DEFINITION AND EVALUATION OF A SYNTHESIS-ORIENTED, USER-CENTERED TASK ANALYSIS TECHNIQUE: THE TASK MAPPING MODEL

by

Kevin Andrew Mayo

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Computer Science

APPROVED:

H. Rex Hartson, Chairman

Helen J. Crawford

Sallie M. Henry

Deborah Hix

Robert C. Williges

January, 1994
Blacksburg, Virginia
DEFINITION AND EVALUATION OF A SYNTHESIS-ORIENTED, USER-CENTERED TASK ANALYSIS TECHNIQUE: THE TASK MAPPING MODEL

by

Kevin Andrew Mayo

ABSTRACT

Analytic evaluation is a term describing a class of techniques for examining a representation of a user interface design and discovering design flaws and/or predicting user task performance. In our work with analytic evaluation, we have observed limitations on the effectiveness and efficiency of analytic techniques for formative evaluation supporting the iterative design and re-design cycle. In our search for an alternative, we have developed the Task Mapping Model, a substantively different approach to analysis for supporting the user interface design.

This body of work defines the task mapping model. This definition includes a task description framework and analysis methodology for using the task mapping model in the iterative user interface development process.

An evaluation experiment was performed to illustrate the usefulness of the task mapping model in comparison to using design guidelines. The evaluation experiment compared both methods ability and utility for synthesizing new user interface design requirements.

As is often the case with first-time users of complex but effective techniques, quality measures of the resulting analyses were more positive than perceived complexity. The study showed that the task mapping model was more difficult to use and learn than design guidelines; however, the task mapping model was shown to produce higher quality UIDs than DGL. This study serves as a stepping stone for future studies involving the task mapping model and real world user interface developers.
This work is dedicated to my parents

George and Rebecca Mayo

Their love has made me who I am today
ACKNOWLEDGMENTS

I would like to thank first and foremost the contributions of my family, specifically my parents George and Rebecca Mayo. Their financial, emotional, and spiritual support has carried me through this, and my accomplishment is their accomplishment.

Next, I would like to thank my committee, and most prominently my committee chair. Dr. Rex Hartson’s efforts and pains on my behalf cannot be understated. It’s nice to find someone who will be a friend for life and always greets you with a smile.

Dr. Deborah Hix, even in her flurry of activity, always took time to read my materials and add a more than helpful hand. Dr. Robert Williges, who initially intimidated me, has turned out to be one of the nicest individuals I have had the pleasure to know, with perhaps the exception of his wife Beverly Williges. I remember late evenings I delayed them with question after question—and they answered them all. Dr. Sallie Henry has been there at every turn to help and encourage me, and for that I am grateful. Finally, I could not have asked for a better psychology mentor than Dr. Helen Crawford. Her enthusiasm for my work and HCI was very much appreciated.

I would like to thank Bill Wake for keeping me on target. Just given the power, Bill and I could correct many of the worlds ills—or at least we like to think we could. Other members of my sanity insurance team include: Ed & Sue Bond, Karen Bowen, Steve Davis, Virginia Hampton, Jeff Price, Richard & Mary Rognlie, Shawn & Lynn Thomson, and Steve & Deb Wake. Also, I don’t want to forget Yoda, Felecia, Alex, Boris, Cossette, Marius, and Sammy.

Finally, a special thank you with love to Allison Moser for being who she is.
# TABLE OF CONTENTS

Abstract ................................................................................................................................. ii  
Acknowledgments ................................................................................................................ iv  
Table of Contents ................................................................................................................ v  
List of Figures ....................................................................................................................... ix  
List of Tables ....................................................................................................................... xii 

I  Executive Summary ........................................................................................................... 1  
   I.1 Problem ...................................................................................................................... 1  
   I.2 Solution ..................................................................................................................... 3  
   I.3 Approach to Solution ............................................................................................... 5  
   I.4 Summary of TMM ..................................................................................................... 5  
   I.5 Evaluation of Solution ............................................................................................ 7  
   I.6 Evaluation Results Summary .................................................................................. 8  
   I.7 Dissertation Overview ............................................................................................ 9  

II  Related Work ................................................................................................................... 12  
   II.1 User Interface Development Process ........................................................................ 12  
   II.2 Task Analysis in HCI .............................................................................................. 15  
      II.2.1 Task Analysis ..................................................................................................... 16  
      II.2.2 Views of User Tasks ......................................................................................... 17  
      II.2.3 Using Task Analysis in HCI ............................................................................. 19  
      II.2.4 Task-Artifact Cycle ........................................................................................ 19  
   II.3 HCI Models ............................................................................................................ 20  
      II.3.1 Prediction Based Models .................................................................................. 21  
         GOMS Family Models ............................................................................................ 21  
         GTN & Cognitive Complexity Theory ..................................................................... 23
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>II.3.2</td>
<td>Knowledge Analysis Models</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td>Action Language</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>TAKD &amp; TKS</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>TAG</td>
<td>28</td>
</tr>
<tr>
<td>II.3.3</td>
<td>Interface Models</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>CLG</td>
<td>30</td>
</tr>
<tr>
<td></td>
<td>ETAG</td>
<td>32</td>
</tr>
<tr>
<td>II.3.4</td>
<td>Usability Inspection Models</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Cognitive Walkthrough</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Heuristic Evaluation and Design Guidelines</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>Claims Analysis</td>
<td>36</td>
</tr>
<tr>
<td>II.3.5</td>
<td>User Models</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Norman’s Theory of Action</td>
<td>38</td>
</tr>
<tr>
<td></td>
<td>Long Term Knowledge Model</td>
<td>39</td>
</tr>
<tr>
<td></td>
<td>UAN</td>
<td>40</td>
</tr>
<tr>
<td>II.3.6</td>
<td>Criticisms of HCI Models</td>
<td>41</td>
</tr>
<tr>
<td>II.4</td>
<td>Comparisons among HCI Models</td>
<td>42</td>
</tr>
<tr>
<td></td>
<td>Modeling Characteristics</td>
<td>43</td>
</tr>
<tr>
<td></td>
<td>Comparison Matrix</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>Comparison between TMM and other HCI Modeling Techniques</td>
<td>48</td>
</tr>
<tr>
<td></td>
<td>Comparisons Based on Design Support Characteristics</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Comparisons Based on User Class Characteristics</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>Comparisons Based on Modeling Type Characteristics</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>TMM’s Rally Cry</td>
<td>51</td>
</tr>
<tr>
<td>II.5</td>
<td>Conclusions</td>
<td>52</td>
</tr>
<tr>
<td>III</td>
<td>The Task Mapping Model</td>
<td>53</td>
</tr>
<tr>
<td>III.1</td>
<td>TMM Life Cycle</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td>The TMM as a Initial Design Technique</td>
<td>56</td>
</tr>
<tr>
<td></td>
<td>The TMM as a Walkthrough Technique</td>
<td>56</td>
</tr>
<tr>
<td>III.2</td>
<td>TMM Task Description Framework</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>III.2.1 Domains</td>
<td>57</td>
</tr>
<tr>
<td></td>
<td>III.2.2 Domain Items</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>III.2.3 Domain Languages</td>
<td>62</td>
</tr>
</tbody>
</table>
Domain Vocabulary ................................................................. 63
Domain Vocabulary: UAN ......................................................... 67
Meta- and Micro-Language Relationships ............................... 67
Domain Grammars ................................................................. 67

III.2.4 Translations and Mappings ........................................... 68
III.2.5 Knowledge Requirements ............................................ 69
User Knowledge Categories .................................................. 69
Manipulating Knowledge Representations ............................. 72

III.3 Other Task Description Issues in the TMM ....................... 74
III.3.1 Domain Skipping in Task Performance .......................... 74
Novice User Domain Skipping .............................................. 74
Expert User Domain Skipping ............................................. 76
III.3.2 Task Contractions ....................................................... 76
III.2.3 Super-Tasks .............................................................. 78

III.4 TMM Methodology ....................................................... 78
III.4.1 Task Description ....................................................... 79
III.4.2 Knowledge Analysis ................................................ 81
III.4.3 UIDR Synthesis ....................................................... 83

III.5 Conclusions .............................................................. 86

IV Comparison of TMM & DGL .................................................. 87

IV.1 Experiment Objectives .................................................... 88
IV.2 Experimental Design Rationale ...................................... 89
IV.2.1 Why not compare TMM with GOMS, CLG, or another HCI modeling technique? ............................... 89
IV.2.2 Analyst Team size .................................................... 90

IV.3 Pre-Test ................................................................. 90
IV.3.1 Participant training in analysis methods ....................... 90
IV.3.2 Interface design prototypes ....................................... 91
IV.4 Phase 1: Determine Usability Problems .......................... 91
IV.4.1 Method .............................................................. 91
IV.5 Phase 2: Synthesize New UIDRs ................................... 95
IV.5.1 Method .............................................................. 95
IV.5.2 Results and Discussion ........................................... 97
Analyst Survey Results .................................................... 97
Direct Comparison between TMM and DGL (ANOVA Results)........97
T-test on Likert Scales ........................................................................................................101
T-test Results on Individual TMM Specific Scales.........................................................101
T-test Results on Individual DGL Specific Scales .........................................................103

IV.6 Phase 3: Evaluate Synthesized UIDRs .................................................................108
IV.6.1 Methods................................................................................................................108
IV.6.2 Results and Discussion ......................................................................................111

IV.7 Conclusions............................................................................................................113

V Conclusions .................................................................................................................115

V.1 Evaluation Experiment Overview ............................................................................115
V.2 Contributions of Research .....................................................................................118
V.3 Limitations of this Research ....................................................................................118
V.4 Future TMM Research Directions ..........................................................................119

Bibliography ..................................................................................................................122
Appendix A: TMM Analysis Manual .............................................................................132
Appendix B: TMM Tutorial ............................................................................................192
Appendix C: UIDR Report Form ....................................................................................227
Appendix D: Analyst Surveys .........................................................................................230
Appendix E: UIDR Quality Report Form ........................................................................247
Appendix F: Phase two Data ..........................................................................................262
Appendix G: UIDRs .........................................................................................................302
Appendix H: Phase Three Data .......................................................................................319
Appendix I: TMM & DGL Training Materials .................................................................327
VITA .................................................................................................................................375
LIST OF FIGURES

FIGURE I.1. Iterative User Interface Development Process ........................................... 2
FIGURE I.2. Iterative User Interface Development Process with TMM Analysis .......... 4
FIGURE I.3. TMM Framework to Describe Tasks ....................................................... 6
FIGURE I.4. TMM Life Cycle ....................................................................................... 7
FIGURE I.5. The TMM Evaluation Study ...................................................................... 8

FIGURE II.1. Star Life-Cycle of the User Interface Development Process .............. 13
FIGURE II.2. Human-Computer Interface Software Design Flow Diagram ............ 14
FIGURE II.3. HCI User Interface Development Model ............................................. 15
FIGURE II.4. Task Analysis in Human-Computer Interfaces .................................. 18
FIGURE II.5. Task-Artifact Cycle .............................................................................. 20
FIGURE II.6. Excerpts from (Kiera & Polson, 1983) GTN example ....................... 24
FIGURE II.7. Excerpts from (Moran, 1983) ETIT Example .................................. 25
FIGURE II.8. Excerpts from (Reisner, 1984) Action Language Example ............. 26
FIGURE II.9. Excerpts from (Diaper, 1989) THD Example .................................. 27
FIGURE II.10. Excerpts from (Diaper, 1989) KRG Example .............................. 27
FIGURE II.11. Excerpts from (Payne, et al., 1989) TAG Description of MacDraw
                   Interface  ................................................................................................. 29
FIGURE II.12. Excerpts from (Taub, 1990) ETAG Example ............................... 33
FIGURE II.14. Model of User Long-Term Knowledge (Shneiderman, 1987) .......... 40
FIGURE II.15. UAN Example to Delete a File (Hartson, et al., 1990) ................. 41
FIGURE II.16. Comparison Matrix for HCI Modeling Techniques and TMM ........ 47
FIGURE III.1. The Iterative User Interface Development Process with TMM Analysis...........54
FIGURE III.2. TMM Life cycle within Iterative User Interface Development .........................55
FIGURE III.3. Alternative Interaction Domains to Investor Task: get stock quotes ...............58
FIGURE III.4. Using a Computer (On-Line System) to get stock quotes ..........................59
FIGURE III.5. How Norman's Theory of Action is Applicable ........................................60
FIGURE III.6. User Task Format A Series Of Disks With Domain Skipping .....................75
FIGURE III.7. User Task Format A Series Of Disks Without Domain Skipping ..................75
FIGURE III.8. Task: Delete a File v.1 ........................................................................77
FIGURE III.9. Task: Delete a File v.2 (with Task Contraction) .......................................77
FIGURE III.10. Task Description Process within TMM ..................................................79
FIGURE III.11. Task Description For Task: Exit Document Preparation System ..............80
FIGURE III.12. Knowledge Analysis within TMM .........................................................82
FIGURE III.13. UDR Synthesis within TMM ...................................................................84
FIGURE III.14. Example Translation of an Unsupported Knowledge Requirement Into
  a New User Interface Design Requirement (UIDR) ..................................................85
FIGURE IV.1. Three Phase Experiment to Evaluate the TMM .........................................88
FIGURE IV.2. Phase 1 of the TMM Evaluation Experiment ...........................................92
FIGURE IV.3. Phase 2 of the TMM Evaluation Experiment ...........................................95
FIGURE IV.4. Phase 3 of the TMM Evaluation Experiment ...........................................108

FIGURE A.F.1. Learnability comparisons (TMM and DGL) ..........................................267
FIGURE A.F.2. DATA FOR TMM Method ..................................................................268
FIGURE A.F.3. DATA FOR TMM Process ..................................................................269
FIGURE A.F.4. DATA FOR TMM Notation & Formalisms ............................................270
FIGURE A.F.5. DATA FOR TMM Learnability .............................................................271
FIGURE A.F.6. DATA FOR Using TMM Domains .......................................................272
FIGURE A.F.7. DATA FOR Finding TMM Domain Items .............................................273
FIGURE A.F.8. DATA FOR Identifying TMM Mappings ..............................................274
FIGURE A.F.9. DATA FOR Identifying TMM Mappings Knowledge Requirements .......275
FIGURE A.F.10. DATA FOR Classifying TMM Knowledge Requirements .....................276
FIGURE A.F.11. DATA FOR Determining if Knowledge is Supported or Unsupported
  (TMM) ..................................................................................................................277
FIGURE A.F.12. DATA FOR Synthesizing User Interface Design Requirements using
  TMM ......................................................................................................................278
LIST OF TABLES

