Visual and Textual Consistency Checking Tools for Graphical User Interfaces

Rohit Mahajan and Ben Shneiderman

Abstract—Designing user interfaces with consistent visual and textual properties is difficult. To demonstrate the harmful effects of inconsistency, we conducted an experiment with 60 subjects. Inconsistent interface terminology slowed user performance by 10 to 25 percent. Unfortunately, contemporary software tools provide only modest support for consistency control. Therefore, we developed SHERLOCK, a family of consistency analysis tools, evaluates visual and textual properties of user interfaces. It provides graphical analysis tools such as a dialog box summary table that presents a compact overview of visual properties of all dialog boxes. SHERLOCK provides terminology analysis tools including an Interface Concordance, an Interface Spellchecker, and Terminology Baskets to check for inconsistent use of familiar groups of terms. Button analysis tools include a Button Concordance and a Button Layout Table to detect variant capitalization, distinct typefaces, distinct colors, variant button sizes, and inconsistent button placements. This paper describes the design, software architecture, and the use of SHERLOCK. We tested SHERLOCK with four commercial prototypes. The outputs, analysis, and feedback from designers of the applications are presented.

Index Terms—Graphical User Interfaces, evaluation tools, consistency, textual and visual style, assessment tools, metrics.

1 Introduction and Previous Research

Consistency in user interfaces follows the second law of thermodynamics. If nothing is done, then entropy will increase in the form of more and more inconsistency in your user interface.

Jakob Nielsen (1989)

GRAPHICAL User Interface (GUI) design is a complex and challenging task. It requires careful requirements analysis, iterative design, and usability testing, [27]. GUI design has become a major part of software development, and is minimally 29 percent of software development budgets [24]. Moreover, data analysis has shown that the user interface is 47 to 60 percent of the total lines of application code [17]. GUI design encompasses more than one third of the software development cycle and plays a major role in determining the quality of a product. Proper human factors techniques, including early completion of user requirements definitions, expert reviews, usability prototype testing, and usability walkthroughs, can significantly speed up software development [14].

Powerful GUI development tools enable the development of working interfaces in a few weeks. However, these interfaces may contain inconsistencies in visual design and textual properties that cannot be detected by current development tools. Such inconsistencies can have a subtle and negative impact on the usability of the interface. Better quality control and GUI test procedures are required, but new analytic and metric-based tools can support the creation of cognitively consistent interfaces having a common "look and feel."

- *R. Mahajan is with BDM International, 1501 BDM Way, McLean, VA 22101. E-mail: rmahajan@bdm.com.*
- *B. Shneiderman is with the Department of Computer Science, Human-Computer Interaction Laboratory and Institutes for Advanced Computer Studies and for Systems Research, University of Maryland, College Park, MD 20742. E-mail: ben@cs.umd.edu.*

Manuscript received 23 May 1996; revised 13 June 1997.
Recommended for acceptance by L. Clarke.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 101217.

1.1 Consistency and Evaluation

Defining Consistency. Consistency is an important aspect of user interface design and is stressed in most guidelines [27], [21]. But, experts have struggled to define exactly "what consistency is?" and "how to identify good consistency?" Reisner [23] states that consistency is neither a property of the system nor the user, but a relation between two potentially conflicting models: the actual system and the user's mental model. Wolf [36] suggests that consistency means that similar user actions lead to similar results. Another definition is that a consistent user interface is one that maximizes the number of shared rules across tasks [22].

Consistency within an application should facilitate human perception and cognitive processes such as visual scanning, learning, and remembering. This applies to spatial properties which includes the organization of menus, placement of frequently used widgets, symmetry, and alignment of widgets. This also applies to fonts, colors, common actions, sequences, terms, units, layouts, typography, and more within an application program. Consistency is naturally extended to include compatibility across the application programs and compatibility with paper or non-computer-based systems. The sequence of pointing, selecting or clicking should be the same throughout the application [29]. Consistency facilitates positive transfer of skills from one system to another leading to ease of use, reduced training time, and improved retention of operating procedures [21], [22].

Kellogg [15] studied the impact of the conceptual dimension of consistency by prototyping a "consistent" version (common look-and-feel and conceptually consistent) and an "inconsistent" version (only common look-and-feel) of an interface. The results of her study, which incorporated a variety of measures like learning time, subjective satisfaction and more, showed that the "consistent" interface was better than the "inconsistent" (consistent in visual appearance and behavior only).

GUI Guidelines. The use of guidelines is important within human-computer interface (HCI) [12]. However there are critics such as Frederiksen, Grudin, and Laursen [9], who showed that consistency guidelines should be applied cautiously to be in harmony with the user's task. According to Grudin [10], interface consistency is a largely unworkable concept and can sometimes work against good design. However, we believe that industrial guidelines documents, such as Apple [1] or Microsoft [19], can educate and direct software developers in positive ways. Empirical evidence of the benefits of many forms of consistency is strong.

Terminology. Inconsistent terminology in interactive dialogs for applications such as text editing can be problematic for users. For example, programs that differ in the names of important commands e.g., "quit" and "exit," are confusing to users [10]. Three studies on command language design showed that users learned positionally consistent systems more readily [3].

Proper use of abbreviation is an important part of terminology. Abbreviations are constructed to reduce typing and optimize the use of screen space, but can impose significant cognitive demands on users. To create an internally consistent design, one abbreviation algorithm, such as truncation, should be used [10].

Tools for Consistent Design. The Interactive Transaction Systems (ITS) project [35] generated consistent interfaces automatically by the use of executable style rules. ITS provided a set of software tools to support four application development roles: an application expert, a style expert, an application programmer, and a style programmer. The ITS architecture divided the application into three parts, namely: application functions, a dialog manager, and views supporting user interfaces. This architecture helped to create a consistent interface for a family of applications and to create multiple consistent interfaces for a given application. ITS has been used to create a number of large-scale applications.

Interface Evaluation Methods. Interface evaluation is a difficult process. Evaluation of a software product's user interface using four techniques—heuristic evaluation, usability testing, guidelines and cognitive walk-throughs—showed that each has advantages and disadvantages [13]. For instance, heuristic evaluation identifies more problems than any other method, but it requires UI expertise and several evaluators. Similarly, usability testing identifies serious and recurring problems, but requires UI expertise and has a high cost. The requirements for these powerful methods, which may include availability of working prototypes, test users, expert evaluators, and time constraints are hindrances in applying these methods more frequently. The study also showed that usability testing, a powerful and effective evaluation method, is not good in finding consistency problems. Therefore, consistency checking tools are likely to be a beneficial complement to usability testing.

Furthermore, usability testing works best for smaller applications. It is too costly to run a usability test on applications with hundreds of dialog boxes. Finding anomalies while reviewing numerous dialog boxes is hard even for expert reviewers, who may fail to detect some flaws and inconsistencies. In contrast, automated evaluation tools can be used in early prototypes (or during late iterations) and can detect anomalies across all the dialog boxes.

1.2 Evaluation Tools for Visual Design and Textual Properties

Automated tools for consistency checking are meant to replace the current manual consistency checking process which is complex, expensive, error prone, and time consuming. These tools can be made independent of platform and development environment. A pioneering tool to evaluate alphanumeric displays derived six measures: Overall Density, Local Density, Number of Groups, Size of Groups, Number of Items, Layout Complexity [31]. These measures were later incorporated into a Display Analysis Program to analyze alphanumeric display [32], [33]. The results of his user study indicated that it can accurately predict the relative search times and subjective ratings.

Streveler and Wasserman [30] proposed novel visual metrics to quantitatively assess screen formats which have similarities with Tullis's Display Analysis Program. They proposed three basic techniques: "boxing," "hot-spot" and "alignment" analysis. A balance measure was also proposed that computed the differences between the center of mass of the array of characters and the physical center of the screen. These proposed metrics were not applied to any system to validate them. Tullis's complexity metrics were later applied to the domain of interactive system design with findings strongly supporting their applicability [7].

The evolution of modern user interfaces, like multimedia interfaces, has sparked research in automated evaluation based on visual techniques. Vanderdonckt and Gillo [34] proposed five visual techniques (Physical, Composition, Association and dissociation, Ordering, and Photographic techniques) that are more sophisticated than traditional properties such as balance, symmetry, and alignment. Dynamic strategies for computer-aided visual placement of interaction objects on the basis of localization, dimensioning, and arrangement were introduced by Bodart et al. [4]. They defined mathematical relationships to improve the practicability, the workability and the applicability of their visual principles into a systematic strategy, but specific metrics and acceptance ranges were not tested.

