CAR-TR-770
CS-TR-3472
ISR-TR-95-52

April 1995

A Family of User Interface Consistency Checking Tools

Rohit Mahajan and Ben Shneiderman[1]

Human-Computer Interaction Laboratory
Center for Automation Research
[1]Department of Computer Science
Institute for Systems Research
University of Maryland, College Park, MD 20742-3255 USA
email: mahajan@cs.umd.edu, ben@cs.umd.edu

ABSTRACT

Incorporating evaluation metrics with GUI development tools will help designers create consistent interfaces in the future. Complexity in design of interfaces makes efficient evaluation impossible by a single consistency checking evaluation tool. Our focus is on developing a family of evaluation tools in order to make the evaluation process less cumbersome. We have developed a dialog box typeface and color table to facilitate detection of anomalies in color, font, font size, and font style. Concordance tools have been developed to spot variant capitalization and abbreviations globally in the interface and specifically in the button widgets. As buttons are frequently used widgets, a button layout table has been created to spot any inconsistencies in height, width and relative position between a given group of buttons if present. Finally, a terminology basket tool has been created to identify unwanted synonyms of computer related terms used in the interface which may be misleading to the end user.

KEYWORDS: Automated metrics, consistency checking tools, concordance tools, spatial and textual evaluation tools, user interface

Proc. of the Twentieth Annual Software Engineering Workshop, SEL-95-004 (Greenbelt, MD, Dec. 1995) NASA Pub., 169-188.

1. INTRODUCTION AND PREVIOUS RELATED RESEARCH

Creating user interfaces is a composite procedure involving iterative design, usability testing and evaluation processes (Shneiderman, 1992). Iterative refinement methods like Formative Evaluations can be used to design/redesign the interface from early development stages through completion stage (Hix & Hartson, 1993). Interactive tools like IDEAL (Interface Design Environment Analysis Lattice) support procedures like Formative Evaluations (Ashlund & Hix, 1992). Recent advances in powerful user interface development tools have expedited the interface development process helping both novice and experienced developers. However these expeditiously created designs may be clogged with spatial and textual inconsistencies that may have a subtle and negative impact on interface usability.

Inconsistencies in spatial and textual style of an interface designed by several designers' may result in a chaotic layout. Each designer may have different interpretation of terminology and uses his/her own style of abbreviations and computer terms. Furthermore designers personal preferences on fonts and colors add to the problem in group designs. Such anomalies in terminology and format lead to poor design, ultimately misleading and confusing the user (Chimera & Shneiderman, 1993). Although many organizations are adopting more stringent usability testing standards to monitor quality and layout of the design, better automated evaluation tools are needed which would scan for inconsistencies in the interface layout at early design and development stages thereby decreasing the complexity of usability testing.

Usability testing is a highly beneficial but costly process when compared with automated evaluation. Prerequisites for these tests may include availability of developed working prototypes, test users and expert evaluators (Sears, 1994). These requirements are hindrances in this very powerful evaluation method. Alternative techniques like Heuristic Evaluations (Nielsen & Molich, 1990) can decrease but not eliminate these requirements. Furthermore usability testing works best for smaller applications. It is practically impossible to analyze every dialog box in an application with thousands of dialog boxes. Finding anomalies or differences while reviewing thousands of dialog boxes is even hard for expert reviewers who may leave undetected flaws and inconsistencies. In contrast automated evaluation tools can be used in early prototypes (or later iterations) and can detect anomalies across thousands of dialog boxes.

Automated evaluation tools can be made independent of platform and development tool, as textual and spatial properties are independent of these constraints. Research in automated design and evaluation tools in recent years has resulted in development of first generation test models and systems based on metrics. Spatial metrics to check consistencies in alignment, screen symmetry, screen balance, average distance between groups of items, percentage of screen used to display information, average size of groups of items were introduced by Strevler and Wasserman (1987) and were later implemented by Tullis(1988). Furthermore Kim and Foley (1993) used metrics as a constraint for design space and layout style. They developed a tool which generated potential designs for an interface when provided with design specifications and guidelines for metrics. Effectiveness of their metrics has not yet been evaluated.