TABLE I.1. Results of Comparing TMM against DGL .......................................................... 9
TABLE II.1. Excerpts of CLG Example (Task and Semantic Levels) ........................................ 31
TABLE II.2. Excerpts of CLG Example (Syntactic Levels) .................................................. 32
TABLE II.3. Comparison of Usability Inspection Methods (Jeffries, et al., 1991) ....................... 35
TABLE III.1. Taxonomies of user Actions (Lenorovitz, et al., 1984) ........................................ 64
TABLE III.4. Example Grammars For TMM Domains ......................................................... 68
TABLE III.5. Two Examples of Knowledge Representation within the TMM ......................... 73
TABLE III.6. Knowledge Requirements For Task: Exit Document Preparation System ................. 81
TABLE III.7. Knowledge Analysis for task: Exit document Preparation system ........................ 83
TABLE IV.1. Interface Design Prototype—IIMTS II 4.01 (Thomas, 1992) ............................. 93
TABLE IV.2. Interface Design Prototype—PERIODIC TABLE (Weber, 1992) ......................... 93
TABLE IV.4. Interface Design Prototype—Celtic museum (Newton, 1991) ........................... 94
TABLE IV.5. Summary of Significant Results Comparing TMM & DGL Analysts’ Reactions to Analysis Method ...................................................................................... 98
TABLE IV.6. Summary of Significant Results Comparing TMM & DGL Analysts’ Reactions to Analysis Method’s Process .............................................................. 99
TABLE IV.7. Summary of Significant Results Comparing TMM & DGL Analysts’ Reactions to Analysis Method’s Notation and Formalisms .............................................. 99
TABLE IV.8. Summary of Significant Results Comparing TMM & DGL Analysts’ Reactions To Analysis Method’s Learnability (For Both Process And Notation) ........................................................................... 99
TABLE IV.9.a. Summary of Significant t-tests on TMM-Specific Scales .................................................. 101
TABLE IV.9.b. Summary of Significant t-tests on TMM-Specific Scales .................................................. 102
TABLE IV.10.a. Summary of Significant t-tests on DGL-Specific Scales ............................................... 104
TABLE IV.10.b. Summary of Significant t-tests on DGL-Specific Scales ............................................... 105
TABLE IV.10.c. Summary of Significant t-tests on DGL-Specific Scales ............................................... 106
TABLE IV.11. Results of Comparing TMM against DGL ........................................................................... 112
TABLE A.F.1. Raw Data for TMM Analyst Surveys .............................................................................. 263
TABLE A.F.2. Raw Data for DGL Analyst Surveys .............................................................................. 264
TABLE A.F.3. Method Comparisons (TMM and DGL) ........................................................................ 265
TABLE A.F.4. Process Comparisons (TMM and DGL) ........................................................................ 265
TABLE A.F.5. Notations And Formalisms Comparisons (TMM and DGL) ........................................ 266
TABLE A.F.6. Comparisons of TMM and DGL using Parametric and Nonparametric Analyses ........... 300
TABLE A.F.7. 10 Randomly Selected Likert Scales comparing Parametric and Nonparametric Analyses .................................................................................................................. 301
TABLE A.H.1. Raw Data From Evaluator 1 ......................................................................................... 320
TABLE A.H.2. Raw Data From Evaluator 2 ......................................................................................... 320
TABLE A.H.3. Raw Data From Evaluator 3 ......................................................................................... 321
TABLE A.H.4. Raw Data From Evaluator 4 ......................................................................................... 321
I EXECUTIVE SUMMARY

This chapter gives an overview of the research presented in this dissertation. We (1) identify problems within current user interface design processes, (2) discuss a solution to the problems, (3) outline an approach to the solution of these problems, (4) outline an evaluation of the solution, and (5) provide a brief overview of evaluation results.

I.1 PROBLEM

In today's society interactive computer systems have become commonplace and the number and diversity of users has increased greatly. Interactive systems aid and solve many day-to-day tasks, but as their power increases—interaction barriers (i.e., usability problems) also increase. User interface development processes produce user interface designs and also usability problems. Human-Computer Interaction (HCI) researchers have made progress on improving usability by improving user interface development processes; however, some problems still remain.

While good user interface development does not guarantee highly usable interactive systems, we believe good development techniques are necessary for the possibility of achieving usable systems. Therefore, the control and support of the user interface development process is crucial to design quality and system usability.
User interface development is a cyclical, iterative process that has been modeled by various researchers (Carroll, Kellogg, & Rosson, 1991; Gould & Lewis, 1983; Gould, 1988; Hartson & Hix, 1989; Hix & Hartson, 1993; Norman & Draper, 1986; Shneiderman, 1987; Williges, Williges, & Elkerton, 1987). However, each of these models suffers from a common problem: there is a gap between the evaluation and redesign phases of user interface development. Specifically, there is a lack of methodological support for turning formative evaluation results into new user interface design solutions. Examining this gap in more depth, we observe a lack of support for translating formative evaluation data into any type of design component (e.g., design requirement, design specification, high-level design solution, low-level design solution). Using a simplified version of Williges’ (1987) Software Design Flow Diagram, Figure I.1 depicts phases of user interface development and the missing methodological support (identified by the ‘?’).

![Figure I.1. Iterative User Interface Development Process](image-url)

---

Chapter I: Executive Summary
The research presented here introduces, defines, and evaluates a new modeling technique that addresses the missing methodological support within the user interface development process. This task-/user-centered technique is the Task Mapping Model (TMM).

1.2 SOLUTION

The research presented here bridges the gap within user interface development by introducing a modeling technique that transforms systems analysis results and formative evaluation findings into new user interface design requirements (UIDRs)—see Figure 1.2.

Analysts use TMM techniques to describe and analyze tasks for synthesizing new UIDRs from systems analysis results and, if available, usability problems identified within formative evaluation. Synthesized UIDRs are based on analysis of user tasks and knowledge needs for task performance.
It is stressed that the TMM produces only new user interface design requirements—\emph{not design solutions} of any form. New interface design requirements focus attention on unsupported user knowledge needs while not inhibiting the creative processes of user interface designers.

It is meaningful to argue that this model, like other similar models, provides a framework for task descriptions and analyses. However, we also argue that this model pragmatically fits within the iterative cycle of design and evaluation. Therefore, this model represents a hybrid form of task analysis—one that examines tasks within a dynamic, evolving interface design, and not based on a static task hierarchy.
I.3 APPROACH TO SOLUTION

A fundamental premise of the TMM is that users perceive tasks within various levels of abstraction. To perform any task with the help of a computer, users map tasks back and forth between their problem domain (e.g., word processing, bibliographic referencing, accounting) and the computer domain (i.e., hardware and software of the computer).

We believe that many usability problems can be traced to a failure of interface designers to understand these mappings and users’ needs as they move through these mappings. Current task analysis techniques, while providing task descriptions, often overlook the modeling or description of task mapping activities. These techniques presuppose, but do not describe, the translations between domain items—they discuss the product without discussing the process. User needs during task performance cannot be supported as well in interface design without understanding task mappings.

Therefore, the approach to solve this problem is development of a model that:

- fits within the iterative user interface development cycle,
- identifies and describes details of user task mappings,
- presents mapping details within a framework that facilitates identification of user needs for task performance, and
- provides mechanisms to synthesize user interface design requirements from task descriptions.

The TMM addresses each of these bullets by providing various levels of user abstraction, mappings among abstraction levels, user knowledge requirements for performing mappings, as well as methodologies to perform analyses within this framework.

I.4 SUMMARY OF TMM

It is a basic premise within the TMM that the use of computers to perform tasks requires each task to be conceptualized across various domains of abstraction. Figure I.3 shows abstraction domains that comprise the TMM task description framework (i.e., Problem, Computer Semantic, Computer Syntactic, and Articulation Domains). Each domain contains domain items (actions,
objects, and sub-tasks represented in the figure as open circles) associated with its level of abstraction.

![Diagram of TMM Framework]

**FIGURE I.3. TMM FRAMEWORK TO DESCRIBE TASKS**

We make three observations about this framework as a model of task performance. Our first observation is that users must map each task being performed with a computer from one domain to the next, starting from the problem domain, which is known to users, and going to the computer articulation domain, which contains the necessary physical interactions. The second observation is that mappings also exist in the opposite direction supporting evaluation as well as execution paths. Finally, the third observation is that the articulation domain is the site of the User Action Notation (UAN), a task-oriented notation developed for behavioral representation of user interface designs (Siochi & Hartson, 1989; Hartson, Siochi, & Hix, 1990; Hartson & Gray, 1992).

For task modeling and description purposes, the TMM has three types of knowledge: **factual, conceptual, and procedural**. These types are used to categorize knowledge and to identify what users know and what users need to know in relation to task performance. In this way, TMM task descriptions are analyzed to identify user knowledge needs that are unsupported (i.e., knowledge users are unaware of and for which the interface does not provide direct support).

Using the unsupported knowledge requirements identified by TMM task description and analysis, new user interface design requirements (UIDR) are synthesized. UIDRs represent one of the end-products of TMM analyses, and these are used directly in user interface re-design processes.
This process of task/user/artifact/knowledge analysis and UIDR synthesis is a three phase process—the TMM life cycle—see Figure I.4. The three phases of this process are: (1) task/interface description, (2) analysis of user knowledge requirements, and (3) synthesis of new user interface design requirements from unsupported user knowledge requirements.

FIGURE I.4. TMM LIFE CYCLE

I.5 EVALUATION OF SOLUTION

The TMM proves a framework for task descriptions and design requirement synthesis; however, this begs the question How usable is TMM? This is a very difficult question to address because of ambiguities associated with modeling usability and the scope of TMM analyses.

To evaluate the TMM, we compared the TMM against design guidelines (DGL) for synthesizing new UIDRs. The evaluation was a three phase experiment in which we (1) identified usability problems from several interfaces, (2) analyzed usability problems to synthesize new UIDRs with both TMM and DGL, and (3) evaluated the quality of the new UIDRs. Figure I.5 depicts the three phases of the TMM evaluation study.
Phases two and three provided data for evaluation of the TMM. In phase two, analysts provided feedback (both essay-style, and Likert scale-style) about each analysis method. While in phase three, evaluators rated the quality of the UIDRs generated in phase two.

### 1.6 EVALUATION RESULTS SUMMARY

Analysts, in phase two, rated the TMM more difficult to learn and use in comparison to DGL. This was not surprising—the TMM has more complicated processes and notations than DGL. Hence, it was expected that the TMM would be rated more difficult to learn and use. Future research can focus on making the TMM more learnable and usable by introducing support for TMM-walkthrough analyses.

The UIDR comparison results, from phase three, identifies several criteria where UIDRs generated using the TMM are significantly different (i.e., more favorable) than UIDRs generated
using DGL. Table I.1 summarizes the results of the criteria comparisons where positive means indicate TMM favorability and negative means indicate DGL favorability. Table I.1 also provides p-values showing the significance level of the grand-mean for each criterion (and averages) against zero.

<table>
<thead>
<tr>
<th>UIDR Quality Analysis</th>
<th>Mean</th>
<th>Std Err</th>
<th>F-Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addresses Usability Problem (UP)</td>
<td>0.578125</td>
<td>0.2213826</td>
<td>8.51</td>
<td>0.0055 **</td>
</tr>
<tr>
<td>Cohesive (COH)</td>
<td>0.250000</td>
<td>0.2293019</td>
<td>2.13</td>
<td>0.1517</td>
</tr>
<tr>
<td>Specificity Level (SPEC)</td>
<td>1.062500</td>
<td>0.1875000</td>
<td>58.06</td>
<td>0.0001 ***</td>
</tr>
<tr>
<td>Average Responses (of UP, COH, SPEC)</td>
<td>0.630208</td>
<td>0.1665206</td>
<td>27.58</td>
<td>0.0001 ***</td>
</tr>
<tr>
<td>User Centered (UC)</td>
<td>0.593750</td>
<td>0.1791719</td>
<td>14.54</td>
<td>0.0004 ***</td>
</tr>
<tr>
<td>Addresses User Needs (UN)</td>
<td>0.640625</td>
<td>0.1907681</td>
<td>17.36</td>
<td>0.0001 ***</td>
</tr>
<tr>
<td>Addresses User Task Structure (UT)</td>
<td>0.531250</td>
<td>0.1876240</td>
<td>12.26</td>
<td>0.0011 **</td>
</tr>
<tr>
<td>Average Responses (of UC, UN, UT)</td>
<td>0.588541</td>
<td>0.1488015</td>
<td>25.76</td>
<td>0.0001 ***</td>
</tr>
</tbody>
</table>

Avoids Introducing Usability Problems (AUP) | 0.109837 | 0.1838159 | 0.39 | 0.5334 |

* = Significant to $\alpha \leq 0.05$  ** = Significant to $\alpha \leq 0.01$  *** = Significant to $\alpha \leq 0.001$

Responses from all criteria of UIDR quality (except COH and AUP) were significantly different from zero—favoring TMM UIDRs. Hence, TMM synthesized UIDRs were judged more favorable over DGL synthesized UIDRs. This supports the claim that TMM produces higher quality UIDRs than DGL based on the criteria above.

In summary, while TMM is more difficult to learn and use, it produces higher quality UIDRs than DGL within the same time frame.

### 1.7 DISSERTATION OVERVIEW

We have identified the problem of missing methodological support within user interface development. This missing support motivates the necessity for a technique to translate both systems analysis and formative evaluation results into new user interface design requirements (UIDRs). In our approach to solving this problem, we developed the Task Mapping Model (TMM).
In order to provide a historical context, Chapter II identifies and discusses several related HCI research models and techniques. Chapter II identifies and compares differences between the TMM and other models while arguing the benefits of the TMM.

Chapter III gives an in-depth discussion on theoretical rationale behind TMM development—*why we did what we did*. Issues discussed within this chapter include: task and knowledge description using TMM, domain languages, and TMM methodologies.

To illustrate the impact of the TMM on user interface development, Chapter IV describes an empirical evaluation and comparison of TMM. The evaluation is based on comparing usability of TMM against usability of DGL for synthesis of UIDRs. The results are also discussed within Chapter IV.

Finally, Chapter V concludes the dissertation by revisiting analysts’ and evaluators’ reactions focusing on TMM strengths and weaknesses to provide avenues for future research.

This document also has several supporting appendices:

- **Appendix A: TASK MAPPING MODEL ANALYST GUIDE—A REFERENCE**
  This guide is defines and describes components of the TMM. This guide was given to TMM analysts in phase two. This is a copy of Mayo (1993a).

- **Appendix B: SITUATIONAL ANALYSIS WITH THE TASK MAPPING MODEL (TMM)—A TUTORIAL**
  This guide is used by TMM analysts in phase two. This is a tutorial approach defining TMM. This is a copy of Mayo (1993c).

- **Appendix C: USER INTERFACE DESIGN REQUIREMENT REPORT FORM**
  This form is used in phase two by all interface analyst groups to record new UIDRs per usability problem per interface. Analysts are also asked to characterize (on a scale) the impact of the usability problem to users and redesign.

- **Appendix D: USER INTERFACE ANALYST SURVEY**
  At the end of evaluation phase two, all analysts are given this survey. This survey allows analyst to assess their analysis method in terms of notation, process, and learnability.

- **Appendix E: PHASE 2 DATA**
  This appendix reproduces all data generated from analysts in phase two.
• Appendix F: SYNTHESIZED UIDRs
  All UIDRs generated in phase 2 are reproduced in this appendix. This appendix also includes analyses on data.

• Appendix G: UIDR QUALITY FORM
  This form is used in phase three of the evaluation by evaluators to characterize quality of the UIDRs.

• Appendix H: PHASE 3 DATA
  This appendix reproduces all data generated from the evaluators in phase three. This appendix also includes a copy of the analyses performed on the data.

• Appendix I: TMM & DGL TRAINING MATERIALS
  This appendix reproduces all overhead transparencies used in phase two training sessions.
II RELATED WORK

This chapter discusses generalized HCI modeling of the user interface development process, and in particular the activity of task analysis. Discussion includes examples of specific models used by practitioners. Following the examples is a comparison of these models against one another and the task mapping model (TMM).