Sears [25], [26] developed a first generation tool (AIDE) using automated metrics for both design and evaluation using Layout Appropriateness metrics. In computing the Layout Appropriateness the designer provides the set of widgets used in the interface, the sequence of actions to be performed by the user, and how frequently each sequence is used. The appropriateness of a given layout is computed by weighing the cost of each sequence of actions by how frequently the sequence is performed. Layout Appropriateness can be used to compare existing layouts and to generate optimal layouts for the designer. AIDE has demonstrated its effectiveness in analyzing and redesigning dialog boxes in simple Macintosh applications and also dialog boxes with complex control panels in NASA applications. Studies by Comber and Maltby [8] assessed the usefulness of layout complexity metrics in evaluating the usability of different screen designs. Mullet [20] developed a systematic layout grid strategy to easily position related controls con-

sistently across dialog boxes. Using this systematic approach, he showed that the GUI of the "Authorware Professional", a leading development tool for learning materials in the Macintosh and Windows environments, could be easily redesigned to create a more coherent, consistent, and less crowded layout.

1.3 User Studies on the Effects of Interface Inconsistencies

Chimera and Shneiderman [5] performed a controlled experiment to determine the effects of inconsistency on performance. This experiment used two interactive computer systems: the original inconsistent version and a revised consistent version. The revised version had consistent screen layouts and colors, and used consistent task-oriented phrases for the description of menu items. The results showed that there was a statistically significant difference favoring the revised interface for five tasks and favoring the original interface for one. They concluded that the revised interface yielded faster performance and higher satisfaction due to consistent location, wording, and color choices.

Bajwa [2] studied the effect of inconsistencies in color, location, and size of buttons on user's performance and subjective satisfaction. For a billing system interface, three inconsistent versions were created with 33 percent inconsistency in color, location, and size. Her results showed that inconsistency significantly effects user's performance speed by about 5 percent.

2 Experimental Support for SHERLOCK

2.1 Introduction

We designed an experiment to test the hypothesis that terminology consistency increases performance speed and subjective satisfaction. The experiment considered only one aspect of inconsistency, misleading synonyms. We developed a GUI in Visual Basic for the students to access the resources of the University's Career Center. Three versions of the interface were created. The first version was terminologically consistent. The second version had a medium level of terminology inconsistency (one inconsistency was introduced for each task and each task had an average of four screens). The third version had a high level of terminology inconsistency (one or two inconsistencies were introduced for each task). Some designers argue that inconsistencies are easily overcome with a few minutes of usage. To test this conjecture we had two levels of training: no prior training and five minutes of training. The resulting 2 × 3 between-groups experiment had two independent variables which were level of training (0 and 5 minutes) and the type of interface (no inconsistency, medium inconsistency, and high inconsistency). For all six treatments, users were given the same task list and their task completion time and subjective satisfaction were evaluated. For each treatment, 10 subjects were selected, making a total of 60 subjects.

2.2 Interface Design

All the screens of the Career Center interfaces had a consistent visual design (sizes of similar screen, placement of similar items, screen density, margins, typefaces, colors etc.). Inconsistencies were introduced in the medium and high inconsistency versions.

In the medium inconsistency version, one terminological inconsistency was introduced for every task. Since the subjects were told to perform seven tasks, the interface had seven terminology inconsistencies. In the high inconsistency version, one or two terminology inconsistencies were included per task for a total of 11.

These inconsistencies included changing the heading of the dialog box from "Questions" to "Inquiries" or changing the widget labels from "Workshops" to "Seminars." Also, menu items were changed from "Career Counseling" to "Career Advising" and "View" to "List." Inconsistency in button labels were also introduced by changing "OK" and "Abort" to "Forward" and "Discard."

2.3 Hypotheses

The task completion time for the interface with no terminology inconsistency will be significantly lower than the interfaces with medium and high terminology inconsistency in the case of subjects with no prior training of the system.

The difference in task completion time for the interface with no, medium, and high terminology inconsistencies will decrease when subjects are given five minutes of training with the system, prior to the execution of the tasks.

The subjective satisfaction will be significantly higher for the interface with no terminology inconsistency as compared to those with medium and high terminology inconsistency.

2.4 Subjects

Sixty University of Maryland students participated. All were familiar with Windows 3.1 and mouse operation. Half were given five minutes training with the no inconsistency interface.

2.5 Materials

The experiment was run on a 100 MHz Pentium machine with a 17 in. color monitor having a 1024 × 768 pixel resolution with 256 colors. A set of written instructions was provided. The subjects were asked to fill out a modified version of the Questionnaire (QUIS) [6] after completing the experiment.

2.6 Task List

The subjects performed seven tasks:

- Register for Workshop II.
- Set appointment with any one of the career counselors for Nov. 8 for 10:00 to 10:30 a.m.
- View part-time job openings in Computer Science Major in Maryland state.
- Submit the following question to the Career Center: Do counselors at the career center perform mock interviews?
- Request graduate school information for masters program in business management in any one of the listed universities.
- Register for an interview with any one of the listed companies on Nov. 20 using any one of the open slots.
- Cancel Registration for Workshop II

2.7 Procedure

Administration. Subjects were asked to read the instructions and to sign the consent form. The no training group was

introduced to the interface by a presentation of the menu items. The training group was shown the menus and all the dialog boxes by opening each of the menu items. They were also allowed to use the interface for two minutes to experience the interface. After the experiment, subjects filled out the 19-item subjective satisfaction questionnaire.

2.8 Results

The experimental results (Table 1) show that the no inconsistency version had a faster average task completion time than the medium and high inconsistency versions. The average subjective satisfaction ratings for the no inconsistency version were higher than the medium, and the high inconsistency versions of the interface (Table 2). Overall, the performance improved when training was administered to the subjects, as the average task completion time was lower for all the three versions of the interface when training was provided.

TABLE 1
AVERAGE TASK COMPLETION TIME AND STANDARD DEVIATION
(10 SUBJECTS PER CELL; SEVEN TASKS PER SUBJECT)

Level of Training	No Inconsistency	Medium Inconsistency	High Inconsistency
None	239.0 sec (61.0)	287.4 sec (42.6)	312.7 sec (88.3)
Five minutes	204.0 sec (41.7)	217.4 sec (50.6)	270.7 sec (30.5)

TABLE 2
AVERAGE SUBJECTIVE SATISFACTION RATING
(HIGHER NUMBERS INDICATE INCREASED SATISFACTION)
AND STANDARD DEVIATION (10 SUBJECTS PER CELL)

Level of Training	No Inconsistency	Medium Inconsistency	High Inconsistency
None	142 (14)	130 (14)	134 (13)
Five minutes	142 (12)	139 (11)	138 (13)

A 2 × 3 ANOVA (Analysis of Variance) was used to determine whether the interface types (versions) and the level of training had statistically significant effects on the task completion time and subjective satisfaction, measured across the three treatments (no, medium, and high level of terminology inconsistency) and two training levels (none and five minutes). There was a statistically significance difference for task completion time by training ($F(1, 54) = 12.38$, $p < 0.05$) and interface type ($F(2, 54) = 8.21$, $p < 0.05$), but no interaction effect. This implies that training reduces the task completion time, but training does not overcome the problems caused by an inconsistent design. Differences in subjective satisfaction were not statistically significant.

2.9 Discussion

In relation to the task completion time, the ANOVA identified that the terminology inconsistencies introduced in each version of the interface significantly slowed the user's performance. In the no training group, the average task completion time for the medium, and high inconsistency treatments were 20 and 31 percent more than the no inconsistency treatment. Similarly in the training group, the average task completion time for medium, and high inconsistency were 7 and 34 percent more than the no inconsistency treatment.

The level of training, according to the ANOVA significantly effected the user's performance. On average, the training decreased the task completion time by 14, 24, and 13 percent in no, medium, and high inconsistency versions, respectively. Although the subjective satisfaction ratings for the medium and the high inconsistency versions were less than the no inconsistency version, the ANOVA analysis found no statistically significant differences. It is difficult to obtain statistically significant differences in preference scores for between-groups design, because subjects do not see the other versions. A future within-subjects study might elicit stronger preference differences.

2.10 Conclusion

The results of this experiment, along with the experiment done by Bajwa [2] supported the encouragement to "strive for consistency" and including consistency as one of the prime guidelines when designing user interfaces [27]. Therefore, developing user interface consistency checking tools seems worthwhile to support software engineers during the development process.