Evolution of modern user interfaces like multimedia interfaces has sparked research in automated evaluation based on visual techniques. Vanderdonckt and Gillo (1994) proposed five visual techniques: physical, composition, association, ordering and photographic which identified more spatial properties than traditional balance, symmetry and alignment. These visual properties also include proportion, neutrality, singularity, repartion, grouping, sparing, simplicity etc. Dynamic strategies for automated evaluation using these visual techniques have been introduced. (Bodart, Hennebert, Leheureux and Vanderdonckt,1994). Visual metrics introduced above for traditional layout grids and multimedia layout frames have not yet been tested.

Sears (1993, 1994) has developed a first generation tool using automated metrics for both design and evaluation using Layout Appropriateness metrics. The tool AIDE (semi-Automated Interface Design and Evaluator) allows designers to create, evaluate and modify an interface using a single tool. Layout Appropriateness compares layout based on user's task sequences and frequencies. AIDE has demonstrated its effectiveness in analyzing simple interfaces.

2. METRICS EVALUATION USING CANONICAL FORMAT

Our research evolved from the concept of converting interface form files generated by Visual Basic into canonical format files and feeding them as input to the evaluation program. The canonical format is an organized set of GUI object descriptions. These object descriptions are enclosed in curly braces and embrace information in a sequence of attribute-value pairs. The canonical format is advantageous because of its lucidity and extendibility characteristics. It can be easily modified to include any new attribute encompassing interface description information in the form files.

A metrics evaluation program takes the canonical format file and first does preprocessing for two basic reasons, to extract relevant information and possibly to expand objects' descriptions (e.g., absolute coordinates, sibling information, etc.). Any class hierarchy or other related specifications are not part of the canonical format. Such things exist as separate documents.

The canonical format file uses unique keys for all objects that are constructed via parent-self ID pairs. (If a parent-self ID pair is not unique, then all non-unique occurrences get enlarged to be parent-parent-self ID triples, and so on until uniqueness is achieved.) The attribute called "unique-id" is provided in the event that the originating system has the capability to supply unique numbers for objects. If not supplied, this value can be filled in by expansion programs. Each object is completely self contained and has no information about child objects, except for menus. Menu objects describe their child objects within the menu's own description in a hierarchical fashion. A generic object can list any of the following attribute-value pairs. Those marked with an asterisk (*) are required.

```
{*type                            Object-value
 *parent                   ID
 *resource-id                     String
  name                            String
  variable-reference              String
  unique-id                Number
 *top                      Coordinate (relative to parent)
 *left                            Coordinate (relative to parent)
 *width                    Coordinate
 *height                   Coordinate
 *unit-of-measure-width    Unit-value
 *unit-of-measure-height   Unit-value
 *border-is-visible        Boolean
 *background-color         Color
 *foreground-color         Color
  font-family                     String
  font-size                Number
  font-is-bold             Boolean
  font-is-italic           Boolean
  font-is-underline        Boolean
  font-is-serif            Boolean
  label-type               Label-value
  label                    String
  label-background-color   Color
  label-foreground-color   Color
  label-justification      Justification-value
  label-placement          Placement-value
  label-font-family        String
  label-font-size          Number
  label-font-is-bold              Boolean
  label-font-is-italic     Boolean
  label-font-is-underline  Boolean
  label-font-is-serif      Boolean
  margin-top               Coordinate (relative to attribute "top")
  margin-left                     Coordinate (relative to attribute "left")
  margin-bottom            Coordinate (relative to calculated "bottom")
  margin-right             Coordinate (relative to calculated "right")
  navigation-order         Number
  navigation-type          Navigation-value
  help-text                String
  is-default               Boolean (uses window origin)
  is-cancel                Boolean (uses window origin)
  is-help                  Boolean (uses window origin)
  mnemonic                 Character
  menu-accelerator         Key-value (type Menu-item only)
  menu-is-pinnable         Boolean (type Menu only)
  menu-content             Menu-value (big hierarchical description)
  task                            String
  comments                 String
}
```