II.1 USER INTERFACE DEVELOPMENT PROCESS

Researchers must understand a process in order to control or enhance it. Researchers often use modeling to abstract and conceptualize processes. Modeling refers to abstract formalization of processes with regards to specification of inputs, outputs, deliverables, and a representation of underlying activities. Abstraction is used as a conceptual tool to assist modelers.

With complex interactive computer systems flourishing and the diversity within user populations increasing, system usability is becoming a greater issue. This challenges HCI researchers and experts to produce more usable user interface designs. Therefore, researchers and experts have attempted to formalize (to various degrees) their approach to developing user interfaces. These attempts have led to several models of the underlying phenomena of the user interface development process.
Gould and Lewis outlined four factors critical to user interface development: early focus on users, interactive design, empirical measurement, and iterative design (Gould & Lewis, 1983; Gould, 1988). The first factor, early focus on users, emphasizes the importance of users in the interface development process—as the end-customers, users must be satisfied. Interactive design also shows the influence of empowering users by incorporating them directly into the design process. Empirical measurement and iterative design help define process by which user interface designs evolve.

A more detailed model of user interface development processes is the star life-cycle—see Figure II.1 (Hartson & Hix, 1989). Each node in the life-cycle represents a development phase, and all the nodes are interconnected through an evaluation phase. This model easily provides a framework for an iterative approach to user interface design based on user feedback and empiricism.

![Figure II.1. Star Life-Cycle of the User Interface Development Process (Hartson, et al., 1989)](image)

Another model of the user interface development process is presented by Williges, et al., (1987)—see Figure II.2. The development process is broken down into three iterative stages. The stages are initial design, formative evaluation, and summative evaluation. The initial design phase focuses on the specification of the initial user interface design for prototyping or implementation. The initial design is derived by iterating within several stages, including: examination of design objectives, performing a task/functional analysis, focusing on users for design inputs, use of guidelines, and design walkthroughs.
The formative evaluation stage of this model represents iterative evolution of the user interface design. Design evolution is driven by usability problems, user feedback, and analysis of the design. In other words, the evolution is based on an evaluation/redesign cycle, where the design is evaluated, often using a prototype, and new interface design requirements are incorporated into a revised design. This process continues until the design objectives are reached.

The final phase of user interface design process is summative evaluation. At this point designs are compared to alternative/previous designs and usability specifications. This phase can take place either in a lab or out in the field. In either case, this phase involves an evaluation of the functionality and acceptability of the final interface design.

The model we use to facilitate discussion is a simplified version of Williges, et al. (1987)—see depiction in Figure II.3. This model is based on a design/implement/evaluate iterative cycle. We feel that this model is representative of activities in which practitioners engage—and encompasses the processes within the previous models. The user interface design process begins with a functional and task analysis that generates an initial set of user interface design requirements. These requirements serve as the basis from which user interface designers formulate design specifications. A prototype, or perhaps a full featured implementation, is produced from the design specifications. The interface evaluation yields usability problems which must be rectified within a new design version.
A leap of faith exists between design evaluation and generation of new interface design requirements, i.e., there is no methodological support for generating new interface design requirements using evaluation findings. The research reported here focuses on a model that addresses this leap of faith by synthesizing interface design requirements using results from formative evaluation—the task mapping model (TMM).

II.2 TASK ANALYSIS IN HCI

Our research is interested in current HCI modeling techniques. Specifically, we are interested in models of the user interface development process, including design and task analysis models.
This section will introduce modeling within HCI research followed by a discussion of commonly accepted HCI models.

II.2.1 TASK ANALYSIS

“A task is an arrangement of behaviors (perceptual, cognitive, motor) related to each other in time and organized to satisfy both an immediate and a longer-range purpose.”

(Meister, 1985)

“Task analysis is a process of identifying and describing units of work, and analyzing the resources necessary for successful work performance. Resources in this context are both those brought by the worker (e.g., skills, knowledge, physical capabilities) and those which may be provided in the work environment (e.g., controls, displays, tools, procedures/aids).”

(Drury, Paramore, van Cott, Grey, & Corlett, 1987)

Task analysis in human-computer interaction is the study of tasks with emphasis on performance, correctness, or overall ease of the task. The examination of tasks can take place in two different frames of reference. The first frame concerns job tasks that already exist. Analyses of this type include examinations of the task with relation to:

- tasks and sub-task breakdowns
- instrumentation and display
- interplay between man and machine
- critical incidents

This type of analysis seeks to examine user tasks to enhance performance. Enhancement may include, among others: re-structuring the job, changing user interface instrumentation, or changing interface inputs or outputs. Examining the task and the system to improve user performance is widely accepted, but this is not the only method available to analysts. An analyst may also examine the user to improve task performance. Types of analysis include: examination of users’ models of the system, examination of necessary user mental needs (i.e., knowledge), and examination of necessary user physical needs. HCI experts have developed several models that examine both system and user for performance gains. (These models are discussed in future sections.)
The second frame concerns jobs that do not yet exist. Methods for this analysis includes (among others):

- examination of overall functionality
- sequencing among events (Timeline Analysis)

By systematically examining user's tasks, before design or implementation, the designer can reduce re-design time. However, often a focus on functionality overshadows the task- or user-centered approaches necessary for user interface development.

Task analysis has a rich background in human-factors and psychology (refer to Fath & Bias, 1992; Kirwan & Ainsworth, 1992). However this research is concerned with the contributions that task analysis methods give to HCI. It is therefore important to gain a perspective of task analysis' impact within this field.

II.2.2 VIEWS OF USER TASKS

Figure II.4 graphically depicts a model which examines task analysis in human-computer interactions (Carey, Stammers, & Astley, 1989). This model separates users' and designers' knowledge structures. Designers approach tasks constructively with the following in mind:

- the users' task
- operandi and operators of the system (i.e., data structures, procedures, etc.)—this includes both internal manipulations as well as external system changes by the user (via the interface)
- abstracted view of the system.

These form the foundations of the designers' knowledge structures by which usable and suitable interfaces are created.

---

1 It is quite possible that the system designer is not the interface designer, therefore, in this discussion 'designer' refers to the user interface designer.
On the other hand, the user does not have the benefit of knowing the underlying structure(s) of the system, and must therefore work with and through the interface to accomplish tasks. The user has a behavioral view of the system and tasks that includes:

- A user's understanding of the task. This understanding is seldom the same understanding attained by the designers.
- A mapping ability between the task understanding and the use of the computer.
- And lastly, a basic understanding of how to use the computer (device).

Simply put, the user must understand a task and the methods involved in executing it on a particular device. However, this is not always the case. In many instances users are unfamiliar with the device as well as with the task.

Carey's model does not show the impact of user diversity on task analysis. As the variety of users grow, the impact of their fluctuating abilities needs to be addressed. For example, a speech controlled interface may need modifications for a class of speech impaired users. Physical limitations, like this example, are not the only differences among users. There can also be differences in the underlying task/device knowledge amongst users.
II.2.3 Using Task Analysis in HCI

Task analysis is very useful to HCI specialists by focusing their attention and resources towards a task-/user-centered approach to interaction. But when should task analysis be used in user interface design? The first stages of the initial design is the first location where task analysis will benefit user interface designers. At this point, task analysis can identify necessary user tasks as well as complex user tasks.

During formative evaluation a prototype can be analyzed with task analysis methods. This provides feedback to designers concerning user requirement clarification, usability issues (e.g., empowered versus hindered users, confusing user interface features) and performance issues (e.g., predictions, task timings, etc.). Designers can also verify their model of the users’ tasks at this level.

Summative evaluation can employ task analysis methods for predictions/comparisons among alternative user interface designs. However, this use of task analysis is seldom the most efficient, and hence, very often it is not cost effective.

Task analysis, as mentioned above, can also be used to examine the users. Task analysis methods can be used throughout the development phase to examine users in terms of performance, cognitive needs and limitations, and physical needs and limitations.

II.2.4 Task-Artifact Cycle

This discussion of task analysis would not be complete without introducing Carroll’s work on the task-artifact cycle. As a reaction to the current human-computer interaction research in design, (Carroll, 1990) argues for a new approach to design. Stating that current task analysis methods are not used in industry because of underlying psychological flaws and non-applicability, Carroll poses the task-artifact cycle.

In this cycle, or model, the task influences the design which creates artifacts (hardware, software, and interface). These artifacts then influence the redesign in a cyclical fashion. The approach is couched in that argument that the artifacts we build are given meaning by users. Or, as Carroll, et al., (1991) puts it: “...designed artifacts can be interpreted as theories, as embodying myriad specific claims about their users and the circumstances of their use.” Scenarios are analyzed to
extrapolate a set of claims that represent the underlying psychological design rationale. (See Figure II.5.)

![Diagram](image)

**FIGURE II.5. TASK-ARTIFACT CYCLE**

Carroll (1990) outlines an example: "...typewriting altered office tasks, word processors altered them again, desktop publishing systems altered them still more. In each case, changed tasks themselves suggested new needs and opportunities for further change."

Successful artifact design, Carroll argues, is contingent on two factors: *infinite detail* and *emulation*. Infinite detail of the artifact is necessary because even the smallest design detail may have serious implications to usability. Also, the design representation should be in real world terminology emulating the users’ context. This cycle is a reaction to models derived within the ‘80s, also referred to the “first cycle of HCI”—which are discussed in the following sections.

### II.3 HCI MODELS

This section introduces current models that are employed during user interface development, at one stage or another. The models are separated into several categories. These categories represent the *general flavor* of the model, and while a model may contain several characteristics common to different categories, a ‘best fit’ was approximated. The categories are:

- **Prediction Based Models** are concerned with predicting various quantifiable measurements of users, interfaces, and tasks. The most common measurement that these models predict is task completion time. These models are often based on production rule systems.
• **Knowledge Analysis Models** attempt to gauge the necessary user knowledge for task completion. These models vary in their results, but are often based on formal grammars.

• **Interface Models** describe the system as the user understands it. Attempts are made to characterize different abstraction levels of user tasks. These models are often grammar based.

• **Usability Inspection Models** analyze interfaces in a design review or walkthrough format to locate usability problems.

These categories group HCI task analysis models. Each category of modeling is discussed and the appropriate models are included. A final section critiques the various categories.

### II.3.1 Prediction Based Models

As mentioned above, prediction models focus on predicting quantifiable measurements of users, interfaces, and tasks. Users are measured for physical and cognitive actions (e.g., time to move pointing device, number of items that can be recalled from memory). Interfaces are measured based on features such as inputs, outputs, buttons, windows, and menus to name a few. Measurements over tasks include task stacking (number of levels within sub-tasks), task completion times, etc. These three groups of measurement are all very dependent on each another; therefore, any attempt at prediction modeling must clearly distinguish measurements from confounding factors.

This research examines two models: GOMS (Card, Moran, & Newell, 1983) and Cognitive Complexity Theory (Kieras & Polson, 1985; Polson & Lewis, 1990). These models, described in the next sections, characterize this type of model. Both are built upon a similar production rule system that forms the basis of the GOMS model (e.g., IF <this> THEN <that>).

**GOMS Family Models**

Card, et. al., (1983) introduces the GOMS family of models. GOMS models have an underpinning of psychology theories. Their model of the human processor relies on nine principles of operation grouped within cognitive system (memory and processing) and human performance (perception, motor skill, and reaction time) categories. In conjunction with producing GOMS task descriptions, these theories are used to predict task completion times.
In this model, GOMS stands for Goals, Operators, Methods, and Selection rules. As (Card, et al., 1983) defines them:

- "A goal is a symbolic structure that defines a state of affairs to be achieved and determines a set of possible methods by which it may be accomplished."
  e.g., Goal: Edit-Manuscript

- "Operators are elementary perceptual, motor, or cognitive acts, whose execution is necessary to change any aspect of the user's mental state or to affect the task environment."
  e.g., Get-Next-Page, Verify-Edit

- "A method describes a procedure for accomplishing a goal."
  e.g., Goal: Acquire-Unit-Task
    Get-Next-Page
    Get-Next-Task

- "Method selection is handled by a set of selection rules."
  e.g., Goal: Locate-Line

  \[ \text{if the number of lines to the next modification is less than 3, then use the Lf-Method} \]
  \[ \text{else use the Qs-Method} \]

GOMS task descriptions are goal-directed tree structures that represent users' knowledge and task structure. GOMS represents a family of models, each of which differ by its level or granularity of analysis. The levels of analysis are: unit-task level, functional level, argument level, and keystroke level. The keystroke level model has the finest grain of description, and is therefore best used to describe tasks for predicting performance times.

GOMS is limited to analyses of error-free expert performance. However, even these limitations can be overlooked depending on the user class and task. GOMS has been used successfully for minimal document preparation (Gong & Elterkon, 1990), and has been extended into the highly complex user interaction domain of video games (John & Vera, 1992). GOMS has also been used in a telephone system analysis (John, 1990; Gray, John, & Atwood, 1992). An in-depth look at

---

2 The examples are from a discussion of the POET text-editing system (Card, et al., 1983). The user's task is to examine a hand-edited document and perform the changes within the POET system.
GOMS, including advantages, disadvantages, and current extensions is provided in (Olson & Olson, 1990).

**GTN & Cognitive Complexity Theory**

Kieras, et al., (1983) uses General Transition Networks (GTN) as a formalism to describe user interfaces. This notation represents system states and their associated translations without regard to actual implementation. GTNs support the testing of designs before prototypes need be generated; however, GTNs do not provide for objects being manipulated or for selection between alternative methods.

Figure II.6 depicts selections from an IBM Displaywriter word processor GTN. Nodes within this GTN represent states while arcs represent conditionally possible transitions. POP is a terminal state corresponding to successful completion. This representation style is more visually appealing than grammars.

Due to GTNs limited ability to only represent states and changes within a system, the cognitive complexity theory/model (CCT) was introduced—it is based on both GOMS-like representations and GTNs (Kieras, et al., 1985; Polson, et al., 1990). The production rules (GOMS-like) description holds the users how-to-do-it knowledge, while the GTN models the device/interface (i.e., how-it-works knowledge). The relationship between the production rules and the GTN form the task-device mappings. This model attempts to capture technology transfer, complexity of devices, learning and performance time, and error frequencies.
Using a simulation of the user over the production rules and GTNs, predictions of performance and learning times can be generated. Boviar, et al., (1990) provide an example analysis over editing tasks using this method. While results are encouraging, this model is very difficult for non-specialist use.

**ETIT**

Moran introduced the External-Internal Task Mapping analysis (Moran, 1983). This analysis examines the relationship between the external and internal task domains. Consider the example in Figure II.7. The external task domain represents the users' real-world tasks, while the internal domain represents the task in a computer setting. This method, while examining the products of
the two domains, does not attempt to isolate the underlying knowledge required to know the mapping rules.