3 DESCRIPTION AND DESIGN OF SHERLOCK

SHERLOCK is a family of consistency checking tools to evaluate visual design and terminology in user interfaces. It consists of a set of seven programs that were implemented in about 7,000 lines of C++ code, and developed on the SUN SPARC Stations / UNIX platform. In order to evaluate a GUI using SHERLOCK, its interface description files need to be converted to a canonical format. These canonical format files are the only input required by the SHERLOCK evaluation tools. SHERLOCK was designed to be a generic GUI consistency and evaluation tool.

3.1 Translator and Canonical Format Design

The canonical format is an organized set of GUI object descriptions. These object descriptions embrace interface visual design and terminology information in a sequence of attribute-value pairs. The canonical format is advantageous because of its lucidity and extendibility. It can be easily modified to include new attributes encompassing interface description information in the form of files.

Translator programs are designed for a particular GUI development tool and convert its interface description (resource) file to a canonical format. Design of the data structure for the translator depends on the format of the interface resource file. Two translators were created, one for Visual Basic 3.0 and the other for Visual C++ 4.0 using a lexical scanner generated by FLEX (Fast Lexical Analyzer Generator) which is a tool for generating programs that perform pattern matching on text. Using the lexical scanner, attribute value strings are detected and converted to the appropriate canonical format. All the dimensional coordinates are converted to pixels and other platform and application independent values.

The canonical format may be created for other interface development tools like Power Builder, Galaxy, and Delphi by writing a translator program for those tools.

3.2 SHERLOCK Design

The SHERLOCK data structure was designed to be flexible, extensible and customizable to changes that may be made by expansion of the canonical format files. SHERLOCK has a sequential modular design and can be divided into the following subsystems.

- Widget store
- Dialog box
- String processing
- Spell checker
- Button processing

3.3 SHERLOCK Tools

SHERLOCK is an extension of previous work by Shneiderman, Chimera et al. [28] in which spatial and textual evaluation tools were constructed. These tools have been modified after evaluating sample applications and new tools have been integrated. Our focus was on evaluating only the aspects of consistency that are relatively task-independent and can be automated. Our tools evaluated layout properties such as sizes of dialog boxes, placement of similar items, screen density, consistency in margins, screen balance, and alignment. We also evaluated consistency in visual design properties such as fonts, font-sizes, font-styles, background colors, and foreground colors. Finally our evaluation includes checking for terminology inconsistencies, abbreviations, variant capitalization, and spelling errors in buttons, labels, messages, menu items, window titles etc.

3.3.1 Dialog Box Summary Table

The dialog box summary table is a compact overview of the visual design of dozens or hundreds of dialog boxes of the interface. Each row represents a dialog box and each column represents a single metric. Typical use would be to scan down the columns looking for extreme values, spotting inconsistencies, and understanding patterns within the design.

Choosing the appropriate metrics is critical in the design of the dialog box summary table. The researchers at the University of Maryland generated a list of approximately 40 metrics after reviewing the relevant previous literature, consulting with colleagues and using their GUI evaluation experience. A similar effort was taken by our partners at General Electric Information Services (GEIS), where they brain-stormed and proposed their metrics based on commercial software development experience. The two lists had many similar items which were grouped into categories such as spatial layout, alignment, clustering, cluttering, color usage, fonts, attention getting, etc. The metric set was revised several times after evaluating a series of interfaces. Ineffective metrics were removed, others were redefined, and new metrics were added. The modified column set of the dialog box summary contained:

Aspect Ratio. The ratio of the height of a dialog box to its width. Numbers in the range 0.5 through 0.8 are desirable. Dialog boxes that perform similar functions should have the same aspect ratio.

Widget Totals. Count of all the widgets and the top level widgets. Increasing difference between all and top level counts indicates greater nesting of widgets, such as buttons, lists, and combo boxes inside containers.

Nonwidget Area: The ratio of the nonwidget area to the total area of the dialog, expressed as a percentage. Numbers closer to 100 indicate high utilization, and low numbers (< 30) indicate possibilities for redesign.

Widget Density. The number of top-level widgets divided by the total area of the dialog box (multiplied by 100,000 to normalize it). Numbers greater than 100 indicate that a comparatively large number of widgets are present in a small area. This number is a measure of the crowding of widgets in the dialog box.

Margins. The number of pixels between the dialog box border and the closest widget. The left, right, top, and bottom margins should all be equal in a dialog box, and across different dialog boxes.

Gridedness. Gridedness is a measure of alignment of widgets. This metric has been refined several times, but we have not been able to find a satisfactory metric to detect misaligned widgets. X-Gridedness counts the number of stacks of widgets with the same X coordinates (excluding labels). Similarly, Y-Gridedness counts the number of stacks of the widgets with the same Y coordinates. High values of X-Gridedness and Y-Gridedness indicate the possibility of misaligned widgets. An extension of Gridedness is Button Gridedness where the above metrics are applied to button widgets.

Area Balances. A measure of how evenly widgets are spread out over the dialog box. There are two measures: a horizontal balance, which is the ratio of the total widget area in the left half of the dialog box to the total widget area in the right half of the dialog box; and the vertical balance, which uses top area divided by bottom area. High values of balances between 4.0 and 10.0 indicate screens are not well balanced. The limiting value 10.0 represents a blank or almost blank (for example, a dialog box that has only one widget which is a button) dialog box.

Distinct Typefaces. Typeface consists of a font, font size, bold and italics information. Each distinct typeface in all the dialog boxes is randomly assigned an integer to facilitate quick interpretation. For each dialog box, all the integers representing the distinct typefaces are listed so that the typeface inconsistencies can be easily spotted locally within each dialog box and globally across all dialog boxes. We recommend that a small number of typefaces should be used in an application.

Distinct Background Colors. All the distinct background colors (RGB values) in a dialog box are displayed. Each distinct color is randomly assigned to an integer for display and comparison convenience and is described in detail at the end of the table. The purpose of this metric is to check if the dialog boxes have consistent background colors. Multiple background colors may indicate inconsistency, depending on the application.

Distinct Foreground Colors. All the distinct foreground colors in a dialog box are displayed.

In addition to the dialog box summary table, a set of independent tools were built, including:

3.3.2 Margin Analyzer

Margin Analyzer is an extension of the dialog box summary table's margins metric. This analyzer calculates the

most frequently occurring values of left, right, top, and bottom margins across the interface and then lists margins in every dialog box that are inconsistent with these frequently occurring values. It also calculates what widgets of the dialog box need to be moved by how many pixels to make the margins consistent. The Margin Analyzer tool depends on the fact that the most frequently occurring value of margins are the optimum margin values that the designer would have ideally used for consistency.

3.3.3 Concordance.

The Concordance tool extracts all the words that appear in labels, buttons, messages, menu items, window titles etc. in every dialog box. It can help designers spot inappropriate word use such as variant spellings, abbreviations, tense and case inconsistency, etc. Occurrences of words in a different case are shown to point out potential inconsistent use. The sort order used was aAbB...zZ so that the occurrence of "cancel" is not separated from "Cancel" or "CANCEL."

3.3.4 Interface Concordance.

The Interface Concordance tool checks for variant capitalization for all the terms that appear in buttons, labels, menu items, and window titles etc. This tool outputs strings that have variant capitalization, listing all the variant forms of the string and its dialog box sources. These variant forms may be acceptable, but they should be reviewed by a designer. For example the words "MESSAGES," "messages," "Messages," and "mesgs" are variant forms of the same word.

3.3.5 Button Concordance

Buttons are one of the most frequently used widgets, performing vital functions like "Save," "Open," "Delete," "Exit," etc. They should also be checked for consistency in their size, placement, typefaces, colors, and case usage. This tool outputs all the buttons used in the interface, listing the dialog boxes containing the buttons plus fonts, colors, and button sizes. The Button Concordance identifies variant capitalization, distinct typefaces, distinct foreground colors, and variant sizes in buttons.