We feel that attribute names are self-explanatory, but many of the value types appearing in the canonical format are abstractions. The following table specifies value types that are not completely obvious.

```
Number ::= nonnegative integer
ID ::= String | Number
Coordinate ::= real number (uses top-left origin)
Unit-value ::= Unit-pixel | Unit-point | Unit-MSWindows dialog-unit | ...
Color ::= String | Color-specification
Color-specification ::= RGB-triple | HSB-triple
Label-value ::=  String | Icon | String-and-icon
Justification-value ::= Justify-left | Justify-center | Justify-right
Placement-value ::= Place-left | Place-right | Place-top | Place-bottom
Navigation-value ::= Nav-parent | Nav-window (whether navigation is
                    relative to parent or entire window)
Character ::= a single alphanumeric character value
Modifier ::= Shift-key | Control-key | Alt-key
Key-value ::= Character + Modifier | Function-key + Modifier
Object-value ::= Window | PopupWindow | ModalPopupWindow |
                PushButton | CheckButton | RadioButton |
                Container | StaticLabel |
                ScrollingListOneSelection | ScrollingListMultipleSelection
                ScrollingListContiguousSelection |
                MenuHighestLevel | Menu | MenuItem | MenuSeparator |
                TextPane | TextPaneReadonly |
                TextPaneOneLine | TextPaneOneLineReadonly | ....
                (basically responsibility of metric conversion
                program to know about these values)
Menu-value ::= hierarchical object description of only
              Object-types  Menu | MenuItem | MenuSeparator
```

Our evaluation tools convert the Visual Basic form files into canonical format through a translator which distills relevant information from the form files. These canonical formats are platform independent and may be created for other interface tools like Power Builder, Galaxy and XVT by writing a translator program for these tools. So our evaluation programs are not specific to Visual Basic and can be used for evaluating interfaces developed by other tools.

2.1 Our Previous Method

This research is an extension of previous work (Shneiderman, Chimera, Jog, Stimart and White, 1995) in which we developed spatial and textual evaluation tools. The spatial tool was a dialog box summary table which gave an overview of spatial and visual properties. Each dialog box corresponded to a distinct row and each column a metric. The metrics Aspect Ratio, Widget Totals, Non-Widget Area, Widget Density, Margins, Griddedness, Top-Bottom Balance, Left-Right Balance and Distinct Typefaces formed our metrics column set. This list of metrics was developed by consultation between analysts at University of Maryland and General Electric Information Services to evaluate categories such as spatial layout, alignment, clustering, cluttering, fonts, etc. The textual tool was a concordance built to extract all the words that appear in labels, menus, buttons, etc. in every dialog box. These words were sorted in one file with reference to the dialog boxes containing them. The concordance was to help designers in appropriate word use such as spelling, abbreviation, tense consistency, case consistency, passive/active voice etc.

The objective of developing the dialog box summary table and the concordance was to provide interface designers rapid feedback on possible flaws and inconsistencies from early prototyping to advanced development stages. The gist of the tools is to help designers spot inconsistencies by providing a compact overview. Thus, assisting them in determining which screens needed redesign/modifications. Our programs allowed the designers to check the dialog boxes for inconsistencies and the output of the dialog box summary table revealed interesting anomalies. For example the margins were irregular and aspect ratio (ratio of the height of a dialog box to its width) was surprisingly variant. The non-widget area (ratio of the non-widget area to the total area of the dialog box, expressed as a percentage) and widget density (number of top level widgets divided by the total area of the dialog box and multiplied by 100,000 to normalize it) varied from single digits to lower hundreds. Numbers closer to 100 and higher for Non-Widget Area indicated high screen utilization and for Widget Density indicated crowding of widgets. The concordance table revealed terms used in the interface which had variant forms including capitalization and abbreviations. The results were useful but required too much interpretation. The concordance report being twenty pages long made it hard for designers to spot anomalies quickly and efficiently. The dialog box summary table was also too cumbersome to interpret inconsistencies at a glance.

2.2 Modification of Previous Method

Our new approach is to modify the large dialog box summary table and the concordance tools by dividing them into smaller tools, plus adding new tools. The family of consistency checking tools was constructed by modifying our previous tools. These new tools perform exception reporting by outputting the possible anomalies and irregularities in textual layout. The reports generated by these mini tools require little interpretation, thereby expediting the quick evaluation process and providing feedback to the designer. The designer then must decide whether the spotted inconsistencies are relevant to the particular prototype. We have developed six consistency checking tools:

- dialog box typeface and color table to spot any anomalies in color, font, font size and font style.
- interface concordance to spot variant capitalization and abbreviation in the interface.
- button concordance to spot variant capitalization and abbreviation in button widgets.
- button layout table to spot any inconsistencies in height, width and relative position among a given group of buttons.
- interface speller to detect terms used in the interface that are nonexistent in the dictionary.
- terminology basket to provide the interface designer with the feedback on misleading synonym computer terms.

