interactive tools were not available, people wished they had them. Recently, a large number of interface builders have appeared for almost every toolkit, so finding a builder will probably not be a problem for future projects.

Some quotes:

> [The toolkit has a] well-designed look-and-feel and api. [The interface builder] generated good samples of the (then) evolving api.
>
> [The toolkit] is very easy to learn, even with limited windows experience.
>
> [The toolkit] gives [us] a lot of low-level control.
>
> I could get a prototype up to show people relatively quickly.
>
> [I liked best] the ease of development and fast development and enhancement times involved. After all, we were (and are) able to achieve our objectives the simple way.
>
> The ... interface builder was powerful (very little coding) and easy to use.
>
> [I liked] drawing dialog boxes interactively; [it was] like using a straight-forward drawing program. [I also liked] the code generation ... [and] source code ‘‘maintenance’’ of [the interface builder]. It lets you define source code modules you require and it generates the makefile for you automatically.
>
> Using the [UIMS] enhanced our productivity significantly.

Many of the complaints dealt with performance problems and bugs in the tools. Other common problems were that the tools were difficult to learn to use, and too slow. For example, some comments were:

> [The toolkit has a] poorly designed look-and-feel [and an] unusually poor application-programming interface....
>
> Both [the graphics package and the toolkit] are absurdly complex and inefficient. They’re slow, poorly documented, plagued by bugs, and eat *incredible* amounts of memory to perform the simplest tasks, which they then neglect to de-allocate. It also requires ridiculous amounts of code to perform those tasks. True, it is quite flexible....
>
> [The toolkit had a] high learning curve. [With it, we are] prone to make mistakes (such as wrong type or number of arguments).
>
> [I like least the] annoying licensing restrictions. We rejected more than one tool simply because we didn’t want to sign up for eternal bookkeeping of license fees.
>
> [The tool] doesn’t let you create the standard ... look and feel—this is true in many ways, large and small. This was enormously costly in time and salaries as we tried over and over again to compenstate for simple flaws....

To be fair, many of these projects used early versions of the tools, and one might expect that some of the problems have been fixed in more recent versions.

Some users called for extended capabilities such as the ability to draw the dynamic parts of windows. Since a few research tools, such as Lapidary [Myers 89] and DEMO [Wolber 91], now support this, we can hope that commercial products will provide this capability in the near future.

5.11 Most difficult aspects of the development of the UI

There were many interesting responses to the question about the most difficult aspects of the development of the user interface. Many of these related to the *design* of the user interface, rather than its implementation. The most commonly raised issues about the design were:

- Getting information from users about what they want, and trying to predict their requirements.
- Designing for the naive user while accommodating the experts.
- Writing the help and documentation text so untrained users could understand it.
- Achieving consistency, especially when there are multiple developers.
- Selecting colors and fonts.
- Understanding and conforming to Motif guidelines.
- Finding appropriate user testing subjects.

Some of these problems can be seen as challenges for future tool developers. For example, future tools can probably help achieve consistency, select colors and fonts, and enforce conformance with guidelines.

The issues raised about the *implementation* included:

- Learning how to use the X library. (But one respondent highly recommended the book by Young [Young 89] to help with this.)
- Achieving acceptable performance.
- Communicating between the user interface part and the application part. This includes problems with the use of call-back procedures.
- Communication between different computer languages.
- Getting enough physical memory. (Almost all DOS users and some Macintosh users complained about memory management.)
- Portability across different windowing systems (e.g., PC and X).
- Finding bugs in the user interface software. One large-scale project noted that the automatic testing mechanisms used by the company did not find a number of serious mouse-driven bugs.

Again, these are clearly issues that future tools, and even future versions of today's tools, would be expected to handle.

6. Conclusions

From this survey, we can tell that user interface development is a significant part of the design and development task, and that user interface tools are being extensively used to help. Users are being involved in the design of most systems, and the design and implementation are often intertwined. Today's tools seem to be helping designers create more sophisticated user interfaces, and the UIMSs and interface builders are helping to decrease the percent of effort devoted to the user interface. However, the amount of time devoted to the user interface has not yet been substantially reduced by the tools. The challenges for future tool creators seem to be to provide tools which are easier to learn and which significantly Increase the efficiency of the user interface designers.