3.3.6 Button Layout Table.

Often a set of buttons frequently occur together (for example, OK Cancel, Help), and therefore it is desirable that these appear in the same order and have the same size. If the first button in the set is detected, then the program outputs the height, width, and position relative to the first button of every button detected in the list. The relative position of every button detected in the set is output as (x + offset, y + offset) to the first button, where offset is in pixels. Buttons stacked in rows would yield a (x + offset, y) relative position and those stacked in columns would yield (x, y + offset). The Button Layout table identifies inconsistencies in button placement, and variant button sizes locally within a dialog box and globally across all the dialog boxes. Additionally, the tool helps to determine synonym button labels in button sets, for example use of both "Quit" and "Exit" with the "OK" button in different dialog boxes. Some of the sample button sets are:

- `OK Cancel Close Exit Quit Help`
- `Stop Halt Pause Cancel End Close Done Exit Quit`
- `Add Remove Delete Copy Clear`

3.3.7 Interface Spellchecker

The Interface Spellchecker reads all the terms from buttons, labels, menu items, messages, titles etc. and outputs terms that are not found in the dictionary. The spell checking operation is performed within the code and all the possible misspelled words are stored in a file. This file can be reviewed by the designer to detect possible misspelled and abbreviated words which may create confusion for users. The output is filtered through a file containing valid computer terms and default Visual Basic terms that may be flagged as spelling errors by the dictionary.

3.3.8 Terminology Baskets

A terminology basket is a collection of computer terms including their different tenses that may be inadvertently used as synonyms by interface designers. Our goal is to construct different sets of terminology baskets by constructing our own computer thesaurus and then search for these baskets in every dialog box of the interface. The purpose of terminology baskets is to provide interface designers with feedback on misleading synonymous computer terms, like "Close," "Cancel," "End," "Exit," "Terminate," and "Quit." The program reads an ASCII file containing the basket list. For each basket all the dialog boxes containing any of the basket terms are output. Some of the idiosyncratic baskets are:

- `Remove Removes Removed Removing Delete Deletes Deleted Deleting Clear Clears Cleared Clearing Purge Purges Purged Purging Cancel Cancels Canceled Canceling Refresh Refreshed`
- `Item Items Entry Entries Record Records Segment Segments Segmented Segmenting Field Fields`
- `Message Messages Note Notes Letter Letters Comment Comments`

4 INTERFACE EVALUATIONS

4.1 Testing the Evaluation Tools

The effectiveness of the SHERLOCK tools was tested with four commercial prototype applications developed in Microsoft Visual Basic. These applications included a 139 dialog box interface for the GEIS Electronic Data Interchange, a 30-dialog box GE business application, a 75-dialog box Italian business application, and a set of University of Maryland AT&T Teaching Theater interfaces combined into an 80-dialog box application. The analysis of the 30-dialog box GEIS application and the Italian business application is not discussed in this paper because the results were similar to the other two applications.

4.2 Evaluation Results, GE Interfaces

The 139-dialog box GEIS Electronic Data Interchange interface was the first prototype evaluated. Although this was a

well-reviewed and polished design, SHERLOCK detected some inconsistencies which may have otherwise been left undetected.

4.2.1 Dialog Box Summary Table Analysis

Aspect Ratio. Aspect Ratio varied from 0.32 to 1.00. Many dialog boxes that performed the same functionality had different Aspect Ratios, indicating a potential inconsistency.

Nonwidget Area. Nonwidget area varied from 2 to 97.5 percent. Some dialog boxes with low Nonwidget area (5 to 15 percent) were candidates for redesign.

Widget Density. Widget Density varied from 14 to 271, but most of the high values were due to exceptions in the metric, as none of the dialog boxes had too many widgets in a small area.

Margins. Left, right, top, and bottom margins were inconsistent within a single dialog box and were also inconsistent across the interface. For example, the average value of the left margin was 12 pixels, but the margin ranged from 0 to 80 pixels. Inconsistencies detected by metrics of dialog box summary table like left margin, can easily be spotted, by plotting the metric (Fig. 1).

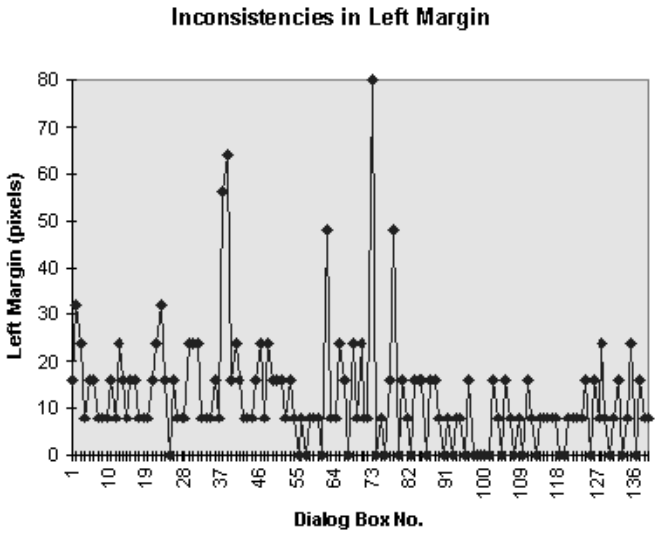


Fig. 1. Inconsistencies in left margin.

Gridedness. Some high values of the Button Gridedness (three or more) metric helped in detecting dialog boxes with misaligned buttons.

Area Balances. Dialog boxes were well balanced as the average value of Left/Right Balance and Top/Bottom Balance was 1.1 and 1.4, respectively.

Distinct Typefaces. Although most of the dialog boxes used a single typeface (MS Sans Serif 8.25 Bold), there were a couple which used more than three typefaces. Altogether seven distinct typefaces were used.

Distinct Background and Foreground Colors. There was much variation in color usage among dialog boxes, indicating inconsistency. The interface used a total of eight foreground and seven background colors (RGB values). Although in some applications, the use of many different colors may be appropriate, in this case it was an inconsistency.

4.2.2 Margin Analyzer.

The margin analyzer successfully detected the dialog boxes that had margin values more than two pixels apart from the most frequently occurring value. For each inconsistent value, it listed the widgets that need to be moved and by how many pixels to make them consistent. There were some exceptions (Visual Basic 3.0 allows widgets to extend beyond the area enclosed by the dialog box and allows the size of label and text boxes to be greater than the text enclosed by them) beyond the capability of the tool to handle, leading to negative margins.

4.2.3 Interface Concordance

The interface concordance tool spotted the terms that used more than one case across the application. For example, terms like “Messages,” “MESSAGES,” and “messages” were detected by the interface concordance tool. Some of the other inconsistencies included variant capitalizations such as “Open,” “OPEN,” and “open.”

4.2.4 Button Concordance

GEIS interfaces did not have any button labels which used more than one case. All the button labels used the title format and were therefore consistent. Also, all the buttons used the same typeface and foreground color. The Button Concordance detected inconsistency in height and width of the buttons across the interface. Table 3 shows a portion of the button concordance output for the “Archive” button. Browsing across the columns of the table, we can see that the width of the “Archive” button varied between 65 and 105 pixels. All the buttons had a top margin of 0 pixels except one which has a top margin of 312 pixels. This is an inconsistency, since all the “Archive” buttons are placed at the top right corner of the dialog box, except one which is placed at the bottom right corner. Button placement inconsistencies were detected in many other buttons including “OK,” “Cancel,” “Close,” “Find,” “Forward,” and “Print.”

4.2.5 Interface Spellchecker.

The tool detected a few misspelled terms, and many potentially confusing, incomplete, and abbreviated words such as “Apps,” “Trans,” “Ins,” “Oprs.”

4.2.6 Terminology Baskets.

The basket browser revealed some interesting terminology anomalies after analyzing the interface that led to reconsideration of the design. As shown in Table 4, terms like “record,” “segment,” “field,” and “item” were used in similar contexts in different dialog boxes. Other interesting inconsistencies included the use of “start,” “execute,” and “run” for identical tasks.

4.2.7 Button Layout Table.

The most common button position and terminology inconsistency was in the button set [OK Cancel Close Exit Help]. The button labels “Cancel,” “Close,” and “Exit” were used interchangeably. Sometimes these buttons were stacked in a column on the top left corner of the dialog box, and in other cases they were stacked in a row at the bottom of the dialog box and were either left, right, or center aligned.