2.2.1 Dialog Box Typeface and Color Table

The dialog box typeface and color table was developed to provide designers feedback on inconsistencies in fonts and colors, two of the most striking appearance features of the interface. Each row represents a single dialog box and the columns represent dialog name, distinct typefaces and distinct background colors.

Dialog Name: Name of the file in which dialog is contained.

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 and is described in detail at the end of the table. 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 among all the dialog boxes. The idea is that a small number of typefaces should be

used for all the dialog boxes. Occurrence of multiple typefaces within a dialog box is not desired.

Distinct Background Colors:
All the distinct background colors 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 all the dialog boxes have the same background colors. Multiple background colors in a dialog box may indicate inconsistency.

This tool was tested with two interfaces containing 30 (small interface) and 140 (large interface) dialog boxes respectively. Inconsistencies in fonts and colors were revealed in both the test interfaces. The results with the smaller interface were surprising. The designers used 17 distinct typefaces and 6 distinct colors in the small interface with only 30 dialog boxes. These inconsistencies were brought to the attention of the designers who were amazed by the results. A portion of the table is shown below:

```
No.   Dialog Name          Distinct Typefaces         Distinct Background Colors
  1    cover.cft                1 2 3 4 5 6                         1
  2    coveraf.cft                1 3 4 7 8                         1
  3    coveruf.cft                1 3 4 7 8                         1
  4    feed.cft                       3 7 9                               2 3
  5    frmcompaz.cft                           3 5                        1
  6    frmcompu.cft                     3 5                         1
  7    frmhand.cft                      3 5                         1
  8    frmlogin.cft                     3 5                               1
  9    frmlogo.cft                3 9 10 11                         1 4 5
 10    frmmatch.cft                       9                         1
```

```
DISTINCT TYPEFACES:
1 = MS Sans Serif 13.5                  2 = Symbol 13.5 Bold
3 = MS Sans Serif 12 Bold               4 = MS Sans Serif 24 Bold Italic
5 = MS Sans Serif 13.5 Bold             6 = MS Sans Serif 18
7 = MS Sans Serif 9.75 Bold             8 = Symbol 9.75 Bold
9 = MS Sans Serif8.25 Bold                     10 = Arial 18 Bold
11 = Symbol 8.25 Bold                   12 = MS Sans Serif 16.5
13 = Times New Roman 24 Bold Italic            14 = Times New Roman 30 Bold
Italic
15 = System 9.75 Bold                   16 = MS Sans Serif 9.75 Bold Italic
17 = MS Sans Serif 8.25
```

```
DISTINCT BACKGROUND COLORS:

1 = ffffffff80000005       2 = c0c0c0   3 = e0ffff    4 = ffffff    5 = c000006
```

2.2.2 Interface Concordance

The interface concordance tool checks for variant capitalization for all the words that appear in buttons, labels , menus, etc. in every dialog box of the interface. This tool outputs strings which have variant capitalization, listing all the variant formats of the string and its dialog box sources. These variant forms are spelling differences and may be acceptable, but they may be something that should be reconsidered. For example the words "Item" , "items" and "Items:" are Variant Capitalization forms of the same word. A portion of the concordance output is shown below:

```
Items
         reconly.cft    reconly.cft     reconly.cft     reconly.cft
         sendrec.cft    sendrec.cft     sendrec.cft     sendrec.cft
Items:
         moreinfo.cft
items
         wastedef.cft
Ship
         create.cft     create.cft      dp.cft          dp.cft
         invoice.cft    po.cft          po.cft          po.cft
SHIP
         invoice.cft
Execute
          exnow.cft     sched.cft       sched.cft       sched.cft
          sched.cft     sview.cft
Execute:
          scriptor.cft
```

2.2.3 Button Concordance

This tool is a further filtration of the interface concordance. As buttons are one of the most frequently used widgets performing vital functions like "Save", "Open", "Delete", "Exit" checking variant capitalization in them becomes more important. The button concordance tool checks for variant capitalization for all the words that appear in buttons in every dialog box of the interface. This tool outputs button labels which have variant capitalization, listing all the variant formats and their dialog box sources. Considerable variant capitalization in button labels was detected in both our test interfaces. The most frequently used button labels such as "OK", "Cancel", "Exit" were not consistently used in the same case. A portion of the button concordance output is shown below:

```
Cancel
        frmcompaz.cft   frmcompu.cft    frmquesaz.cft   frmquesu.cft
CANCEL
        frmmatch.cft    l_login.cft     passwd.cft      r_login.cft
DONE
        feed.cft        graph.cft       ibm_az.cft      ibm_um.cft

        infoas.cft      infoums.cft     mulq.cft        omp.cft
Done
        info2ums.cft
Exit
```

```
        cover.cft       coveraf.cft     coveruf.cft     frmhand.cft
        frmlogin.cft
EXIT
        syllabus.cft
```