<table>
<thead>
<tr>
<th>EXTERNAL TASK SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMS: CHARACTER, WORD, SENTENCE, LINE, PARAGRAPH (TEXT)</td>
</tr>
<tr>
<td>TASKS: ADD, REMOVE, TRANSPOSE, MOVE, COPY, SPLIT, JOIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INTERNAL TASK SPACE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TERMS: STRINGS</td>
</tr>
<tr>
<td>TASKS: INSERT, CUT, PASTE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MAPPING RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. SPLIT, JOIN, SENTENCE ⇒ CHANGE TEXT</td>
</tr>
<tr>
<td>2. TEXT ⇒ STRING</td>
</tr>
<tr>
<td>3. ADD ⇒ INSERT</td>
</tr>
<tr>
<td>4. REMOVE ⇒ CUT</td>
</tr>
<tr>
<td>5. TRANSPOSE ⇒ MOVE</td>
</tr>
<tr>
<td>6. SPLIT ⇒ INSERT</td>
</tr>
<tr>
<td>7. JOIN ⇒ CUT</td>
</tr>
<tr>
<td>8. CHANGE TEXT ⇒ CUT STRING + INSERT STRING</td>
</tr>
<tr>
<td>9. MOVE TEXT ⇒ CUT STRING + PASTE STRING</td>
</tr>
<tr>
<td>10. COPY TEXT ⇒ CUT STRING + PASTE STRING + PASTE STRING</td>
</tr>
</tbody>
</table>

FIGURE II.7. EXCERPTS FROM (MORAN, 1983) ETIT EXAMPLE

In this example the task space of deleting a section of text is examined. Both the external task space (problem domain) and internal task space (computer domains) items are listed. A set of mapping rules relates the external to internal task space, and thus outlines the mappings that users must make during task execution. However, this technique does not go beyond identifying the mappings and into identifying user needs for these mappings. Also, the use of this model by industry is questionable—there are no reported industrial experience papers or research.

II.3.2 Knowledge Analysis Models

Knowledge models attempt to analyze tasks and interfaces to discern the necessary user knowledge for task completion. Tasks may be analyzed to understand the user’s necessary knowledge for task completion with regards to task structure and control flow, task inputs/outputs, and possible task completion states among other characteristics.

Models of this variety are often based on grammars. The models that this research addresses, Action Language, TAKD/TKS, and TAG, are all based on grammars. Grammars can easily model the apparent flow of tasks within interfaces, however, they do have some drawbacks as we will see.
Action Language

Reisner’s Action Language is an BNF (Backus-Naur Form) representation of task-action mappings (Reisner, 1984). This phrase structured grammatical representation is a collection hierarchical rules representing the language users must use—legal action sequences. These rules, while identifying ‘interaction language’ user knowledge, can also predict performance time and errors; however, the analyst must supply time estimates which is a very problematic task in itself.

Figure II.8 shows an example of how Reisner’s formal grammar can be used in an interface development and modeling setting. The notation used within her examples follow from standard BNF notations: ‘|’ represents a selection, e.g., A | B is A or B; ‘⇒’ represents a substitution within the execution sequence; and brackets, ‘< >’, delineate cognitive versus physical actions. This example shows a ‘delete’ command within an editor application context.

<table>
<thead>
<tr>
<th>EMPLOY DN</th>
<th>⇒</th>
<th>&lt;RETRIEVE INFO. ON DN SYNTAX&gt; + USE D</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;RETRIEVE INFO. ON DN SYNTAX&gt;</td>
<td>⇒</td>
<td>&lt;RETRIEVE FROM HUMAN MEMORY&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;RETRIEVE FROM EXTERNAL SOURCE&gt;</td>
</tr>
<tr>
<td>&lt;RETRIEVE FROM HUMAN MEMORY&gt;</td>
<td>⇒</td>
<td>&lt;RETRIEVE FROM LONG TERM MEMORY&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;RETRIEVE FROM WORKING MEMORY&gt;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;USE MUSCLE MEMORY&gt;</td>
</tr>
<tr>
<td>RETRIEVE FROM EXTERNAL SOURCE</td>
<td>⇒</td>
<td>RETRIEVE FROM BOOK ⇒ ASK SOMEONE ⇒ EXPERIMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>USE ON-LINE HELP</td>
</tr>
<tr>
<td>USE DN</td>
<td>⇒</td>
<td>...</td>
</tr>
</tbody>
</table>

FIGURE II.8. EXCERPTS FROM (REISNER, 1984) ACTION LANGUAGE EXAMPLE

The action language representations of user tasks can be used to compare different system designs, as well as, form predictions based on the grammars and estimations of timings. (Reisner, 1984) produces an analysis of user performance with differences based in user knowledge (expert verses novice), and shows the formulation of estimates. She goes on further to show these estimates in both a physical action modeling and physical and cognitive action modeling setting.

TAKD & TKS

Task Analysis for Knowledge Descriptions (TAKD) focuses on deriving the knowledge necessary for successful task completion (Johnson, Diaper, & Long, 1984; Diaper, 1989). TAKD begins with a task description of objects and actions and synthesizes a Task Descriptive Hierarchy (TDH). The TDH maps objects from highly abstracted generic objects to specific objects. Figure II.9 gives an example over newsstand publication objects.
Mappings from specific to generic objects are represented in a Knowledge Representation Grammar (KRG) sentence. Specific KRGs are derived by examination of the THD, and then generic KRGs are synthesized. Figure II.10 gives the KRG statements (K1, K2, K3, and K4) derived from Figure II.9.

**K1. The Times**
NEWS-STAND PUBLICATIONS
/STATUS(QUALITY((GENERAL)))/
/FORMAT(NEWSPAPER(DAILY)(MONOCHROME PICTURES))//

**K2. THE OBSERVER**
NEWS-STAND PUBLICATIONS
/STATUS(QUALITY((GENERAL)))/
/FORMAT(NEWSPAPER(WEEKLY)(MONOCHROME PICTURES))//

**K3. THE SUN**
NEWS-STAND PUBLICATIONS
/STATUS(POPULAR((GENERAL)))/
/FORMAT(NEWSPAPER(DAILY)(MONOCHROME PICTURES)
[COLOUR PICTURES])//

**K4. THE KENSINGTON POST**
NEWS-STAND PUBLICATIONS
/STATUS(POPULAR((GENERAL)))/
/FORMAT(NEWSPAPER(WEEKLY)(MONOCHROME PICTURES))//

**FIGURE II.10. EXCERPTS FROM (DIAPER, 1989) KRG EXAMPLE**
Both ideal and non-ideal knowledge are identified and collected, but the final KRG represents only ideal knowledge that the user must possess. The user knowledge required is then the combination of all the knowledge required for each action/object pair.

Diaper (1990) supports that TAKD, "...is likely to lead to a better selection of the tasks that need to be observed" and "is likely to reduce analysis costs." However, TAKD does not differentiate knowledge for task mapping analysis, and TAKD does not model any task dynamics.

TAKD was later surpassed by task knowledge structures (TKS) (Johnson & Johnson, 1991). TKS is an attempt to capture user knowledge necessary for task completion, much like TAKD. The knowledge is broken down into several categories: goal structure, task plan, task strategies, procedures, task objects and task actions (Johnson & Johnson, 1991). They propose that knowledge required for successful task completion is identified within these categories. By identifying necessary knowledge, they argue that fewer errors will occur and task performance will be easier for users—i.e., identifying knowledge for design purposes will increase usability.


TAG

The Task Action Grammar (TAG) was designed to capture the mappings from tasks to actions within a formalized grammar (Payne & Green, 1986; Payne & Green, 1989). This representation supported consistency checking between task languages and system design specifications. Task languages are the interface commands and objects that are communicated between user and system. TAG grammars can be used to generate grammatically correct system input based on interaction style (e.g., command line, direct manipulation). TAG was originally not a performance model, but a competence model. Yet, complexity metrics can be gathered over a TAG to assess learnability.

As an example, Figure II.11 gives a TAG description of an idealized MacDraw interface. These rules support the general cases of object manipulation.³

³Objects in this context refer to rectangles, ellipses, and arcs.
SIMPLE TASKS

Create new object[Effect=create, Case=regular]
Create special object[Effect=create, Case=special]
Move object[Effect=move, Case=regular]
Move object along restricted path[Effect=move, Case=special]
Change default style attributes[Effect=change-default-style]
Change object style attributes[Effect=change-object-style]

RULE SCHEMAS

6.1 Task[Effect=create, Case] ⇒
   select tool
   + point-to-2-places[Case, Place1=value-from-goal, Place2=value-from-goal]

6.2 Task[Effect=move, Case] ⇒
   point-to-2-places[Case, Place1=value-from-goal, Place2=object-location]

6.3 Task[Effect=modify-object-style] ⇒ select-object + select-style

6.4 Task[Effect=change-default-style] ⇒ select-style

6.5 point-to-2-places[Case=regular, Place1, Place2] ⇒
   action[Type=point, Place1]
   + drag-to-place[Place2]

6.6 point-to-2-places[Case=special, Place1, Place2] ⇒
   action[Type=point, Place1]
   + "depress mouse button"
   + "depress SHIFT"
   + action[Type=point, Place2]
   + "release mouse button"
   + "release SHIFT"

6.7 drag-to-place[Place] ⇒
   "depress mouse button:"
   + action[Type=point, Place]
   + "release mouse button"

6.8 select-style ⇒ point-to-2-places[Place1=style-menu, Place2=menu-item]

6.9 select-object ⇒ action[Type=point, Place=object-location]
   + "click mouse button"

6.10 select-tool ⇒ action[Type=point, Place=tool-icon]
    + "click mouse button"

---

FIGURE II.11. EXCERPTS FROM (PAYNE, ET AL., 1989) TAG DESCRIPTION OF MACDRAW INTERFACE

Analyzing this description for organization consistency and conflict yields a new rule for 6.2—
6.2a:

6.2a Task[Effect=move] ⇒
   select-object
   + drag-to-place[Place=value-from-goal]

With this new rule, (Payne, et al., 1989) gives a consistent approach to the set of tasks by imposing
a selection on all tasks. (The selection can be a tool, object, or style.)
TAG "explicitly excludes any mechanism to describe how users choose their strategy to achieve their goals," (Wilson, Barnard, Green, & Maclean, 1988) This is a drawback when a designer considers different user populations and individual differences.

### 2.3.3 Interface Models

Interface models, as mentioned before, examines the interface as the users sees and abstracts it. The two models here, like knowledge analysis models, are based on grammars. This is not surprising, if it is possible to characterize user knowledge in grammars then the user abstraction of the system could also be characterized in grammars. The two models discussed here are CLG and ETAG.

**CLG**

The Command Language Grammar (CLG) models systems and users’ conceptions of systems (Moran, 1981). This modeling breaks down descriptions into three components: conceptual component (task level, semantic level), communicational component (syntactic level, interaction level), and physical component (spatial layout level, device level). These levels support modeling of users’ conception of the system. This leveling also serves to separate the command language for analysis.

Moran describes a mail system (EG) using CLG (Moran, 1981). Moran’s discussion of EG is quite lengthy, even though EG is a “toy” system, so only selected examples are shown here.

Table II.1 gives representative examples of the task and semantic levels. The task level contains entities and tasks defined within the application domain (e.g., MESSAGE-FILE, GET-INFORMATION). The semantic level objects (entities, operations, and methods) are also shown. These objects are task level concepts within an abstracted device domain (e.g., EG-SYSTEM, LOOK). Semantic methods relate semantic operations and entities to tasks (e.g., SEM-M1 = GET-INFORMATION (task-level task)). Table II.2 defines the syntactic level objects.

The analysis of a CLG description is three fold: linguistic, psychological, and design oriented. The linguistic analysis examines the command language structure (syntactic/interaction levels), meaning (semantic level), and purpose (task level). The psychological analysis perceives the CLG description as a representation of users’ knowledge. As a design tool, CLG provides the mechanism for creating a conceptual model of the system.
As can be seen, CLG is a powerful mechanism for describing systems, but CLG is also very cumbersome.

**TABLE II.1. EXCERPTS OF CLG EXAMPLE (TASK AND SEMANTIC LEVELS)**

<table>
<thead>
<tr>
<th>CLG Level</th>
<th>Object</th>
<th>Selected examples from (Moran, 1981) EG mail system</th>
</tr>
</thead>
</table>
| Task Level| Entities | SEND-MESSAGE = (AN ENTITY NAME = "Send message"
  
  (* This is a message sent by the SEND system
  
  A Send Message has a Header and a Body.
  
  The Header contains the fields To, From, Date, Time, and Subject.
  
  The Body contains arbitrary text.)
  
  MESSAGE-FILE = (A TEXT-FILE
  
  NAME = "Message File"
  
  OWNER = (A USER)
  
  (* There is only one MESSAGE FILE for each USER)
  
  (* Although the user may want to think of this file as having a
  
  sequence of SEND-MESSAGES, the operating system only treats it
  
  as a text file, i.e., as a sequence of characters.)
  
  )
  
  Task Level| Tasks | GET-INFORMATION = (A TASK
  
  (* Get a piece of information from the last SEND-MESSAGE
  
  from a given person and delete the SEND-MESSAGE from MESSAGE-FILE.)
  
  NEW-MAIL = THIN-OUT = (A TASK
  
  DO (SEQ: (NEW-MAIL) (THIN-OUT-MESSAGES))
  
  NEW-MAIL = (A TASK
  
  (* Check for new SEND-MESSAGES and, if any, read them.)
  
  (* This is the most frequent task.)
  
  DO (SEQ: (CHECK-FOR-NEW-MAIL) (READ-NEW-MAIL))
  
  Not shown: CHECK-FOR-NEW-MAIL, READ-NEW-MAIL, THIN-OUT-MESSAGES.
  
  )
  
  Semantic Level| Entities | EG-SYSTEM = (A SYSTEM
  
  NAME = "EG"
  
  ENTITIES = (SET: MESSAGE, SUMMARY
  
  MAILBOX DIRECTORY SCREEN)
  
  OPERATION = (SET SHOW DELETE)
  
  MESSAGE = (AN ENTITY
  
  REPRESENTS (A SEND-MESSAGE)
  
  NAME = "Message"
  
  AGE = (ONE-OF: OLD NEW)
  
  (* A MESSAGE has a Header and a Body. The Header
  
  contains the fields To, From, Date, Time, and Subject. The body
  
  contains arbitrary text.)
  
  (* The AGE is a time-dependent mark)
  
  )
  
  Not shown: MAILBOX, DIRECTORY, SCREEN, SUMMARY
  
  )
  
  Semantic Level| Operations | DELETE = (A SYSTEM OPERATION
  
  OBJECT = (A PARAMETER
  
  VALUE = (A MESSAGE)
  
  (* The OBJECT is removed from MAILBOX and its SUMMARY is removed
  
  from DIRECTORY.)
  
  LOOK = (A USER-OPERATION
  
  IN (A PARAMETER
  
  VALUE = (A PLACE ON THE SCREEN)
  
  DEFAULT VALUE = UNKNOWN)
  
  AT (A PARAMETER
  
  VALUE = (AN ENTITY))
  
  FOR (A PARAMETER
  
  VALUE = (PATTERN ENTITY))
  
  RESULT = (AN ENTITY (* that satisfies the FOR pattern)
  
  FAILURE = (* if nothing can be found that satisfies the FOR pattern.)
  
  (* The USER looks in some place AT some entity FOR something,
  
  which is another entity.)
  
  )
  
  Not shown: SHOW, READ.
  