TABLE 3
BUTTON CONCORDANCE

BUTTON LABEL	DIALOG BOX NAME	BUTTON TYPEFACE	BUTTON FG_COLOR	BUTTON (H, W)	BUTTON LEFT	POSITION RIGHT	TOP
Archive	xref	1	1	25,105	208	311	0
	file	1	1	25,89	448	87	0
	file2	1	1	25,73	360	72	0
	filefind	1	1	25,73	408	142	312
	hold	1	1	25, 65	320	55	0
	in	1	1	25, 81	464	79	0
	out	1	1	25, 73	304	55	0
	sent	1	1	25, 81	344	78	0

DISTINCT TYPEFACES IN BUTTONS:
1 = MS Sans Serif 8.25 Bold No Label
DISTINCT FOREGROUND COLORS IN BUTTONS:
1 = Default Color

TABLE 4
TERMINOLOGY BASKETS

Basket: Entries, Entry, Field, Fields, Item, Itemized, Itemizing, Items Record, Records, Segment, Segmented, Segmenting, Segments			
BASKET TERM	FORM CONTAINING THE BASKET TERM		
Field	search		
Items	reconly	reconly	reconly
	reconly	sendrec	sendrec
	sendrec	sendrec	wastedef
Record	ffadm	profile	
Segment	addr	search	

TABLE 5
BUTTON LAYOUT TABLE

DIALOG BOX	OK		Cancel			Exit			Help	
	(H,W)	(H,W)	Rel.	Pos.	(H,W)	Rel.	Pos.	(H,W)	Rel.	Pos.
admprof	22, 68	22, 68	x+16	y	25 82	x+18	y	22, 68	x+98	
checkps	25, 82							25, 82	x+116	
nbatch	25, 62	25, 62	x-1	y+20				25, 62	x,	y+66
systinp	26, 72	26, 72	x+1	y+13				25, 73	x+2	y+48

A portion of the output from the button set [OK Cancel Close Exit Quit Help] is shown in Table 5. Inconsistency in height and relative button positions within a button set can be checked by moving across the table rows. Inconsistency in height and relative position for a particular button can be spotted by moving down the columns. For example, browsing the "OK" button column we found that the height of the "OK" button varied between 22 and 26 pixels and the width varied between 62 and 82 pixels. Scanning across the rows, we found that the relative position of "OK" and "Cancel" buttons varied in all three dialog boxes in which they occurred together. In two of the dialog boxes the "Cancel" button was 20 and 13 pixels below the "OK" button, but in the third dialog box, the buttons were adjacent in the same row. Both "Cancel" and "Exit" were used with the "OK" button to perform the same task which was a terminology inconsistency.

4.3 Evaluation of University of Maryland Interface

The 80-dialog box University of Maryland AT&T Teaching Theater Interface was a combination of applications, all designed for the students to use. Evaluation of this interface highlighted the intra-application inconsistencies that may exist among applications designed for the same users.

4.3.1 Dialog Box Summary Table

A portion of the dialog box summary table from this application is shown in Fig. 2.

Aspect Ratio. Aspect Ratio, in general varied between 0.5 and 0.8, but outliers above or below were detected. All the About (Fig. 3), Cover and Exit dialog boxes had different aspect ratios. These applications were designed for the same set of users and these inconsistencies in Aspect Ratio, especially in the dialog boxes with the same functionality, should be minimized.

Widget Totals. Some dialog boxes had a high value of widget totals i.e., 70 or more widgets. This indicated complexity in the dialog box.

Nonwidget Area. High values of nonwidget area (above 90 percent) were found in some of the dialog boxes, indicating that the use of screen space was not optimum.

Widget Density. Some of the values of widget density (around 150 or more) indicated that too many widgets were present in a small area. Only dialog boxes which had high widget density, but a nonwidget area of 40 percent or more were acceptable.

Margins. Left margins varied from 0 to 192 pixels, although the most frequently used margin values were between 8 and 16 pixels. A quarter of the dialog boxes had left

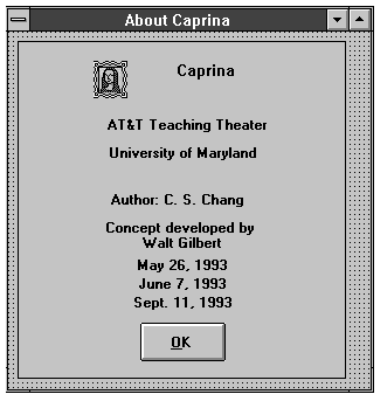
No.	Dialog Name	Aspect Ratio (H/W)	-WIDGET-- TOTALS All	Top-Level	Non-Widget Area (%)	Widget Density widget/area	-----M A R G I N S------ Left	Right	Top	Bottom (pixels)	----GRIDEDNESS----- Top Level X	Y	Buttons X	Y	-Balances-- Area Ratios Horiz (L/R)	Vert (T/B)	Distinct Typefaces	Distinct Background Colors	Distinct Foreground Colors
30	form9	0.74	19	13	88.7	69	0	381	0	17	3	3	0	0	10.0	2.4	4 9	1 2 5 6	3 4 6
31	frmcompaz	0.79	5	4	67.5	11	104	97	56	69	1	2	0	1	1.0	1.4	9 13	2	3 4 7
32	frmcompu	0.79	5	4	67.5	11	104	101	56	69	1	2	0	1	1.0	1.4	9 13	2	3 4 7
33	frmhand	0.48	6	5	67.5	33	32	31	32	31	1	2	0	1	1.0	1.5	9 13	2	3 4 7
34	frmlogin	0.85	8	7	77.1	25	96	96	40	64	1	1	0	1	0.6	1.7	9 13	2	3 4 7
35	frmlogo	0.38	9	8	18.3	25	0	223	0	0	2	2	1	1	0.8	1.3	4 9 16 17	1 2 10	2 3 11
36	frmmatch	0.50	7	6	70.0	60	16	50	24	43	1	1	0	1	0.8	1.3	4	2	3 4
37	frmquesaz	0.42	6	5	75.6	25	56	54	48	61	0	1	0	1	1.1	1.5	9 13	2	3 4 7
38	frmquesu	0.45	6	5	73.9	23	72	14	48	74	0	1	0	1	0.8	1.3	9 13	2	3 4 7
39	graph	0.55	14	4	57.0	22	0	0	0	7	2	3	0	1	1.0	1.0	7 8	2 5 8	2 3 7
40	grid3	0.89	5	4	42.5	23	21	15	13	60	2	1	1	0	1.5	1.2	4	2	3
Maximum		1.60	102	101	100.0	184	192	381	56	276	9	13	2	3	10.0	10.0			
Minimum		0.13	0	0	0.0	0	0	0	0	0	0	0	0	0	0.0	0.0			
Average		0.73	14	9	57.5	48	19	52	11	29	2	2	0	0	1.8	1.6			

DISTINCT TYPEFACES:
1 = Arial 13.5 Bold
2 = Symbol 9.75 Bold
3 = Arial 8.25 Bold
4 = MS Sans Serif 8.25 Bold
5 = System 9.75 Bold
6 = Arial 15.75 Bold
7 = MS Sans Serif 9.75 Bold
8 = MS Sans Serif 16.5 Bold
9 = MS Sans Serif 12 Bold
10 =MS Sans Serif 13.5
11 = Symbol 13.5 Bold
12 = MS Sans Serif 24 Bold Italic
13 = MS Sans Serif 13.5 Bold
14 = MS Sans Serif 18
15 = MS Serif 30 Bold
16 = Arial 18 Bold
17 = Symbol 8.25 Bold
18 = Times New Roman 24 Bold Italic
19 = Times New Roman 30 Bold Italic

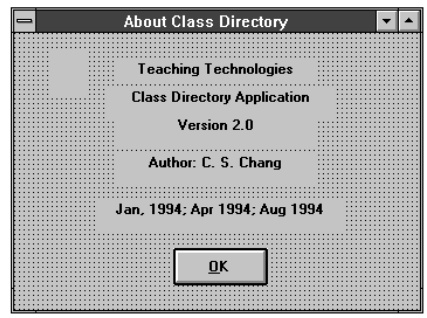
DISTINCT BACKGROUND COLORS:
1 = ffffff
2 = ffffffff80000005
5 = c0c0c0
6 = ff
8 = e0ffff
10 =c00000
12 404040
14 =ffffffff8000000f

DISTINCT FOREGROUND COLORS:
2 = ffffffff80000005
3 = ffffffff80000008
4 = 0
6 = ff
7 = ff0000
9 = c000c0
10 =c00000
11 =ffff
13 =808080
15 =c000

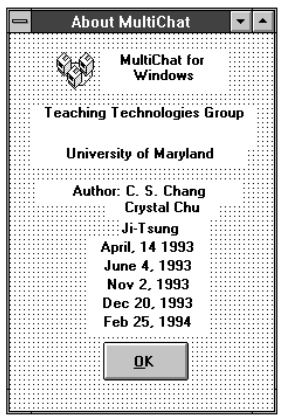
Fig. 2. A portion of dialog box summary table.