2.2.4 Button Layout Table

Given a set of buttons that frequently occur together (e.g. OK Cancel Help), if the first button in the set is detected in the dialog box then the program prints 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 outputted as (x ± offset, y± offset) to the first button, where offset is in pixels. Buttons stacked in rows would yield (x± offset, y) relative position and those stacked in columns would yield (x, y± offset). These button columns enable us to spot highly inconsistent sizes and relative positions of buttons within a set. Dividing the button analysis into a family of button sets expedites inconsistency checking process. Designers can determine inconsistencies while browsing each button set output in a single glance.

Our program reads an ASCII file containing different sets of buttons. These button sets were constructed after analyzing many previously developed interfaces. Variations in terminology were considered while constructing these button sets. Button set (Start Stop Exit) is incomplete as designers may use "Close" , "Done" or "Cancel" instead of "Exit". The set (Start Stop End Pause Halt Exit Done Cancel Close) forms a much better button detector set. The button sets may be easily updated as more interfaces are analyzed in the future. Some of the sample button detector sets are:

```
•  OK       Cancel      Help
•  Start    Stop        End       Pause      Halt      Exit       Done
   Cancel Close
•  Cut      Copy        Paste
•  Add      Remove      Delete    Copy       Clear     Cancel     Close    Exit
•  Help     Close       Cancel    Exit
```

Output of the button detector set (Add Remove Delete Copy Clear Cancel Close Exit) tested with the larger prototype is shown below. Inconsistency in height and relative button positions within a button set can be checked by moving across the rows of the table. Inconsistency in height and relative position for a given button can be spotted by moving down in columns. For example, the height of all the "Add" buttons are constant (25 pixels) but the width varies from 65 pixels to 97 pixels. Also, the relative position between "Add" and "Remove" buttons varies in all the three files in which they occur together. In the files "archive.cft" and "autoff.cft" the "Remove" button is 15 pixels and 39 pixels down respectively from the "Add" button, but in the file "dp.cft" the buttons occur next to each other in the same row. Also, both the buttons "Remove" and "Delete" have been used with the button "Add" which is a terminology inconsistency.

```
           Add       Remove          Delete          Cancel          Close
          (H,W)  (H,W)  Rel. Pos.  (H,W)  Rel. Pos.  (H,W) Rel. Pos.  (H,W)
Rel. Pos.

archive.cft
          25,65   25,65  x, y+15                  25,73  x-9,y+151
autoff.cft
          25,73   25,73  x, y+39                                    25,73  x, y+71
dp.cft
          25,97   25,89  x+1,y                                      25,81  x+351,y
famdef.cft
          25,89                 25,89  x+1, y                       25,97  x+87, y
standard.cft

          25,89                 25,89  x+1, y                       25,89  x+175,y
```

2.2.5 Interface Speller

Interface Speller is a tool which reads all the terms used in widgets including menus, buttons, list boxes, combo boxes etc. throughout the interface and outputs words 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 the end users. The file may also contain proper names, esoteric words or computer words which are valid computer science terms, but are not found in the dictionary.

Interface speller was tested on the two prototypes outputting words not found in the dictionary and their corresponding dialog boxes. The tool detected many misspelled, incomplete and abbreviated words such as "App", "Trans", "Quik", "Provence", "Interchg" which are spelling errors or potentially confusing abbreviations. A small portion of the output is shown below:

```
addfamdf.cft
                Doc             Msg
addr.cft
                App             EDI             HDLR            UNDA
       WDGT
admpwd.cft
                ADMIN           EDI
contacts.cft
                Provence        Quik
docsearc.cft
                ILOG            Interchg
dp.cft
                NAD             Trans
```

```
profile.cft
                Ctrl            FAX             Provence
```

2.2.6 Terminology Baskets

A terminology basket is a collection of computer task terms including their different tense formats which may be used as synonyms by the 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 synonym computer terms, e.g. Search, Retrieve, Query and Select.