  )
  
  Semantic Level| Methods | SEM-M1 = (A SEMANTIC-METHOD
  
  FOR GET-INFORMATION
  
  DO (SEQ: (START EG-SYSTEM
  
  (SHOW DIRECTORY)
  
  (LOOK AT DIRECTORY FOR (A MESSAGE))
  
  (SHOW (THE RESULT OF LOOK))
  
  (READ (THE RESULT OF LOOK))
  
  (DELETE (THE RESULT OF LOOK))
  
  (STOP EG-SYSTEM)))
  
  )
  
  Not shown: SEM-M2 (CHECK-FOR-NEW-MAIL), SEM-M3 (READ-NEW-MAIL), SEM-M4A & SEM-M4B (THIN-OUT-MAILBOX)
### Table II.2: Excerpts of CLG Example (Syntactic Levels)

<table>
<thead>
<tr>
<th>CLG Level</th>
<th>Object</th>
<th>Selected examples from (Moran, 1981) EG mail system</th>
</tr>
</thead>
</table>
| Syntactic Level | Entities | EG-CONTEXT = (A COMMAND-CONTEXT
| | | STATE VARIABLES = (SET: CURRENT-MESSAGES)
| | | DESCRIPTORS = (SET: MESSAGE-NO)
| | | DISPLAY-AREAS = (SET: DIRECTORY-AREA MESSAGE-AREA
| | | COMMAND-AREA)
| | | COMMANDS = (SET: SHOW-MESSAGE SHOW-NEXT-MESSAGE
| | | DELETE-CURRENT-MESSAGE QUIT-EG)
| | | ENTRY-COMMANDS = (SET: ENTER-EG
| | | ENTER-EDIT-NEW-MAIL)
| | | CURRENT-MESSAGE = (A STATE-VARIABLE
| | | CONTEXT = EG-CONTEXT
| | | VALUE = (A MESSAGE)
| | | NAME = "Current Message"
| | | Not shown: EG-DISPLAY-AREA, DIRECTORY-AREA, MESSAGE-AREA, COMMAND-AREA, MESSAGE-NO) |
| Syntactic Level | Commands to enter EG-CONTEXT | ENTER-EG = (A COMMAND
| | | CONTEXT = OS-CONTEXT
| | | NAME = "EG"
| | | DOES (IF ( THERE-IS (A MESSAGE IN MAILBOX))
| | | THEN (SEQ: (ENTER EG-CONTEXT)
| | | (SHOW DIRECTORY IN DIRECTORY-AREA)
| | | (SHOW MESSAGE THE RESULT OF THERE-IS)))
| | | ELSE (REPORT ("No Messages in Mailbox")))
| | | Not shown: ENTER-EG-IF-NEW-MAIL |
| Syntactic Level | Commands in EG-CONTEXT | EG-COMMAND = (A COMMAND
| | | CONTEXT = EG-CONTEXT)
| | | SHOW-MESSAGE = (AN EG-COMMAND
| | | NAME = "MESSAGE"
| | | OBJECT = (AN ARGUMENT
| | | FORM = (A MESSAGE-NO)
| | | DOES (SET: (SHOW (SUMMARY OF (THE OBJECT))
| | | IN DIRECTORY-AREA)
| | | (SHOW (MESSAGE-NO OF (THE OBJECT))
| | | IN DIRECTORY-AREA)
| | | (SHOW (THE OBJECT) IN MESSAGE-AREA))
| | | SIDE-EFFECT = (BIND CURRENT-MESSAGE TO (THE OBJECT))
| | | (* Displays the SUMMARY by highlighting the SUMMARY of
| | | OBJECT in the DIRECTORY in DIRECTORY-AREA)
| | | Not shown: (SHOW-NEXT-MESSAGE, QUIT-EG, DELETE-CURRENT-MESSAGE) |
| Syntactic Level | Methods | SYN-M1 = (A SYNTACTIC-METHOD
| | | FOR GET-INFORMATION
| | | DO (SEQ:
| | | (ENTER-EG
| | | (LOOK IN DIRECTORY-AREA AT DIRECTORY
| | | FOR (A MESSAGE-NO))
| | | (SHOW-MESSAGE THE RESULT OF LOOK)
| | | (READ (THE CURRENT-MESSAGE)
| | | IN MESSAGE-AREA)
| | | (DELETE-CURRENT-MESSAGE)
| | | Not shown: SYN-M2 (CHECK-FOR-NEW-MAIL), SYN-M3 (READ-NEW-MAIL), SYN-M4 & SYN-M8 (THIN-OUT-MAILBOX) |

---

**ETAG**

The Extended Task-Action Grammar (ETAG) model is another interface modeling technique (Tauben, 1990). This technique adopts TAG's task description abilities, but it also includes computer system task-related semantics within its notation. Using levels similar to CLG, ETAG
defines the user's virtual machine (UVM), a dictionary of 'basic tasks', and production rules describing the system command language.

*PART OF THE CANONICAL BASIS FOR A UVM*

[CONCEPT] ::= [OBJECT] | [VALUE] | [PLACE] | [STATE] | [EVENT]
[PLACE] ::= [place.IN([OBJECT])], [place.on([OBJECT])], ...
[STATE] ::= [state.IS-AT([OBJECT],[PLACE])], ...
[EVENT] ::= [event.KILL-ON([OBJECT],[PLACE])], ...

type [EVENT > event.MOVE-TO([OBJECT:* o],[PLACE:* p1])]
precondition: [state.IS-AT ([OBJECT:* o],[PLACE:* p0])]
clears: [state.IS-AT ([OBJECT:* o],[PLACE:* p0])]
postcondition: [state.IS-AT ([OBJECT:* o],[PLACE:* p1])]
end [EVENT]

A conceptual object and a conceptual event of a UVM

type [OBJECT > MESSAGE]
supertype: [TEXT];
themes: [HEADER], [BODY];
relations: [place.ON-POS(1) ([MESSAGE])], for [HEADER], ...
attributes: <SENDER>, <SENDING_DATE>, <RECEIVING_DATA>, <STATUS>, ...
end [MESSAGE]

type [EVENT > MARK_FOR_DELETION]
description: for ([MESSAGE:* x])
[STATE.IS-AT([MESSAGE:* x],<DELETION_MARK>); marked)];
precondition: [place.ON-POS(i) ([MESSAGE_FILE: * y]); * p1]);
comments: 'marks messages for deletion';
end [MARK_FOR_DELETION].

A basic task from the dictionary

[TASK > MARK_FOR_DELETION],
[EVENT > MARK_FOR_DELETION],
[MESSAGE_FILE:* y],
T9 [EVENT > MARK_FOR_DELETION][[OBJECT > MESSAGE [*x]]],
"mark messages of the current message file for deletion"

FIGURE II.12. EXCERPTS FROM (TAUBER, 1990) ETAG EXAMPLE

This example, Figure II.12, shows a conceptual specification for a UVM task (Mark_for_deletion).

Using production rules, this model provides mechanisms to further define the task at Specification, Reference, Lexical, and Keystroke abstraction levels. Tauber states that ETAG's main motivation "was to provide more 'descriptive power'" to aid the design process.

ETAG's strength is in describing commands and task structures for systems in a very system-centered approach. Like CLG, ETAG is very cumbersome, difficult, and more constructive than behavioral in nature.

II.3.4 Usability Inspection Models

As mentioned above, usability inspection models take a much less formal approach to analyzing user interfaces to locate usability problems. These methods are often form driven, and the underlying representation is often verbal protocol from designers and interaction experts, as
opposed to a formal grammar or production rule system. This research examines cognitive walkthroughs and heuristic evaluation as examples of the usability inspection models.

**Cognitive Walkthrough**

Cognitive walkthroughs is an attempt to “bring cognitive theory closer to practice; to enhance the design and evaluation of user interfaces in industrial settings” (Wharton, Bradford, Jeffries, & Franzkle, 1992). This technique provides methodologies for theory based usability inspection (Lewis & Polson, 1992; Lewis, Polson, Wharton, & Rieman, 1990). Cognitive walkthroughs center on users’ cognitive activities differentiating itself from other usability inspection models.

The cognitive walkthrough methodologies are based on a theory of exploratory learning, CE+, and are performed in a three phase process: preparation, evaluation, and results interpretation phase. Set in the iterative interface development process, cognitive walkthroughs are performed by examining and analyzing tasks using specific forms. These forms guide the analyst in ‘defect detection, amplification, and removal,’ a sample of these forms is found in Lewis, et al., (1992).

Strengths of this technique include: prediction of user actions, focusing and shorting empirical evaluation, and the method is usable by the designers. However, limitations of this model include: complex terminology and concepts couched in cognitive psychology, and absence of task selection criteria.

**Heuristic Evaluation and Design Guidelines**

Nielsen and Molich (1990) introduced heuristic evaluation as an alternative to costly evaluation of user interfaces. This method relies on several HCI experts examining a user interface focused on nine different usability heuristics:

- Simple and natural dialogue
- Speak the user’s language
- Minimize user memory load
- Be consistent
- Provide feedback
- Provide clearly marked exits
- Provide shortcuts
- Good error messages
- Prevent errors
Nielsen argues for these nine usability heuristics because they can be presented within a single lecture and are generally accepted within the HCI community. Focusing on nine heuristics as opposed to the long lists of guidelines (e.g., Smith & Mosier, 1986) also cuts evaluation costs. Heuristic evaluation, as a 'discount usability engineering' method, is cheap, easy to use, does not require advanced planning, and can be used early in the development process (Nielsen, et al., 1990).

Nielsen, et al., (1990) showed that the number of usability errors found increases dramatically if several people perform the evaluation (three to five specialists are suggested). Nielsen (1992) extended the findings to state that HCI specialists have a better chance of finding usability errors than non-specialists, and specialists with task domain knowledge perform even better than other specialists. Also, major usability problems have a greater chance of being found. Studies have compared heuristic evaluation with other methods showing that heuristic evaluation can be very useful in locating usability problems because it costs much less in terms of time and money (Jeffries, Miller, Wharton, & Uyeda, 1991; Karat, Campbell, & Fiegel, 1992; Nielsen, 1993). A summary of Jeffries, et al., (1991) findings is presented in Table II.3.

<table>
<thead>
<tr>
<th>Heuristic Evaluation</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Identifies many more problems</td>
<td>Requires UI expertise</td>
</tr>
<tr>
<td></td>
<td>Identifies more serious problems</td>
<td>Requires several evaluators</td>
</tr>
<tr>
<td></td>
<td>Low Cost</td>
<td></td>
</tr>
<tr>
<td>Usability Testing</td>
<td>Identifies serious and recurring problems</td>
<td>Requires UI expertise</td>
</tr>
<tr>
<td></td>
<td>Avoids low-priority problems</td>
<td>High cost</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Misses consistency problems</td>
</tr>
<tr>
<td>Guidelines</td>
<td>Identifies recurring and general problems</td>
<td>Misses some severe problems</td>
</tr>
<tr>
<td>Can be used by software developers</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cognitive Walk-through</td>
<td>Helps define users' goals and assumptions</td>
<td>Needs task definition methodology</td>
</tr>
<tr>
<td>Can be used by software developers</td>
<td></td>
<td>Tedious</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Misses general and recurring problems</td>
</tr>
</tbody>
</table>

Other research argues that heuristic evaluation cannot be used as is because it generates too many false positives (Bailey, Allan, & Raiello, 1992)—this is also reflected in (Jeffries, et al., 1991; Nielsen, 1992).

Heuristic evaluation is a useful tool in interface development; however, its lack of formalism, need for experts, and non-systematic approach may leave some usability problems undiscovered.
Also, this method, while locating usability problems, does not always suggest direct interface
design revisions.

The heuristics used in heuristic evaluation are a set of design guidelines. There has been much
research concerning the definition and validation of design guidelines (Apple®, 1987; Brown,
1989; Fox, 1992; Marshall, Nelson, & Gardiner, 1987; Mayhew, 1992; Microsoft®, 1992; OSF, 1991;
Smith, et al., 1986; U.S.L., 1992). The general approach for using design guidelines is locating
appropriate guidelines and tailoring them to a specific need. For example, the Smith and Moser
(1986) guideline for data display

2.1.18 TEMPORAL SEQUENCE: WHEN A SENTENCE DESCRIBES A SEQUENCE OF
EVENTS, PHRASE IT WITH A CORRESPONDING WORD ORDER,

is tailored to:

WHEN LISTING USER (BANK TELLER) ACTIONS ON ACCOUNTS, LIST THEM IN
THE TASK ORDER.

This new guideline is specific to the application/problem domain of banking systems.

Claims Analysis

Claims analysis focuses on deriving psychological assertions based on artifact scenario analysis
(Kellogg, 1990; Carroll, 1990). In other words, each artifact associated with an interface has
certain psychological claims. These claims have the following characteristics: “organized by
scenarios of use, associating aspects of design with important consequences for users, being
stated as tradeoffs, comprising and supporting a variety of levels of detail, and being in the
language of design” (Kellogg, 1990).

As a part of the task-artifact cycle analysis, the generation of claims is based on analyzing
scenarios concentrating on claims (Carroll, Rosson, & Singley, 1992). The questions can be
divided into user task stages reflecting Norman’s Theory of Activity (Norman, 1986).

* Goal setting

How does the artifact invoke goals in the user? How does the artifact suggest that a
particular task goal is appropriate or inappropriate? What inappropriate goals are
most likely?
• **Planning**
  What planning mistakes are most likely? How does the artifact encourage the use of background knowledge in planning a task?

• **Execution**
  How does the artifact make it easy or difficult to perform a task? Is one task make easy at the expense of another?

• **Noticing consequences**
  What are the most salient features of the artifact? What features are commonly missed?

• **Interpreting consequences**
  How does the artifact guide the user to make correct inferences? What incorrect inferences are most likely?

• **Goal evaluation**
  How does the artifact help users to recognize, diagnose and recover from errors?

From these questions a series of claims can be made about an artifact. These support an underlying psychological claim that designers have made. For example, a claim from the **Type and Print A Letter** scenario in (Carroll & Rosson, 1992) is: "*a structural description of major system components conveys a mental model of how the device works (which may help ground or rationalize the learner's understanding of how to use it). (but the structural description may distract the learner, may frustrate the learner's concern with typing and printing, and may simply fail to support use of the device)*" Notice how the claim identifies important design considerations for the end user—stated as a tradeoff. This claim is suggested by the claims question involving users' background knowledge and tasks. I.e., how does users' background knowledge help in task specification, and in this case, the systems components corresponds to a user mental model.

Claims analysis is an analytical method to examine tasks and artifacts in a scenario based approach. Current difficulties with this model includes the capturing and description of the psychological claims—currently an unstructured approach. Formalization of the process and resultant notation is also needed for this method.
II.3.5 USER MODELS

This section discusses several user models within human-computer interaction. These do not attempt to model the underlying cognitive processes (e.g., memory recall, logic deductions), but only to illustrate these processes existence within a modeling framework. This is not a complete introduction, but will present the reader to several notable user models to motivate this research.

*Norman's Theory of Action*

Norman (1986) introduces a theory of action. In describing a task, Norman divides user actions into seven states:

- Forming the Goal
- Forming the Intention
- Specifying an Action
- Executing the Action
- Perceiving the System State
- Interpreting the System State
- Evaluating the Outcome

Figure II.13 graphically displays Norman's seven stage theory of action. It should be noted that goals are often composed of several underlying goals. Feedback within this model accounts for underlying tasks—thus, goals lead to sub-goals, and intentions to sub-intentions within this closed system.
The creation of user goals and intentions, as well as, the evaluation of results is not always clear. Users must navigate many obstacles in order to understand and/or fully use a system. According to Norman, these obstacles lie within two user gulf{s}, the gulf of execution and the gulf of evaluation.

The gulf of execution is the mapping from the user’s intention (What the user wants to do—semantic distance) and action specification (How the user does the intention in this system’s context—articulatory distance) to the execution of action.

Likewise, the gulf of evaluation is the mapping from interpretation (How is the output interpreted—articulatory distance) and evaluation (What is the meaning of this output—semantic distance) back into the user’s goals.

During performance of a task, successful traversal of this goal-directed closed system model is the user’s job; the job of the designer is to provide an interface that minimizes gulf{s} (execution and evaluation) and obstacles.