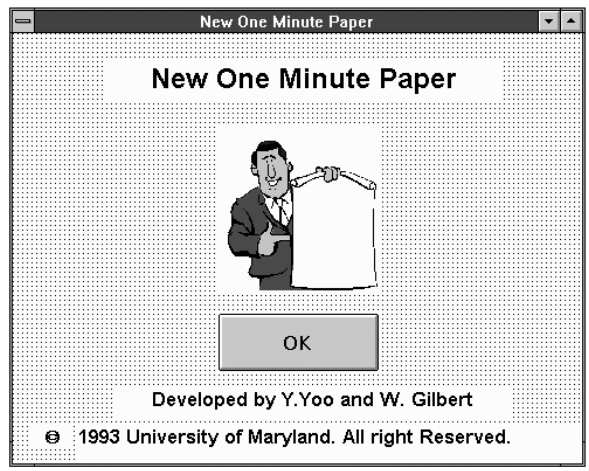
About2.frm = 0.99



Aboutbox.frm = 0.68



About_m.frm =1.5



About_n.frm = .84

Fig. 3. Aspect ratios shown below each dialog box reveal inconsistencies.

margin values of 0 pixels and a few had high values above 70 pixels. Right margins varied from 0 to 381 pixels. In some cases, high values of the right margins were not a problem, such as the cases when the dialog box only had labels or center-aligned buttons. Top margin varied from 0 to 56 pixels and was more consistent than left and right margins. Similarly, bottom margins were more consistent than left and right margins with values clustered between 8 and 30 pixels.

Gridedness. Most dialog boxes had well aligned widgets, with low X-gridedness and Y-gridedness values (1 or 2). Some dialog boxes which had high values of gridedness (4 or more) required minor alignment changes. A small number of dialog boxes had higher values of button gridedness due to misalignment of buttons by a few pixels.

Area Balances. High balance ratios (greater than 4) were detected in few dialog boxes. These screens were often poorly designed.

Distinct Typefaces. In total, 19 distinct typefaces were used, which was high. This revealed that different designers worked on the applications without following any guidelines. We recommended that the applications be modified to use fewer typefaces.

Distinct Background and Foreground Colors. The application used 15 different colors: eight background and 10 foreground colors. We recommended more consistent use of colors.

4.3.2 Interface Concordance

A few terms that had different cases across the application, such as "Cancel," "cancel," and "CANCEL" or "Delete," and "DELETE."

4.3.3 Button Concordance

The following inconsistencies were detected by the Button Concordance tool (see Table 6):

- Designers used six distinct typefaces in button labels. Designers used three distinct foreground colors in button labels.
- Button sizes were inconsistent across the application. For example, the height of the "Cancel" button varied between 24 and 49 pixels and width varied between 57 and 122 pixels. We found inconsistencies in button sizes in "OK," "Done," "Exit," "No," "Previous," and "Start."
- Buttons like "OK," "Cancel," "Done," "Exit," and "No" used inconsistent cases across the application. Also, the designers used the button labels "Save Left" and "Save Right" in some dialog boxes and "Left Save" and "Right Save" in others.
- The button positions metric detected many inconsistencies. For example, the "Cancel" button had a different right button position for every dialog box. In the case of the "Close" button, the left position was 8 pixels in two dialog boxes and was 291 in the third, indicating that the "Close" button was left aligned in the first case and right aligned in the second case. Similar inconsistencies existed in "Done," "Exit," "OK," and other buttons.

4.3.4 Interface Spellchecker.

The spell checking tool detected abbreviations and a few misspelled terms such as: "qiz," "veryfying," "peronal" and "btrieve."

4.3.5 Terminology Baskets.

The output from the basket [Browse, Display, Find, Retrieve, Search, Select, Show, View] showed that "Display," "View" and "Show" were used in this application. Also, both "Find" and "Search" were used. Similarly the output from the basket [Cancel, Clear, Delete, Purge, Refresh, Remove] indicated that the terms "Cancel," "Delete," "Clear," "Refresh" and "Remove" were all used in the application. The use of both "Find" and "Search" was an inconsistency.

4.3.6 Button Layout Table.

The Button Layout Table revealed inconsistencies in button sizes and placement within a dialog box and across the application. For example, the button set [OK, Cancel, Exit, Help] revealed inconsistencies in the sizes of the "OK," "Cancel," and "Help" buttons. The "Cancel" and "Help" buttons were often placed next to "OK" buttons in a row, but other times stacked below the "OK" button in a column, with the distance between these buttons varying from 0 to 40 pixels. Fig. 4 shows the dialog boxes in which button placement inconsistencies of "OK" and "Cancel" buttons were detected.

4.4 Conclusion

Evaluation of the four applications using SHERLOCK helped us to determine which tools were most successful in detecting inconsistencies. The dialog box summary table had limited success in detecting inconsistencies. Certain metrics of the dialog box summary table such as aspect ratio, margins, distinct typefaces, distinct foreground and background colors were more successful in finding inconsistencies. Many of the extreme values computed by the metrics like non-widget area, widget density, and area balances were due to the limitations of SHERLOCK or the Visual Basic development tool and were not inconsistencies. These metrics were modified several times to deal with exceptions and further work is required to validate these metrics. The Button Concordance and the Button Layout Table proved to be the most useful tools and were able to detect inconsistencies in the size, position, typeface, color, and terminology used in buttons. The Interface Concordance and the Interface Spellchecker tools were successful in detecting terminology inconsistencies such as variant capitalization, abbreviations, and spelling errors. The Terminology Basket tool helped in detecting misleading synonyms. In summary, SHERLOCK was successful in detecting major terminology inconsistencies and certain inconsistencies in visual design of the evaluated interfaces.

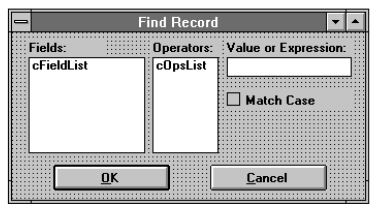
SHERLOCK is a collection of programs that requires detailed knowledge to use effectively. Additional programming and a graphic user interface would be necessary to make it viable as a widely used software engineering tool. The source code and documentation that exists are available (http://www.cs.umd.edu/projects/hcil in the FTP area under Demo software).

TABLE 6
BUTTON CONCORDANCE

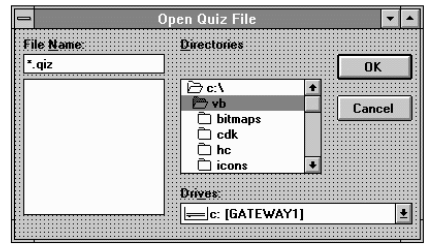
BUTTON LABEL	DIALOG BOX	BUTTON TYPEFACE	BUTTON FG_COLOR	BUTTON (H, W)	BUTTON POSITION LEFT	RIGHT	TOP
Exit	attapp94	1	2	56, 120	464	56	240
	cover	2	2	57, 153	680	182	536
	coveraf	3	2	41, 89	448	0	392
	coveruf	3	2	41, 89	448	85	392
	frmhand	4	3	49, 105	384	31	136
	frmlogin	4	3	41, 97	248	96	256
	winstat	6	1	49, 113	368	70	424
EXIT	delete	1	1	41, 97	280	45	352
	syllabus	1	1	33, 137	856	7	512
Left SAVE	feed			33, 81	272	647	448
Left Save	omp			33, 97	112	796	456
Right SAVE	feed			33, 89	648	263	448
Right Save	omp			33, 97	544	364	456
SAVE Left	mulq			33, 97	256	653	488
SAVE Right	mulq			33, 97	504	405	488

DISTINCT TYPEFACES IN BUTTONS:
1 = MS Sans Serif 8.25 Bold
2 = MS Sans Serif 18
3 = MS Sans Serif 13.5
4 = MS Sans Serif 12 Bold
5 = MS Sans Serif 9l75 Bold
6 = MS Serif 12 Bold

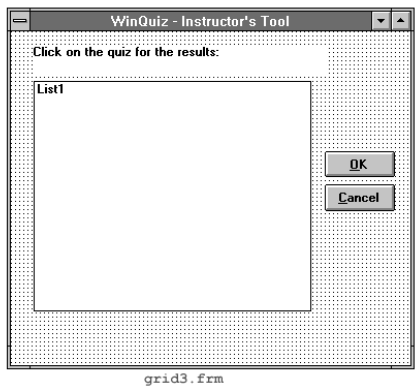
DISTINCT FOREGROUND COLORS IN BUTTONS:
1 = Default Color
2 = ffffffff80000005
3 = 0
4 = ff0000