Our program reads an ASCII file containing the basket list. The baskets are sorted alphabetically and for each basket all the dialog boxes containing any of the basket terms are outputted. Occurrence of labels in different cases are not preserved as unique occurrences of terms so the labels like "VIEW", "View" and "view " are considered the same. Any punctuation characters are stripped before comparing the labels and the basket terms. The list of baskets may be easily updated as more interfaces are analyzed in the future. 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 Refreshing
- Execute Executes Executed Executing Run Runs Running Start Starts Started Starting Enable Enables Enabled Enabling Begin Begins
- Item Items Entry Entries Record Records Segment Segments Segmented Segmenting Field Fields
- Add Adds Added Adding Insert Inserts Inserted Inserting Create Creates Creating
- Message Messages Note Notes Letter Letters Comment Comments

Our basket browser revealed some interesting terminology anomalies after analyzing the large interface that led to reconsideration of designs. As shown below terms like "record", "segment", "field" and "item" were used in similar context in different dialog boxes. Other interesting anomalies included use of "start", "execute" and "run" for identical tasks in different dialogue boxes.

```
---------------------------------------------------------------------------------
------Entries      Entry        Field        Fields      Item       Itemized
Itemizing       Items           Record       Records      Segment     Segmented
Segmenting   Segments
---------------------------------------------------------------------------------
------
Field
        search.cft
```

```
Items
        reconly.cft             reconly.cft             reconly.cft
        reconly.cft             sendrec.cft             sendrec.cft
        sendrec.cft             sendrec.cft             wastedef.cft
Record
        ffadm.cft               profile.cft
Segment
        addr.cft                search.cft


--------------------------------------------------------------------------------
------
Enable    Enabled    Enables    Enabling    Execute   Executed   Executes
Executing
Run       Running    Runs       Start       Started   Starting   Start
--------------------------------------------------------------------------------
------
Enable
        admpwd.cft              admpwd.cft              eepwd.cft
        logger.cft              preferen.cft            preferen.cft
Execute
        exnow.cft               sched.cft               sched.cft
        sched.cft               sched.cft               scriptor.cft
        sview.cft
Run
        sched.cft
Start
        addr.cft                docsearc.cft
```

3. Testing Our Evaluation Tools

Effectiveness of these consistency checking tools has been determined by evaluating two commercial prototype applications developed in Microsoft Visual Basic. Our testing method incorporates a sequence of steps beginning with applying the tools to the prototype application followed by analysis and review of the interface screen shots and outputs generated by our tools. Furthermore test results and interpretations were shown to developers to elicit feedback and reactions. One prototype application was a 140 dialog box Electronic Data Interchange Interface developed at GE Information Services and the other was a small 30 dialog box interface developed at the University of Maryland for teaching courses remotely via networked PC's. Our evaluation tools were not created with reference to any particular test prototype and can evaluate any interface that is converted to the canonical format.

The interface textual and spatial descriptions of the prototypes were inputted to the evaluation tools in canonical format after running the translator program on the Visual Basic form files. To facilitate analysis of the prototypes, screen shots of the interface dialog boxes were printed. The results generated by these family of consistency checkers show possible anomalies or irregularities in textual and spatial layout of the interface. These evaluation tools act as consistency patrollers reporting exceptions and anomalies, making interpretation easier for the developers. Developers at the University of Maryland, College of Business & Management were surprised to know that their 30 dialog box interface had 17 distinct typefaces plus terminology inconsistencies in button labels. It was discovered that these inconsistencies occurred as the interface was created by two developers. Developers at GE Information Services are adapting the reports to fit their development environment and to ensure that their internal guidelines are being adhered to. They plan to apply these automated evaluation tools to all their projects. Our consistency checking team is in the process of testing more complex commercial prototypes created by GE Information Services.

4. Limitations

Our evaluation tools are designed to aid the interface evaluation process by providing a compact overview of possible inconsistencies and anomalies on certain textual and spatial characteristics of the interface. The designer must decide what to do, if anything with these possible inconsistencies. Certain issues like efficiency in screen layout including proper placement of widgets on the dialog box, violation of any design constraints, use of inappropriate widgets types are not evaluated by our tools. 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, confusing terminology, chaotic screen layouts, unexpected sequencing of screens, misleading menus, excessive demands on short-term memory, poor error messages, or inadequate help screens. Currently, the evaluation is limited to Visual Basic applications, but any experienced programmer can write a translator to convert interface form files created by other development tools to a canonical format read by our evaluation tools.