**Long Term Knowledge Model**

Shneiderman (1979) introduces a model of programmer long-term knowledge, which was later expanded to include user knowledge (Shneiderman, 1987). This model is based on two components: semantic and syntactic knowledge. Figure II.14 graphically depicts the model.
Semantic knowledge is made of Task and Computer components. These components are general knowledge about the task and computer systems. The task component is broken down into actions and objects. It is imperative that the user have an understanding about the actions and objects (operators and operandi) of a given task domain, e.g., word processor objects (letters, paragraphs, words) and actions (move-text, change-word, indent). Like wise, the computer component also has actions and objects.

The syntactic component of the model represents system dependent knowledge. This includes specific knowledge on: command names, function key usage, etc. This syntactic knowledge is often gathered through memorization, and is often quickly lost.

**UAN**

The User Action Notation (UAN) is a user behavior representation (Hartson, Siochi, & Hix, 1990). The UAN provides a notation to describe physical user actions and system feedback. User actions range from atomic ‘key presses’ and ‘mouse movement’ to aggregates like ‘item selection.’

Figure II.15 gives an example of deleting a file within the Macintosh-OS environment. UAN divides the user actions, interface feedback and interface state into three separate columns. The symbols used within the notation serves as visually direct clues to actions. E.g., in UAN the notation \([\text{file} \_\text{icon}] \text{Mv}\) means to move to a file icon and mouse down. Further examples of notations and task descriptions can be found in Hartson, et al., (1990).
While the initial purpose of the UAN was for designer-implementor communication, it also allows for the description of user task structures through hierarchical decomposition.

UAN is currently evolving to meet new UAN-user needs (e.g., representations of user actions within three dimensions for virtual reality, more explicit representations of timing constraints). Sharratt (1990) has included memory and cognitive activities within a UAN-like framework.

11.3.6 Criticisms of HCI Models

The models outlined in the previous sections represent current modeling techniques used in HCI. Each technique attempts to enhance the user interface development process in one way or another; however, in many cases these techniques are not practical. This section identifies several practical, as well as theoretical, drawbacks of the various modeling categories.

The prediction based analytical techniques focus on results that estimate user task performance times. In order to gather these estimates, an entire system (or detailed sub-system) must be designed and represented using the technique’s formalism. Continually producing and maintaining these task representations (production-rule systems) is a highly time consuming venture within an iterative development process—often prohibitive and not cost-effective. This questions the value of these techniques within an iterative design cycle. Predictions are based on physical and cognitive actions of users, with place holders and estimated times for highly complex cognitive actions and tasks. These techniques do not model individual differences in users and user classes, and are often limited to expert error-free behaviors.

Knowledge analysis models, like prediction based models, are complex and difficult to use. These modeling techniques are often grammar based and require expertise. These grammars are difficult to construct and maintain especially in an iterative user interface development process where a single interface modification can dictate large changes to a descriptive grammar. Individual differences and user classes are often not addressed, but obviously there are
differences between expert- and novice-users’ knowledge structures. Also, results of these models do not indicate direct user interface design solutions.

The interface modeling techniques described are grammar based, and therefore, they are subject to the same criticisms as other grammar based models (knowledge analysis models). These criticisms include difficulty to construct and maintain, needed expertise, and multiple user classes. While these techniques yield user knowledge—it is unclear how to incorporate these analysis results into user interface redesigns.

Usability inspection techniques capture user interface usability problems in a designer ‘round table’ format. Designers who are knowledgeable in usability issues and design heuristics examine interfaces for usability error sites—often indicating the sites, but not interface design solutions. These methods rely on evaluator expertise and are often unstructured—both of which lead to problems in replicatability.

User models defined here are to informal to be used beyond the conceptual level, with UAN the exception. The UAN is used to describe user actions to designers. This notation, currently, does not support multiple user classes. This notation does find a nice fit as a communicational tool within the iterative development process, and it is currently used within TMM.

In many cases, criticisms presented here are pragmatic in nature. It is very hard for the ‘real world’ to use models that are not cost-effective or intuitive. These complaints may seem trite and superficial, but they are representative of industry’s protests, and they indicate these models’ inability to fit within an iterative development process. These complaints also reflect a need for synthesis of interface design solutions through modeling, as opposed to analysis for prediction estimates, knowledge structures, or usability problem sites.

The following section continues comment and criticisms on current HCI modeling techniques, and includes a comparison of our research model (TMM) with these other models.

II.4 COMPARISONS AMONG HCI MODELS

This section defines three grouped sets of characteristics that can be used for comparison of the various HCI modeling techniques. Each model is examined with relation to the each

Chapter II: Related Work
characteristic, and the results presented in a comparison matrix. The discussion of matrix includes a comparison of TMM against other HCI modeling techniques.

II.4.1 MODELING CHARACTERISTICS

All the models outlined previously attempt to serve the user interface development process. There are differences among each models' approach and corresponding results. For comparison and classification of these models, we define a model characteristic as a distinguishing attribute that a model may possess or support. This section identifies the modeling characteristics used to compare and contrast the different HCI models and techniques. The importance each characteristic has or should have in user interface development is presented elsewhere (Hartson & Mayo, 1993; Mayo & Hartson, 1993).

The characteristics are grouped into three categories: design process support, user class, and modeling type.

Design process support characteristics represents the various types of assistance each model provides for the user interface design process. These characteristics are: analysis orientation, synthesis orientation, performance prediction, and design support feedback.

Analysis Orientation
Models with this characteristic generally have an orientation based on analyzing representations of user tasks and interface designs. These analyses produce results that are often not readily incorporated back into the design process, i.e., the results are often descriptive. These analyses can be used within formative evaluation, but often the analyses are limited in practicality by the size and complexity of their representations.

Synthesis Orientation
Models that derive (or analyze towards) new user interface design objectives are synthesis oriented. Synthesis orientation requires that the model's results identify new user or interaction requirements. This orientation directs the designer towards new user-computer interactions that were not previously considered, e.g., discovery of alternative task methods.

Performance Prediction
Models that calculate timing estimates based on user physical and/or cognitive actions during task performance have this characteristic.
Design Support Feedback

Models whose results are directly applicable to designs or the design process have this characteristic. This requires that a model's analysis/synthesis results can be directly incorporated back into the design process. In other words, the results is part of a new design (e.g. design requirements, design specifications, or possible design solutions).

Overlap between related characteristics is possible. For example, it can be seen that GOMS is both an analysis method (analyzing task structure and user knowledge requirements) and a performance prediction method (calculates estimates of task performance times).

The second category of characteristics, user class category, represent the ability of models to include individual differences in users. These include the following characteristics: error-free performance, error performance, expert performance, novice performance, multiple user classes, user mental modeling, and user knowledge.

Error-Free Performance

This characteristic identifies models that support description/analysis of error-free user behavior and performance. The ability to describe error-free performance does not preclude the ability to describe errored performance.

Error Performance

Models that can describe/analyze user behavior and performance that includes user errors have this characteristic. This characteristic is much more problematic for many task analysis techniques.

Expert Performance

The ability to describe/analyze expert behavior and performance is represented by this modeling characteristic. Many models have this characteristic.

Novice Performance

The ability to describe/analyze novice behavior and performance is represented by this modeling characteristic. Novice users comprise a growing class of users that must be addressed, yet, many models only partially support them.

Multiple User Classes

Users are not homogenous. Models that allow description/analysis of multiple user classes per task have this characteristic. Models that center on expert error-free performance often cannot have this increasingly important characteristic.
**User Mental Modeling**

Models that rely on *user mental model* have this characteristic. A user mental model, as this research defines it, is a collection of user mental states, translations among the states, and cognitive operators to effect the state changes. This can include a representation of an interface or user knowledge within this context.

**User Knowledge**

Models that have this characteristic attempt to capture necessary user knowledge. This differs from user mental modeling in that this characteristic does not in any way attempt to represent the mental state of user, but only necessary user knowledge.

The *modeling type* characteristic category represents different alternative modeling styles. These characterize the way in which the modeling is performed apart from how it impacts design or includes users in the modeling. These characteristics include: *essentially analytical, essential empirical, global modeling, situational modeling, closed-loop modeling, and open-loop modeling.*

**Essentially Analytical**

Models that examine and manipulate representations of tasks and interfaces apart from empirical user measurements/observations are analytical. This is a common characteristic among several models.

**Essentially Empirical**

Models based on incorporating empirical evidence into description/analysis have this characteristic. Empirical evidence can be gathered in formative evaluation of the interface and can include: verbal protocol, timing values, user satisfaction surveys, and critical incidents.

**Global Modeling**

This characteristic represents models requiring description/analysis over all (or large portions of) user tasks or interfaces for modeling accuracy. This is very costly within an iterative design cycle.

**Situational Modeling**

Models that allow description/analysis of single tasks or interface features are situational and have this characteristic. These models are employed during formative evaluation due to their utility.
Closed-Loop Modeling

Closed-loop modeling views tasks as cycles involving user action-feedback-reaction processes. Models that approach tasks in this manner have this characteristic.

Open-Loop Modeling

Open-loop modeling views a task as performing actions to a desired result without benefit of feedback, or that feedback is irrelevant. Models with expert user behaviors often ignore feedback, the assumption is that experts do not need feedback or that experts always understand feedback.

Like design process support characteristics, modeling type characteristics can overlap.

II.4.2 Comparison Matrix

With the modeling characteristics defined in the previous section, a comparison matrix was derived. This matrix compares the models against each other in relation to the modeling characteristics.

This matrix (Figure II.16) presents the modeling characteristics on the vertical axis, and the modeling techniques on the horizontal axis. Each intersection defines the relationship between a model and a characteristic—with the relation defined as Fully-, Partially-, or Not-Supported.

A model fully supports a modeling characteristic if and only if the model’s definition provides explicit support of the characteristic. In other words, a model can only fully support a characteristic if its initial intent was to do so. E.g., GOMS provides full support for performance prediction because it was designed for that purpose.

A model partially supports a modeling characteristic if the modeling can be ‘converted’ to support the characteristic. A model’s results can be ‘converted’ such that it can be reinterpreted for different reasons. E.g., CLG models the interface command language, but it can also be thought of as an ideal user model of the system command language. This only provides an incomplete model of user knowledge, and thus CLG only partially supports the user knowledge modeling characteristic.

A model may have no mechanism or support a modeling characteristic. E.g., UAN does not currently support performance prediction.
![Comparison Matrix for HCI Modeling Techniques and TMM](image)

Each technique has modeling goals, and some modeling characteristics are orthogonal or irrelevant to these goals. This is neither good nor bad, it just reflects the different focuses each model has for supporting user interface development. These modeling characteristics draw our attention on the different models’ abilities, so that a proper model can be selected depending on the desired analysis results.
This matrix provides much data on the relationships among the modeling techniques and the modeling characteristics, and therefore, analysis of this matrix should be systematic. There are several ways to analyze this matrix: comparison within the modeling type grouping, comparison within the modeling characteristics groupings, and comparison of a single model to the rest.

The first way to examine the matrix, comparison within the modeling type grouping, is concerned with examining commonalities among modeling characteristics focusing on related modeling techniques (e.g., performance prediction models GCMS and CCT). Because the grouped techniques’ underlying representations are often similar, the models within a grouping are very similar with regard to the modeling characteristics. This is not a surprise, in fact, there are very few radical differences among the models within each group. These differences are often a function of modeling emphasis as opposed to the models’ representation or definition. There are, however, marked differences among the different groupings of modeling techniques—which is also not surprising. Models with different underlying representations (e.g., formal grammars, production rule systems, GTNs) and varying objectives (e.g., performance prediction, knowledge analysis) are bound to produce dissimilar ratings over the modeling characteristics.

The second method of matrix examination, comparison within the modeling characteristics groupings, focuses on the modeling characteristics. These characteristics can be examined singularly or in groups. Because the modeling characteristics are grouped by design process support, user class, and modeling type, there is little commonality over the results.

The third method, comparison of a single model to the rest, can help distinguish the uniqueness of a model. This type of matrix analysis is performed for TMM in the following section.

II.4.3 COMPARISON BETWEEN TMM AND OTHER HCI MODELING TECHNIQUES

This analysis compares TMM against the other HCI modeling techniques with regards to the defined modeling characteristics. This comparison deals with fundamental differences of power and focus among the models, as opposed to specific notation or representation differences. These comparisons are often between TMM and a group of models (e.g., performance prediction models, usability inspection methods, all models, etc.), and there are some instances where TMM is compared directly to another model. Also, because there is a large number of modeling characteristics to be considered, this comparison tries to bundle the characteristics together into pairs or groups.
Comparisons Based on Design Support Characteristics

TMM, like others, is analysis oriented. This is not surprising—what are these models for if not to analyze interface designs? TMM, however, is synthesis oriented unlike the others—TMM synthesizes new user interface design requirements for use in redesign. Performance prediction methods generate time estimates, but how does this synthesize design solutions? An argument could be made that alternative interface designs can be compared based on performance time estimates, but this is unrealistic due to time involvement. Knowledge analysis models synthesize user knowledge—but this knowledge is only helpful if it can be tied back to the design in some meaningful manner—so these models, in general, only partially support the design process in a synthesis orientation. Interface modeling techniques examine the command or articulation structure, which occasionally provides partial support for synthesis. Usability inspection models are concerned with locating user problems based on probable usability error sites, and therefore, these models concentrate on analysis for errors, as opposed to synthesis of solutions.

TMM is not a performance prediction model; however, future research on TMM includes performance predictions. Several models' main focus is performance prediction (e.g., GOMS, CCT) while other models provide partial support (as a side-effect) for performance predictions (e.g., ETAG, UAN).

TMM synthesizes user interface design requirements representing analysis artifacts that feedback directly into the design process—hence, TMM fully supports the design support feedback modeling characteristic. Other models provide analysis in terms of estimated numbers, identified usability problems, required user knowledge, but these results still need translation into user interface design requirements, specifications or design solutions. Also, models that require representations of entire system often cannot provide feedback to the design process in a timely fashion.

Comparisons Based on User Class Characteristics

User populations are often composed of users with widely variable skills and abilities. Classifying users into similar groups can help identify special user needs, and in turn, increase the usability of interfaces. TMM uses user class profiles and definitions to increase its quality of analysis.
TMM provides analysis support for both error-free and erred performance. Performance prediction models and grammar based modes are most often limited to error-free performance. Performance prediction models, in order to get accurate prediction times, error is often overlooked, or a representative constant is factored into the prediction equation (which still ignores the actual error). Grammar based models are also restricted to error-free performance.

TMM provides analysis support for novices and experts for both error-free and erred performance. Again, the performance prediction models and the grammar based models are limited—in this case to supporting expert users only.

It is naive to think that there are only two different user classes in every population: novice and expert. There are many different user classes within an given general population. Modeling and analysis should take these different classes into consideration to improve usability. TMM treats different user classes as an added dimension to its analysis. User class definitions are integral to TMM analysis. Many other models have no support for multiple user classes, and a few have partial support.

By our usage, the modeling characteristic of user mental modeling and user knowledge represent the process and product in terms of knowledge, respectively. TMM, while it captures the necessary user knowledge needs, makes no claims about the users' mental processes. Others attempt to model users' cognitive processes, but this seems to be an unnecessary analysis burden in user interface design. Empirical evidence about users cognitive abilities serves as an adequate and cost-effective base for much of the needs within user interface design.