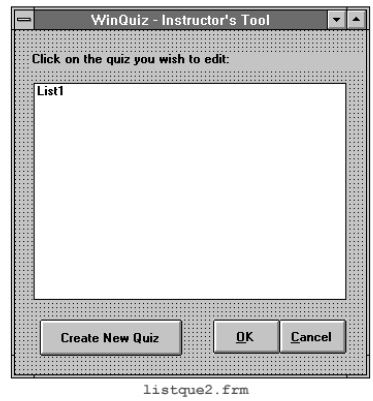
find.frm



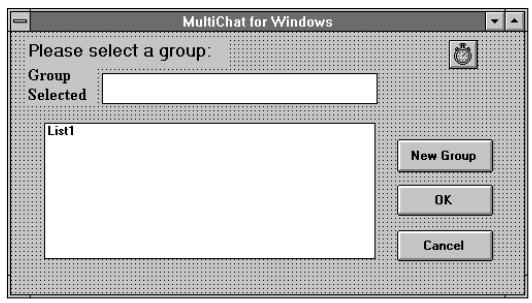
opensav.frm



grid3.frm



listque2.frm



list2.frm

Fig. 4. Button placement inconsistencies in OK and cancel buttons.

4.5 Limitations of SHERLOCK

SHERLOCK evaluations are limited to certain visual design and terminology aspects of user interfaces. Screen layout issues such as proper placement of widgets in a dialog box, violations of design constraints, and inappropriate widgets types are not evaluated by SHERLOCK. Other evaluation methods, such as usability testing and heuristic evaluation, are needed to locate typical user interface design problems such as inappropriate metaphors, missing functionality, chaotic screen layouts, unexpected sequencing of screens, misleading menus, excessive demands on short-term memory, poor error messages, or inadequate help screens.

5 Feedback from Designers

Output from the tools and the screen shots of the interface along with the analyses were forwarded to the developers and designers to elicit feedback.

5.1 GEIS Interfaces

We worked closely with the people at GE Information Services to get feedback on the effectiveness of SHERLOCK, as these tools were being iteratively refined. The feedback suggested that the outputs of the dialog box summary table were simple for the designers to interpret. They were able to detect inconsistencies by scanning down the columns for extreme values, guided by the statistical analysis at the bottom. They recommended that we develop some "goodness" measures for the metrics after analyzing more applications. We have succeeded partly in assigning measures to certain metrics after analyzing the four applications. Detailed analysis of each metric was recommended by the designers for future implementations.

The incorporation of a spell checking tool in SHERLOCK had a positive response from the designers, since none of the current GUI building environments on the PCs had one. Button typeface, terminology, and placement inconsistencies detected by Button Concordance and Button Layout Table were corrected by the GEIS designers. The Terminology Basket tool helped GEIS designers in rectifying a few terminology inconsistencies. Overall the use of SHERLOCK helped to modify the layout, visibility, and terminology of GEIS interfaces by detecting many small inconsistencies.

5.2 University of Maryland Interface

Since this application was a combination of several applications, the output was given to two design groups. Their feedback on the dialog box summary table was positive for some metrics. They showed interest in the ability of the dialog box summary table to detect the typeface and color inconsistencies in their application. When asked for an explanation of these inconsistencies, they explained that different designers worked on different portions of the application, with few guidelines on visual design. Similar reasons were given for other inconsistencies such as different aspect ratios for functionally similar screens and the use of inconsistent margins.

Designers liked the statistical analysis at the end of the table with mean, maximum, and minimum values and wanted an additional function that listed the optimum values for the metrics. Many of the terminology inconsistencies detected by the Button Layout Table and the Terminology Basket tool were valid inconsistencies that they will take into consideration in preparing the next version of the application.

6 Recommendations

6.1 Recommendations for the GUI Application Developers

The following guidelines are recommended as a step towards creating consistent interfaces:

- Aspect ratio should be consistent, especially for dialog boxes having similar visual design and functionality.
- Nonwidget area (white space) should be at least 20 percent of total area enclosed by the dialog box.
- Margins should be consistent within and across dialog boxes.
- Widgets within a dialog box should be horizontally and vertically aligned.
- Designs with many widgets in a small area should be avoided.
- Background colors, foreground colors, and typefaces should be consistent.
- Location and size of frequently used widgets should be consistent.
- Terminology should be consistent.
- Button Labels should be consistent across the interface, for example synonyms like "Abort," "Cancel," "Close," and "Exit" should not be used for similar tasks.
- In addition to these consistencies within the dialog boxes, the interface should be consistent in terminology with the current commercial applications running on that system, and in accordance with the user's task domain.

6.2 Recommendations for Designers Looking for GUI Evaluation Metrics

- Use metrics such as nonwidget area and widget density. Explore the use of other metrics for dialog box crowdedness like the Local Density [32] and Hot-Spot metric [30].
- Use metrics such as aspect ratio, margins, and balance to evaluate size of dialog boxes and create a more consistent layout.
- Explore metrics using additional visual techniques of regularity, proportion, neutrality, transparency and grouping [34].
- Develop metrics to check consistency in typefaces and colors across dialog boxes and for every widget type.
- Develop a better metric for detecting misaligned widgets similar to the gridedness metric and layout complexity and alignment metrics [32], [30].
- Develop metrics to check consistency in size and location of widget types across dialog boxes. These metrics may be similar to those used in SHERLOCK's Button Concordance and Button Layout Table tools.
- Expand the Terminology Basket tool to detect misleading synonyms for specific widget types and across all the widgets.

- Implement tools similar to Interface Concordance and Interface Speller to detect variant capitalization, spelling errors and abbreviations.

6.3 Recommendations for Designers of New GUI Tools

The applicability of SHERLOCK's canonical format to other GUI development tools beyond Visual Basic was explored. The following are recommendations for designers of new GUI tools after analyzing existing tools like Visual Basic, Visual C++, Galaxy, and Tcl/Tk.

- Store visual design and textual properties of dialog boxes including widget labels, widget coordinates, widget sizes, typefaces, and colors in an ASCII resource file. Many existing GUI development tools store this information as binary files.
- Do not allow a widget in the dialog box to extend beyond the area of the dialog box.
- Create default left, right, top and bottom margins for dialog boxes beyond which widgets may not be extended
- Promote consistency by creating default dialog box templates so that developers are aware of the positioning of frequently used widgets.
- Incorporate a spell checking tool.
- If the GUI development tool allows the user to dynamically change the widget size and position within the code, these changes should be updated to the corresponding resource files.

7 Future Directions

We recommend work on these extensions to SHERLOCK:

- Refine the dialog box summary table metrics.
- Validate the canonical format with other GUI building tools.
- Subdivide the dialog box summary table into smaller tools. This subdivision would enable reporting exceptions to facilitate the interpretation of results.
- Link the dialog box summary table to a spreadsheet program, such as Microsoft Excel, to graph the metric values.
- Add tools to detect visual design and textual inconsistencies in any widget, including combo boxes, drop down boxes, and text boxes.
- Expand the terminology thesaurus in button sets of the button layout table and baskets of the terminology basket tool.
- Enable users to fix inconsistencies in margins, aspect ratio, typefaces and colors by mapping the canonical format files back to the Visual Basic .frm files.

SHERLOCK is a foundation for the next generation of GUI consistency checking tools. More work needs to be done to build a complete structure on this foundation.

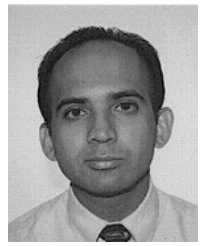
Acknowledgments

Funding for this research was provided by the GE Information Services and the Maryland Industrial Partnership Program. We would like to thank Ren Stimart at GE Information Services for his help and support and Ninad Jog for his early work on this project. We appreciate the efforts of the staff of the AT&T Teaching Theater at the University of Maryland and the Italian company, Sogei, for providing test applications. We thank American Management Systems, Microsoft, and NASA for inviting us to present this work, and for their supportive feedback.