Acknowledgements

First, we would like to thank all the respondents for filling out the surveys, as well as their managers for allowing them to. Also, a few people were instrumental in getting us surveys from their organizations. For help with this paper, we would like to thank Brad Vander Zanden and Bernita Myers.

References

[Carroll 85] John M. Carroll and Mary Beth Rosson.
Usability Specifications as a Tool in Iterative Development.
Advances in Human-Computer Interaction, Volume 1.
In H. Rex Hartson,
Ablex Publishing, New York, 1985, pages 1-28.

[Fox 86] Mark Fox.
Private communication.
1986.
Carnegie Group, Inc., Pittsburgh, PA. 1986.

[Gould 85] J.D. Gould and C.H. Lewis.
Designing for Usability - Key Principles and What Designers Think.
Communications of the ACM 28(3):300-311, March, 1985.

[Hartson 89] H. Rex Hartson and Deborah Hix.
Human-Computer Interface Development: Concepts and Systems for Its Management.
Computing Surveys 21(1):5-92, March, 1989.

[Lee 91] Ed Lee, Mark Linton, John Ousterhout, Len Bass, and Frank Hall.
Interface development tools: Feast or Famine (panel).
In *ACM SIGGRAPH Symposium on User Interface Software and Technology*.
Proceedings UIST'91, Hilton Head, SC, November, 1991.

[Mittal 86] Sanjay Mittal, Clive L. Dym, and Mahesh Morjaria.
Pride: An Expert System for the Design of Paper Handling Systems.
IEEE Computer 19(7):102-114, July, 1986.

[Myers 88] Brad A. Myers.
Creating User Interfaces by Demonstration.
Academic Press, Boston, 1988.

[Myers 89] Brad A. Myers, Brad Vander Zanden, and Roger B. Dannenberg.
Creating Graphical Interactive Application Objects by Demonstration.
In *ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 95-104. Proceedings UIST'89, Williamsburg, VA, November, 1989.

[Myers 91a] Brad A. Myers and Mary Beth Rosson.
User Interface Programming Survey.
SIGCHI Bulletin 23(2):27-30, April, 1991.

[Myers 91b] Brad A. Myers and Mary Beth Rosson.
User Interface Programming Survey.
SIGPLAN Notices 26(8):19-22, August, 1991.

[Myers 92] Brad A. Myers.
State of the Art in User Interface Software Tools.
Advances in Human-Computer Interaction, Volume 4.
In H. Rex Hartson and Deborah Hix,
Ablex Publishing, 1992, pages (in press).

[Rosson 87] Mary Beth Rosson, Suzanne Maass, and Wendy A. Kellogg.
Designing for Designers: An Analysis of Design Practices in the Real World.
In *Human Factors in Computing Systems*, pages 137-142. CHI+GI'87, Toronto, Ont., Canada, April, 1987.

[Schmucker 86] Kurt J. Schmucker.
MacApp: An Application Framework.
Byte 11(8):189-193, August, 1986.

[Sutton 78] Jimmy A. Sutton and Ralph H. Sprague, Jr.
A Study of Display Generation and Management in Interactive Business Applications.
Technical Report RJ2392, IBM Research Report, November, 1978.

[Wolber 91] David Wolber and Gene Fisher.
A Demonstrational Technique for Developing Interfaces with Dynamically Created Objects.
In *ACM SIGGRAPH Symposium on User Interface Software and Technology*, pages 221-230. Proceedings UIST'91, Hilton Head, SC, November, 1991.

[WordPerfect 91] .
WordPerfect for Windows in The Final Stretch.
WORDPERFECT REPORT 5(3):1-3, Fall, 1991.

[Young 89] Douglas A. Young.
The X Window System: Programming and Applications with Xt.
Prentice-Hall, Englewood Cliffs, N.J., 1989.