**Comparisons Based on Modeling Type Characteristics**

The modeling type characteristics represent different approaches to modeling and analysis. Each of the characteristics represent an ability that each model should posses. Of course, this is not possible with many of the models, but TMM attempts to accommodate all these characteristics.

TMM is essentially empirical in nature. TMM task descriptions are based on observations of users (when possible), and the user class definitions are also derived from empirical evidence. TMM is partially analytical in that it provides methods to derive necessary user knowledge. Many models are essentially analytical, and do not take full advantage of formative evaluation qualitative and quantitative findings during the iterative user interface design process. It is easily seen that performance prediction, interface modeling, and knowledge analysis models are
essentially analytical, but it is not so apparent that usability inspection methods are also analytical—designers are analyzing interfaces apart from empirical evidence.

TMM can focus its analysis and synthesis on a single usability problem or a host of usability problems; hence, TMM supports both global and situational analysis. Most, if not all, techniques provide for global analysis, but only a few provide mechanisms for situational evaluation. The need for situational evaluation should be clear—global evaluation is extremely costly (in time and effort) in the very dynamic, iterative user interface design process. Performance prediction models, like GOMS and CCT, require the entire system (or command path) to be modeled for a single prediction, and if the interface design changes, a costly re-analysis must be done. On the other hand, usability inspection models, like TMM, can focus the analysis attention on a single user task.

Finally, one of the most important components in any human-computer interaction is input and output. Currently many techniques only model the human-output/system-input side of the interaction. TMM also models this side, but TMM goes one further and models the system-output (feedback)/human-input side. This is because TMM approaches interaction as a closed loop process—where system feedback can drive the task(s). Models that only consider the human-output/system-input interaction side view the interaction as an open looped process. TMM can view interaction both ways, but I feel that it is a waste of power not to consider system feedback (the system-output/human-input interaction side).

**TMM's Rally Cry**

TMM is first and foremost a synthesis-oriented task/user-centered empirical model that aids the user interface design process. This model captures necessary user knowledge for performing tasks, in order to synthesize new interface design requirements. TMM is designed to fit into the iterative user interface development process. TMM is used during initial design and formative evaluation by designers and analysts. Situational analysis is stressed in TMM for a cost-effective alternative to global analysis—however, TMM supports both styles of analysis. TMM also includes user class profiles and definitions as an added dimension its analysis.

The other HCI modeling techniques discussed here provide some, but not all, of the necessary modeling characteristics that designers and analysts need.
II.5 CONCLUSIONS

This chapter reviews several current HCI modeling techniques. These techniques were separated into several different groups, and each group and model was discussed and an example was given. Then a taxonomy of comparison characteristics were defined. These characteristics serve as comparison points among the models within a comparison matrix. Finally, a comparison based on the modeling characteristics was performed between TMM and the other HCI models.
III THE TASK MAPPING MODEL

This chapter identifies, discusses, and motivates theoretical rationale underlying development of the task mapping model (TMM). The purpose of this chapter is to introduce the reader to the TMM by marshaling arguments and discussions about user interface design and task analysis. The TMM is also discussed in Task Mapping Model Analysis Manual: A Reference and in Situational Analysis with the Task Mapping Model (TMM): A Tutorial (Mayo, 1993a; Mayo, 1993c).

A general understanding of the user interface development process is required for discussing the rationale behind the TMM (refer to Chapter II, Section II.1: User Interface Development Process). We briefly revisit this topic here with emphasis on the TMM life cycle within the user interface development process.

Next, the discussion turns towards the framework of TMM task descriptions—a multi-domain approach to analysis of user tasks. The components of the framework are introduced and supported. The components represent objects that populate the various domains, associative mappings, and user knowledge requirements. Finally, this chapter addresses special issues associated with task description and analysis.
III.1 TMM LIFE CYCLE

The function of the TMM analyses, as outlined in Chapter I, is to derive new user interface design requirements (UIDRs) for use in initial design or re-design. By synthesizing UIDRs, the TMM bridges the gap between systems analysis, evaluation and re-design (refer to Chapter II section II.1). Figure III.1 illustrates the context and function of TMM analyses.

![Diagram of the iterative user interface development process with TMM analysis]

**FIGURE III.1. THE ITERATIVE USER INTERFACE DEVELOPMENT PROCESS WITH TMM ANALYSIS**

Briefly, the user interface development process model used by this research is similar to other commonly accepted interface development process models (Carroll, Kellogg, & Rosson, 1991; Gould & Lewis, 1983; Gould, 1988; Hartson & Hix, 1989; Hix & Hartson, 1993; Norman & Draper,
This iterative model begins with systems analysis which includes functional, task, and user analyses. The results of systems analysis are used as input to TMM analysis which produces design requirements. These design requirements are used in interface design/re-design to produce design specifications. Interface prototyping/implementation uses the design specifications to produce a prototype which is then evaluated. The evaluation identifies interface usability problems which, coupled with systems analysis results, form the input to TMM analysis. The TMM analysis produces new user interface design requirements (UIDRs) which are used in another iteration of the cycle.

Using the TMM life cycle is a three-phase process: (1) description of task and interface, (2) analysis of user knowledge requirements, and (3) synthesis of new user interface design requirements from unsupported user knowledge requirements. Figure III.2 depicts the TMM analysis life cycle (with reference to its place within the context of user interface development).

**FIGURE III.2: TMM LIFE CYCLE WITHIN ITERATIVE USER INTERFACE DEVELOPMENT**

Phase 1 of the TMM life cycle, *TMM Task Description*, analysts use results from systems analysis (i.e., functional, task, and user analyses) and interface evaluation (i.e., user observables and current interface design) to analyze and produce TMM task descriptions. (Task descriptions are generated only for user tasks selected due to complexity or user error occurrence—this has the effect of limiting the scope, and increasing the efficiency of TMM analyses.) The task descriptions are then used in conjunction with current interface designs and user class profile(s) to categorize user knowledge requirements within phase 2—*TMM Knowledge Analysis*. Finally, in phase 3,
analysts synthesize new user interface design requirements (UIDRs) from user knowledge requirements identified previously.

Each TMM phase produces analysis results that are used as inputs to subsequent phases. Yet, analysis results can have other uses outside the TMM life cycle. For example, task descriptions from phase 1 can be used by analysts and designers for examining user actions at various levels of abstraction. Also, results of the second phase help to characterize the interface in terms of user knowledge needs and support. Knowledge requirements can be saved for future reference and used within training materials. The new UIDRs, generated in phase 3, feed directly back into the user interface development cycle. (Discussion of these phases is continued in Section III.4: TMM Methodologies.)

There are several ways the TMM can be employed within user interface development. The work reported here focuses on using the TMM as a redesign tool; however, the TMM can also be used within initial design and as a walkthrough technique, as discussed below.

**The TMM as a Initial Design Technique**

The TMM can help redefine and crystallize initial UIDRs during beginning development phases. By using the TMM to describe and analyze tasks, apart from user interface designs, designers and analysts benefit from more complete user task models which incorporate more user abstraction levels. Using TMM for initial design follows the three-phase process described above; however, having no current interface design has the effect of limiting task descriptions to all but the lowest abstraction levels—physical articulation. Even with this limitation, TMM analyses provide detailed information about user task structures and knowledge requirements at higher abstraction levels.

**The TMM as a Walkthrough Technique**

Task descriptions, within the TMM framework, can also be used in walkthrough settings. That is, task descriptions can be inspected by groups using a structured approach. For example, analysts present their TMM task descriptions to other HCI professionals who simulate user interactions and reconceptualizations. Using walkthroughs has the effect of more widely distributing user task descriptions and interface design ideas/features to verify assumptions incorporated within the design. This improves the design by bringing more user interface developers into the iterative process of evaluation and re-design. A primary strength of walkthroughs is that they
provide immediate feedback and criticisms which spurs re-design and analysis. Further, analysts can use walkthroughs to help determine correctness and completeness of their task descriptions.

Walkthroughs are not the only form of technical reviews available to analysts. The reader is directed to Freedman, et al. (1990) for a more complete discussion of review principles and techniques.

III.2 TMM TASK DESCRIPTION FRAMEWORK

Task description is crucial to all phases of the TMM. This section discusses the TMM task description framework, i.e., domain structures, domain items, domain item languages, mappings, user knowledge, and knowledge requirements.

III.2.1 DOMAINS

One of the major components of TMM analyses is its task description framework. TMM task descriptions, as in other task modeling techniques (Moran, 1981; Tauber, 1990), are abstracted across several domains. Domains represent task abstraction levels based on both users and physical reality. Users employ abstraction to understand and conceptualize tasks. Physical reality supports users' abstraction because, although tasks are often conceptualized within a problem domain apart from computers, using computers to perform tasks imposes a computer level abstraction on the user.

To illustrate user task abstraction, consider the user task GET STOCK QUOTES. This task exists within the problem domain of STOCK INVESTMENT as a common task for investors who have gambled money in the stock market. To accomplish this task investors can: (1) contact a brokerage service and speak with a broker, (2) consult a newspaper, (3) use an automated on-line service to get stock quotes, or (4) contact individual stock issuers (i.e., companies listed on the exchange), etc. Each of these alternatives requires knowledge from different interaction domains—see Figure III.3. For example, to get stock quotes from a newspaper, the investor must understand the form in which the quotes are presented, as well as how to find the quotes within the paper.
Our interest, of course, is modeling investors using an on-line service to get stock quotes. In this case investors (i.e., users) must interact in both an investing domain and an on-line service (stock quote system) domain. The investing domain is a specific problem domain representing user investing tasks directly in real-world terms. The on-line service domain is a specific computer domain that represents user investing tasks in terms of computer artifacts and operations.

Interacting with computers involves working in a virtual world created by the computer. In these interactive systems, actions are performed using specific input devices to generate desired results. For our example, the investor must know how to convey user tasks to the system (e.g., get a stock quote, buy 1000 shares of IBM stock). Also, the investor needs to know how to interpret results. Figure 3.4 depicts the dual need of investors.
FIGURE III.4. USING A COMPUTER (ON-LINE SYSTEM) TO GET STOCK QUOTES

The two user concerns identified within Figure III.4 correspond to *execution* and *evaluation paths* from Norman’s Theory of Action (Norman, 1986; Hutchins, Hollan, & Norman, 1986). Incorporating Norman’s model into the picture in Figure III.4 yields a more complete model of interaction for our example task—see Figure III.5. Norman’s model helps to separate the different activities that investors, as users, must perform while using an on-line system.
FIGURE III.5. HOW NORMAN'S THEORY OF ACTION IS APPLICABLE
INCLUDES EXCERPTS FROM (NORMAN, 1986; HUTCHINS, ET AL., 1986)

To understand the questions posed in Figure III.4 and modeled in Figure III.5, users of interactive systems utilize two abstraction levels for the computer domain: computer semantic domain and computer syntactic domain. Users conceptualize tasks in generic computer environments (computer semantic domain) and also in platform-, interaction style-, and articulation-specific computer environments (computer syntactic domain).

The TMM uses domains much as do CLG (Moran, 1981) and ETAG (Tauber, 1990). Specifically, TMM has the four domains: Problem, Computer Semantic, Computer Syntactic, and Articulation.
These domains also reflect a structure similar to a model of long term knowledge (Shneiderman, 1987).

The **Problem Domain** represents the users' real world task space; as such, all of its domain items represent real world entities. The problem domain is equivalent to the task domain within CLG, but we changed the name to avoid confusion—all TMM domains represent task domains (only each at a different level of abstraction). For example, the **DOCUMENT PREPARATION** problem domain includes items like: **PARAGRAPH, WORD (objects); CUT, DELETE (actions); ADD TABLE OF CONTENTS, EDIT MANUSCRIPT** (sub-tasks).

The **Computer Semantic Domain** contains computer representations of real world entities (from the problem domain). For example, a **DICTIONARY DATABASE** is the computer representation of the **DOCUMENT PREPARATION** problem domain **DICTIONARY object**.

The **Computer Syntactic Domain** differs from the computer semantic domain in that representations at this level use specific computer related interface entities with grammatical ordering of interactions. In other words, the computer semantic domain describes **what to express** to the computer, whereas the computer syntactic domain describes **how to express** to the computer. Thus, items in the computer syntactic domain are somewhat specific to general interaction styles and techniques. For example, the computer semantic domain task **INDICATE DICTIONARY (what to express)** is represented within the computer syntactic domain as **RECOGNIZE <DICTIONARY FILENAME>, RECOGNIZE <SELECT COMMAND NAME>, and EXECUTE SELECT DICTIONARY USING <DICTIONARY FILENAME> AND <SELECT COMMAND NAME> (how to express)**.

Finally, the **Articulation Domain** is composed solely of user physical actions, hence it is the lowest level of abstraction. The TMM employs the User Action Notation (UAN) to describe user and system behavior (Siochi & Hartson, 1989; Hartson, Siochi, & Hix, 1990; Hartson & Gray, 1992).

Using domain abstraction for task description affords TMM analysts the ability to characterize and analyze user tasks at varying levels of abstraction. Describing tasks in this way makes it possible to show user tasks (and knowledge requirements) at levels of abstraction corresponding to the TMM domain framework.
III.2.2 **DOMAIN ITEMS**

Each domain specifies or redefines user tasks at the domain's level of abstraction. Each domain can be thought of having its own language or set of items used for task description. *Domain items*, as we refer to them, include *objects*, *operations*, and *sub-tasks* are written in domain specific language.

*Objects* represent physical or conceptual *entities* that are defined within a specific domain. For example, *buildings* are objects within the *architecture* problem domain; *files* are objects within the computer semantic domain; and *menu items* are objects within the computer syntactic domain.

*Operations* are representations of actions applicable within a specific domain's abstraction level. For example, *designing* is an operation within the *architecture* problem domain; *data transformation* is an operation within the computer semantic domain; and *menu item selection* is an operation within the computer syntactic domain.

While domain objects and operations can be defined and analyzed independently of each other, often it is useful to describe them in the context of one another. Sub-tasks provide a context for objects and operations, because sub-tasks are simply unordered-pairs of objects and operations. For example, *design a building* is a sub-task within the *architecture* problem domain; *move a file* is a sub-task within the computer semantic domain; and *select <open> menu item from <file> menu* is a sub-task within the computer syntactic domain.

Domain items are the basic components of task description within each domain. However, the TMM standardizes on specific terms and notations for consistency among task descriptions.

III.2.3 **DOMAIN LANGUAGES**

Notations based on arbitrary vocabularies and grammars (i.e., languages) are fraught with ambiguities. Therefore, it is clear that language standardization benefits task description and analysis in the following ways:

- Standard languages and notations are well defined.
- Standard languages and notations remove terminology ambiguity. In other words, ambiguities arising from natural-language task descriptions are removed because only one term can be used for one action or object.
• Language/notation standardization allows for a more concise and precise definition of terms.

• Standard languages and notations reduce structure ambiguity.

For these reasons, the TMM uses two very different languages depending on the abstraction level of the description (i.e., domain). The first language is used in all domains with the exception of the articulation domain. It provides a vocabulary which is used to describe domain items (i.e., operations, objects, and sub-tasks). The second language is used only in the articulation domain as a notation for describing user actions and system responses.

Domain Vocabulary

The TMM employs an adaptation of the taxonomy reported by Lenorovitz, et al. (1984), as a standard vocabulary for describing domain items—particularly operations. Adopting this taxonomy helps eliminate ambiguities by providing a common, well-defined descriptive vocabulary.