5. Future Directions

Currently, the printouts provided by our tools showing the possible anomalies and inconsistency patterns need to be compared manually with the interface dialog boxes. Checking back and forth between the printouts and dialog boxes to make corrections can be time consuming for large interfaces. It would be good to have these mini evaluation tools in the form of interactive evaluation and modification tools. This would help developers to interactively make changes to the prototype while creating it rather than amassing printouts. In the future, we plan to incorporate the canonical format file translator and the evaluation tools into a single tool in Visual Basic. We also plan to diversify the metrics set of our evaluation tools to perform more detailed interface evaluation.

Acknowledgments

We appreciate the support for this project from GE Information Services and the Maryland Industrial Partnerships program. We thank Ren Stimart and David White at GE Information Services for their help in every part of this project. We thank Mun Yi at University of Maryland, College of Business & Management for providing the test prototype. We are grateful for draft comments from Richard Chimera, Catherine Plaisant, Anne Rose, Andrew Sears and Zhijun Zhang. We thank Eser Kandogan and Adil Rajput for providing programming assistance.

References

Ashlund, S. and Hix, D. (1992), "IDEAL: A tool to Enable User- Centred Design", *Proc. of CHI' 92* (*Posters and short talk supplement to proceedings*) , ACM, New York, 119-120.

Bodart, F., Hennebert, A.-M., Leheureux, J.-M., and Vanderdonckt, J. (1994), "Towards a dynamic strategy for computer-aided visual placement", In Catarci, T., Costabile, M., Levialdi, S., and Santucci, G. (Editors), *Proc. Advanced Visual I nterfaces Conference '94,* ACM Press, New York, 78-87.

Chimera, R. and Shneiderman, B. (1993), "User interface consistency: An evaluation of original and revised interfaces for a videodisk library", In Shneiderman, B. (Editor), *Sparks of Innovation in Human-Computer Interaction* , Ablex Publishers, Norwood, NJ, 259-271.

Hix, D. and Hartson, H. R. (1993), *Developing User Interfaces: Ensuring Usability Through Product & Process*, John Wiley & Sons, New York, NY.

Kim, W. and Foley, J. (1993), "Providing high-level control and expert assistance in the user interface presentation design", *Proc. of CHI'93*, ACM, New York, 430-437.

Nielsen, J. and Molich, R., (1990) "Heuristic evaluation of user interfaces", *Proc. of CHI'90* , ACM, New York, 249-256.

Sears, A. (1993), "Layout Appropriateness: A metric for evaluating user interface widget layouts", *IEEE Transactions on Software Engineering* 19, 7, 707-719.

Sears, A. (1994), "Using automated metrics to design and evaluate user interfaces", DePaul University Dept of Computer Science Technical Report #94-002, Chicago, IL.

Shneiderman, B. (1992), *Designing the User Interface: Strategies for Effective Human-Computer Interaction: Second Edition*, Addison-Wesley Publ. Co., Reading, MA.

Shneiderman, B., Chimera, R., Jog, N., Stimart, R. and White, D. (1995), "Evaluating spatial and textual style of displays", *Proc. of Getting the Best from State-of the-Art Display Systems '95*, London.

Streveler, D. and Wasserman, A. (1987), "Quantitative measures of the spatial properties of screen designs", *Proc. of INTERACT '87,* Elsevier Science, Amsterdam, 125-133.

Tullis, T. S. (1988a), "Screen design", In Helander, M. (Editor), *Handbook of Human-Computer Interaction*, Elsevier Science, Amsterdam, The Netherlands, 377-411.

Tullis, T. S. (1988b), "A system for evaluating screen formats: Research and application", In Hartson, H. Rex and Hix, Hartson, *Advances in Human-Computer Interaction: Volume 2,* Ablex Publishing Corp., Norwood, NJ, 214-286.

Vanderdonckt, J. and Gillo, X. (1994), “Visual techniques for traditional and multimedia layouts”, In Catarci, T., Costabile, M., Levialdi, S. and Santucci, G. (Editors), *Proc. Advanced Visual Interfaces Conference ‘94,* ACM Press, New York, 95-104.