References

[1] Apple Computer Inc., *Macintosh Human Interface Guidelines.* Reading, Mass.: Addison-Wesley, 1992.
[2] S. Bajwa, "Effects of Inconsistencies on Users Performance and Subjective Satisfaction," unpublished report, Dept. of Computer Science, Univ. of Maryland, 1995.
[3] P. Barnard, J. Hammond, J. Morton, J. Long, and I. Clark, "Consistency and Compatibility in Human-Computer Dialogue," *Int'l J. Man-Machine Studies,* vol. 15, pp. 87-134, 1981.
[4] F. Bodart, A.-M. Hennebert, J.-M. Leheureux, and J. Vanderdonckt, "Towards a Dynamic Strategy for Computer-Aided Visual Placement," T. Catarci, M. Costabile, S. Levialdi, and G. Santucci, eds., *Proc. Advanced Visual Interfaces Conf.,* pp. 78-87, New York: ACM Press, 1994.
[5] R. Chimera and B. Shneiderman, "User Interface Consistency: An Evaluation of Original and Revised Interfaces for a Videodisk Library," B. Shneiderman, ed., *Sparks of Innovation in Human-Computer Interaction,* pp. 259-273, Norwood, N.J.: Ablex Publishers, 1993.
[6] J.P. Chin, V.A. Diehl, and K. Norman, "Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface," *Proc. Conf. Human Factors in Computing Systems, CHI '90,* New York: ACM, pp. 213-218, 1988.
[7] R. Coll, and A. Wingertsman, "The Effect of Screen Complexity on User Preference and Performance," *Int'l J. Human-Computer Interaction,* vol. 2, no. 3, pp. 255-265, 1990.
[8] T. Comber and J. Maltby, "Investigating Layout Complexity," J. Vanderdonckt, ed., *Computer-Aided Design of User Interfaces,* Press Universitaires de Namur, Namur, Belgium, pp. 209-227, 1996.
[9] N. Frederiksen, J. Grudin, and B. Laursen, "Inseparability of Design and Use: An Experimental Study of Design Consistency: *Proc. Computers in Context'95, Aarhus,* Aarhus Univ., pp. 83-89, 1995.
[10] J. Grudin, "The Case against User Interface Consistency," *Comm. ACM,* vol. 32, no. 10, pp. 1,164-1,173, New York: ACM Press, 1989.
[11] J. Grudin and D. Norman, "Language Evolution and Human-Computer Interaction," *Proc. 13th Ann. Conf. Cognitive Science Soc.,* Hillsdale, N.J., pp. 611-616, 1991.
[12] M.D. Harrison and H.W. Thimbleby, "Formalizing Guidelines for the Design of Interactive Systems," *Proc. BCS HCI Specialist Group Conf. HCI85,* pp. 161-171, 1985.
[13] R. Jeffries, J. Miller, C. Wharton, and K. Uyeda, "User Interface Evaluation in the Real World: A Comparison of Four Techniques," *Proc. CHI '91,* pp. 119-127, New York: ACM, 1991.
[14] C.-M. Karat, "Cost-Justifying Human Factors Support in Development Projects," *Human Factors Soc. Bull.,* vol. 35, no. 8, 1992.
[15] W. Kellogg, "Conceptual Consistency in the User Interface: Effects on User Performance," *Proc. Interact '87 Conf. Human-Computer Interaction,* Stuttgart, Germany, 1987.
[16] P. Lynch, "Visual Design for the User Interface, Part 1 Design Fundamentals," *J. Biocommunications,* vol. 21, no. 1, pp. 22-30, 1994.
[17] F. MacIntyre, K.W. Estep, and J.M. Sieburth, "Cost of User-Friendly Programming," *J. Fourth Application and Research,* vol. 6, no. *2,* pp. 103-115, 1990.
[18] R. Mahajan and B. Shneiderman, "A Family of User Interface Consistency Checking Tools: Design Analysis of SHERLOCK, *Proc. NASA 20th Ann. Software Eng. Workshop,* pp. 169-188, 1995.
[19] Microsoft, Inc., *The Windows Interface: An Application Design Guide.* Redmond, Wash.: Microsoft Press, 1992.
[20] K. Mullet, "Organizing Information Spatially," *Interactions,* pp. 15-20, July 1995.
[21] H. Nielsen, "Coordinating User Interfaces for Consistency Checking," J. Nielsen, ed., London: Academic Press, 1989.

[22] P. Polson, E. Muncher, and G. Engelbeck," A test of a common elements theory of transfer, *Proc. CHI '86*, pp. 78- 83, New York: ACM, 1986.
[23] P. Reisner, "What is Consistency?" *Proc. IFIP Third Int'l Conf. Human-Computer Interaction, Interact '90*, pp. 175-181, Elsevier Science, B.V., North-Holland, 1990.
[24] D. Rosenberg, "Cost Benefit Analysis for Corporate User Interface Standards: What Price to Pay for Consistent Look and Feel?" *Coordinating User Interfaces for Consistency Checking*, J. Nielsen, ed., pp. 21-34, London: Academic Press, 1989.
[25] A. Sears, "Layout Appropriateness: A Metric for Evaluating User Interface Widget Layouts," *IEEE Trans. Software Eng.*, vol. 19, no. 7, pp. 707-719, 1993.
[26] A. Sears, "AIDE: A Step Towards Metric-Based Interface Development Tools," *Proc. UIST '95*, pp. 101-110, New York: ACM, 1994.
[27] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Third Edition.* Reading Mass.: Addison-Wesley, 1998.
[28] B. Shneiderman, R. Chimera, N. Jog, R. Stimart, and D. White, "Evaluating Spatial and Textual Style of Displays," L.W. MacDonald and A.C. Lowe, eds., *Display Systems: Design and Applications.* Chichester, U.K.: John Wiley & Sons, pp. 83-96, 1997.
[29] D.C. Smith, C. Irby, R. Kimball, B. Verplank, E. Harslem, "Designing the Star User Interface," Byte 7, no. 4, pp. 242-282, 1982.
[30] D. Streveler and A. Wasserman, "Quantitative Measures of the Spatial Properties of Screen Designs," *Proc. INTERACT '87*, Amsterdam: Elsevier Science, pp. 125-133, 1987.
[31] T.S. Tullis, "The Formatting of Alphanumeric Displays: A Review and Analysis," *Human Factors*, vol. 25, pp. 657-682, 1983.
[32] T.S. Tullis, "A System for Evaluating Screen Formats: Research and Application," Hartson, H. Rex and Hix, Hartson, ed., *Advances in Human-Computer Interaction:* vol. 2. Norwood N.J.: Ablex Publishing Corp., pp. 214-286, 1988.
[33] T.S. Tullis, "Screen Design," M. Helander, T. Landauer, and P. Prabhu, eds., *Handbook of Human-Computer Interaction: Second Edition*. Amsterdam, The Netherlands: Elsevier Science, pp. 503-531, 1997.
[34] J. Vanderdonckt and X. Gillo, "Visual Techniques for Traditional and Multimedia Layouts," T. Catarci, M. Costabile, S. Levialdi, and G. Santucci, eds., *Proc. Advanced Visual Interfaces Conf. '94*, New York: ACM Press, pp. 95-104, 1994.
[35] C. Wiecha, W. Bennett, S. Boies, and J. Gould, "Generating Highly Interactive User Interface," *Proc. CHI '89*, pp. 277-282, New York: ACM, 1989.
[36] R. Wolf, "Consistency as Process," *Coordinating User Interfaces for Consistency Checking*, J. Nielsen, ed., pp. 89-92, London: Academic Press, 1989.



Rohit Mahajan received a BS degree in electrical engineering in 1993 and an MS degree in systems engineering in 1996, both from the University of Maryland at College Park. Mr. Mahajan is a software engineer at BDM International in McLean, Virginia. Mr. Mahajan was a graduate research assistant in the Human-Computer Interaction Laboratory, University of Maryland from 1994–1996.

Ben Shneiderman received his BS degree from City College of New York in 1968 and his PhD degree from the State University of New York at Stony Brook in 1973. He received an honorary doctorate of science from the University of Guelph, Ontario, Canada in 1996. He is a professor in the Department of Computer Science, head of the Human-Computer Interaction Laboratory, and a member of the Institutes for Advanced Computer Studies and for Systems Research, all at the University of Maryland, College Park. Dr. Shneiderman is the author of *Software Psychology: Human Factors in Computer and Information Systems* (1980) and *Designing the User Interaface: Stategies for Effective Human Computer Interaction* (1987, second edition 1992, third edition 1998) Addison-Wesley, Reading, Massachusetts. Dr. Shneiderman was elected a fellow of the Association of Computing Machinery in 1997.