Lenorovitz’s taxonomy is divided into four sub-taxonomies of the User-System Interface: computer-output, computer-internal, user-input, and user-internal taxonomies. The user-input and user-internal taxonomies are used in the TMM—see Table III.1. The taxonomies are arranged hierarchically so that the more general concepts are simple unions of the less general items. For example, the user-input action of replicate is either a copy or instance action, but also can be used to refer to both; whereas create is either associate, introduce, assemble, or replicate, or their union.
The TMM uses these two taxonomies as vocabulary bases for the problem-, computer-semantic-, and computer-syntactic domains. However, TMM task descriptions are not limited to these vocabulary sets. Any vocabulary term, regardless of its abstraction within the vocabulary hierarchy, can be used. Also it is interesting to note, on initial inspection the more general vocabulary words seem to be used within the problem and computer semantic domains, while less general vocabulary words are used within computer syntactic and articulation domains. More research on these taxonomies for use as task description vocabulary bases is needed.

Definitions of words are given in Tables III.2 and III.3 (still showing taxonomical hierarchies).
<table>
<thead>
<tr>
<th>Process</th>
<th>Action</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PERCEIVE</td>
<td>ACQUIRE</td>
<td>Discover or notice an occurrence (usually unsolicited).</td>
</tr>
<tr>
<td>SEARCH</td>
<td>DETECT</td>
<td>Purposeful exploration or looking for specified item(s).</td>
</tr>
<tr>
<td>SCAN</td>
<td>EXTRACT</td>
<td>Glance over quickly, usually looking for overall patterns or anomalous occurrences (not details). Directed, attentive reading, observing, or listening with the purpose of gleaning the meaning or contents thereof.</td>
</tr>
<tr>
<td>CROSS-REFERENCE</td>
<td>IDENTIFY</td>
<td>Accessing or looking up related information usually by means of an indexing or organized structuring scheme set up for that purpose. Roughly classify or differentiate an entity in terms of a gross level grouping or set membership—frequently on the basis of only a limited number of attributes.</td>
</tr>
<tr>
<td>RECOGNIZE</td>
<td>DISCRIMINATE</td>
<td>Specific, positive identification of an entity.</td>
</tr>
<tr>
<td>MEDIATE</td>
<td>ANALYZE</td>
<td>Classify or sort one or more entities into specific sets or groupings, usually on the basis of a well-defined classification scheme. Reckon, mentally compute, or computationally determine.</td>
</tr>
<tr>
<td>CATEGORIZE</td>
<td>CALCULATE</td>
<td>List or specify the various components of a grouping.</td>
</tr>
<tr>
<td>ITEMIZE</td>
<td>ITEMIZE</td>
<td>Tally or enumerate the frequencies or values of the members of an itemized list or table.</td>
</tr>
<tr>
<td>TABULATE</td>
<td>SYNTHESIZE</td>
<td>Mentally gauge, judge, or approximate, often on the basis of incomplete data.</td>
</tr>
<tr>
<td>ESTIMATE</td>
<td>INTERPOLATE</td>
<td>Assign an approximate value to an interim point based upon knowledge of values of two or more bracketing reference points.</td>
</tr>
<tr>
<td>TRANSLATE</td>
<td>TRANSLATE</td>
<td>Convert or change from one form or representational system to another according to some consistent mapping scheme.</td>
</tr>
<tr>
<td>INTEGRATE</td>
<td>INTEGRATE</td>
<td>Pull together, and mentally organize a variety of data elements so as to extract the information contained therein.</td>
</tr>
<tr>
<td>FORMULATE</td>
<td>FORMULATE</td>
<td>Generate and put together a set of ideas so as to produce an integrated concept or plan.</td>
</tr>
<tr>
<td>PROJECT OR EXTRAPOLATE</td>
<td>PROJECT OR EXTRAPOLATE</td>
<td>Assign an approximate value to a future point based upon the value(s) of preceding point(s).</td>
</tr>
<tr>
<td>ASSESS</td>
<td>ASSESS</td>
<td>Consider two or more entities in parallel so as to note relative similarities and differences.</td>
</tr>
<tr>
<td>COMPARE</td>
<td>COMPARE</td>
<td>Determine the value, amount, or worth of an entity, often on the basis of a standard rating scale or metric.</td>
</tr>
<tr>
<td>EVALUATE</td>
<td>DECIDE</td>
<td>Arrive at an answer, choice, or conclusion.</td>
</tr>
<tr>
<td>COMMUNICATE</td>
<td>TRANSMIT</td>
<td>Signal to a specific recipient or set of recipients that a message is forthcoming.</td>
</tr>
<tr>
<td>CALL</td>
<td>ACKNOWLEDGE</td>
<td>Confirm that a call or message has been received.</td>
</tr>
<tr>
<td>RESPOND</td>
<td>RESPOND</td>
<td>Answer or reply in reaction to an input.</td>
</tr>
<tr>
<td>REQUEST</td>
<td>REQUEST</td>
<td>Solicit, query, or ask for.</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>RECEIVE</td>
<td>Get, obtain, or acquire an incoming message.</td>
</tr>
</tbody>
</table>

Chapter III: The Task Mapping Model
| TABLE III.3. DEFINITIONS FOR LENOROVITZ, ET AL. (1984) USER-INPUT TAXONOMY |
|-----------------------------|---------------------------------------------------------------------|
| CREATE | ASSOCIATE | NAME | Give title to or attach label to for purposes of identification/reference. |
| | | | GROUP | Link together or associate for purposes of identification. |
| INTRODUCE | INSERT | Make space for and place an entity at a selected location within the bounds of another such that the latter wholly encompasses the former, and the former becomes an integral component of the latter. |
| ASSEMBLE | AGGREGATE | Combine two or more components so as to form a new composite entity. |
| | OVERLAY | Superimpose one entity on top of another so as to affect a composite appearance while still retaining the separability of each component layer. |
| REPLICATE | COPY | Reproduce one or more duplicated of an entity (no links to master). |
| | INSTANCE | Reproduce an original (master) entity in such a way as to retain a definitional link to the master—i.e., such that any subsequent changes or modifications made to the master will automatically be reflected in each and every instance created therefrom. |
| INDICATE | SELECT (POS/OBJ.) | Opt for or choose an entity (e.g., a position or an object) by pointing to it [plus other possible actions]. |
| INDICATE | REFERENCE | Opt for or choose an entity by invoking its name. |
| ELIMINATE | REMOVE | Cut | Remove a designated portion of an entity and place it in a special purpose buffer (residual components of the original entity usually close in around hole left by cut-out portion). |
| | | DELETE | Remove and (irrevocably) destroy a designated portion of an entity. |
| STOP | SUSPEND | Stop a process and temporarily hold in abeyance for future restoration. |
| | TERMINATE | Conclude a process such that it cannot be restarted from the point of interruption, only by complete re-initiation. |
| DISASSOCIATE | RENAME | Change an entity’s title or label, without changing the entity itself. |
| | UN-GROUP | Eliminate the common bond or reference linkage of a group of entities. |
| DISASSEMBLE | SEGREGATE | Partition and separate an entity into one or more component parts such that the structure and identity of the original is lost. |
| | FILTER | Selectively eliminate one or more layers of an overlaid composite. |
| | SUPPRESS | Conceal or keep back certain aspects or products of a process without affecting the process itself (i.e., affects appearance only). |
| | SET-ASIDE | Remove entire contents of current (active) work area and store in a readily accessible buffer (for future recall). |
| MANIPULATE | TRANSFORM | Manipulate or change one or more of an entity’s attributes (e.g., color, line type, character font, size, orientation) without changing the essential content of the entity itself. |
| ACTIVATE | EXECUTE __ _FN. | Initiate or activate any of a set of predefined utility or special purpose functions (e.g., sort, merge, calculate, update, extract, search, replace). |
Domain Vocabulary: UAN

Following the arguments for language consistency and precision outlined above, the TMM employs the User Action Notation (UAN) within the articulation domain (Siochi, et al., 1989; Hartson, et al., 1990; Hartson, et al., 1992). The UAN helps to provide a common language by which designers and analysts can communicate about low-level or articulatory-level user actions.

Meta- and Micro-Language Relationships

As mentioned above, each domain contains an abstract description of the users' tasks. Therefore, all the representations at the various abstraction levels are related. It is interesting to note the relationships among the domain description languages. Each domain description language forms a macro-language for the following, less abstract domain language. For example, the computer semantic domain language forms a macro-language for the computer syntactic domain language—just as the computer syntactic domain language is a macro-language for the articulation level language.

Likewise, each domain language forms a micro-language for the next more abstract domain language. For example, the computer syntactic domain language forms a micro-language for the computer semantic domain language.

Domain Grammars

The grammars used for TMM task descriptions are determined by the domain abstraction level being described—i.e., while each TMM domain may use a similar vocabulary (e.g., Lenorovitz, et al. (1984)), each has a different grammar. To illustrate the differences, consider the task ELIMINATE PARAGRAPH within the DOCUMENT PREPARATION problem domain. The equivalent computer semantic description is the unordered pair (REMOVE, <PARAGRAPH>). The computer semantic description is unordered because there is no indication which command selection style is used at this level of description (e.g., action selection before object selection, or object selection before action selection\(^1\)). The computer syntactic description, which is platform-specific, provides descriptions that specify the command selection style, for example it could be action-object selection as in: INDICATE <DELETE>, INDICATE <PARAGRAPH>, EXECUTE <INDICATED DELETE OPERATION>. The grammars fall into the structures illustrated in Table III.4.

---

\(^1\)These are also referred to as VERB–NOUN and NOUN–VERB command selection styles, respectively.
The TMM includes other dimensions of task performance within its task descriptions, such as task timing relationships. Therefore, these are not complete grammars—they only represent the grammatical use of vocabulary within each TMM domain. The grammar for the articulation domain is defined by the UAN.

### 3.2.4 TRANSLATIONS AND MAPPINGS

At this point, we have introduced the abstraction domains of the TMM, items that exist within the domains, and languages for these items. We know that each domain represents user tasks at different abstraction levels—which prompts the question: How are domain items related across domain boundaries?

Domain items are connected or linked across domain boundaries by mappings. These mappings represent the reconceptualization that users must do in order to perform the task. For example, users are often required to map **PERSONAL DATA** (or some other problem domain dataset) into **DATABASE FILE** (computer semantic domain) and further into **FILENAME** FILE (computer syntactic domain), as in:

\[
\text{PERSONAL DATA} \rightarrow \text{DATABASE FILE} \rightarrow \text{FILENAME FILE}
\]

where **Problem Domain Item** (Problem Domain Item) **Computer Semantic Domain Item** (Computer Semantic Domain Item) **Computer Syntactic Domain Item** (Computer Syntactic Domain Item)

Read (→) or (←) as maps to
Users must be able to reconceptualize problem domain tasks through computer domains, into the articulation domain, and back again—corresponding to user execution and evaluation paths. Knowing about and being able to perform reconceptualizations of domain items, we believe, forms the crux of task performance. Thus, mappings (i.e., reconceptualizations) identify barriers to computing that users must be able to cross. In this example, users would be hard pressed to accomplish any task involving personal data without understanding the mappings to the actual file name (<FILENAME> FILE).

III.2.5 Knowledge Requirements

Each mapping has an associated set of knowledge requirements representing what users need to know to perform reconceptualizations of domain items across domains. It should be emphasized that the TMM presents knowledge as requirements for task performance and not within the context of a cognitive processing or belief model.

Mappings are defined between all pairs of abstraction levels of the TMM domain framework, and as such knowledge requirements are also defined between all domain-pairs. For example, mapping the computer semantic domain item FILE to the computer syntactic domain item <FILENAME> requires knowledge associated with locating or recalling the name of the file—knowledge particular to the mapping of items from the computer semantic to the computer syntactic domains.

This section discusses knowledge representations and how to work with user knowledge requirements.

User Knowledge Categories

The TMM identifies three categories of knowledge: factual, conceptual, and procedural—much like a schema (Evans, 1988). In our experience, these categories encompass all user knowledge requirements for reconceptualizing tasks within the TMM domain framework.

Factual knowledge represents simple propositions or assertions, e.g., BIRDS FLY, DISKS ARE DAMAGED BY MAGNETIC FIELDS, ADVERTISING CLIENT LIST IS IN CLIENT-FOLDER.

Conceptual knowledge is generalized knowledge about aggregates of factual knowledge, e.g., knowledge of UNIX is made up of knowledge of a large collection of facts plus generalized knowledge of rules, principles, and conventions that govern the facts. Conceptual knowledge is
inherently set-theoretic and often within a hierarchical structure—e.g., KNOWLEDGE ABOUT DISK USAGE is subordinate to KNOWLEDGE ABOUT DATA STORAGE DEVICES.

Procedural knowledge, as used within the TMM, is knowledge about possible courses of action, task ordering and structure, i.e., knowledge about how to do things—e.g., KNOWLEDGE ABOUT HOW TO LOCATE DATABASE WITH QUARTERLY REPORT, KNOWLEDGE ABOUT HOW TO LOG-OUT OF THE SYSTEM. There are two logical components to procedural knowledge: knowledge of possible task paths\(^2\) and knowledge needed to choose a path correctly. Knowledge of possible task paths can be thought of as user knowledge of alternative methods or different ways to perform a task. For example, users may not know how to perform a task, that is, users are unaware of the task structure and/or methods. On the other hand, users may know how to perform a task using one method, but be unaware of alternative methods that increase performance, are more cognitively direct, or that are more appropriate given starting conditions.

The second component of procedural knowledge is knowledge to correctly choose a path. For example, the task LOCATE <FILENAME> IN WINDOW <WINDOW NAME> has the outcomes of FOUND and NOT FOUND. In this example, users must know that there are two possible task outcomes (i.e., knowledge of possible task paths) and also that a choice is necessary (i.e., knowledge to correctly choose a path).

**Knowledge Representation in the TMM**

Representing and manipulating knowledge (factual, conceptual, and procedural) is very important for TMM analyses, and it should be stressed again that the TMM outlines knowledge requirements without providing or relying on an underlying cognitive processing model. There are many knowledge representation techniques presented in the literature, e.g., rules, heuristics, schemas, scripts, semantic networks, schemas, scripts, frames, and numerous mental models (Evans, 1988; Johnson-Laird & Wason, 1977; Johnson-Laird, 1983; Johnson-Laird, 1993; Shank, 1977). Knowledge representation techniques vary greatly in complexity of description and difficulty of use.

While developing the TMM, we opted for a much less rigid knowledge representational technique. The TMM's knowledge representation is based on natural language—i.e., English.

\(^2\)Other models of interaction provide for alternative method selection, e.g., GOMS (Card, Moran, & Newell, 1983). However, others do not address our concern that users may not be aware of the existence of alternative methods. TMM does address this concern.
Natural language is the only medium by which all knowledge requirements can be represented in a practical fashion by TMM analysts. However, using natural language leads to certain inherent language problems associated with ambiguity and verbosity.

One attempt to reduce ambiguity and verbosity occurred early this century during the great wars. A movement started towards a single global language—one in which all mankind can communicate. This language, Basic English, is a streamlined version of English consisting of only 850 words—100 operations, 400 general things, 200 picturable [sic] things, 100 general qualities, and 50 opposites (Ogden & Richards, 1925; Johnsen, 1944; Richards, 1943). Basic English is relevant to this research because it was an attempt to reduce the highly complex and ambiguous set of English terminology into a simple set. This simple set (with grammatical rules) can be used for defining the TMM user knowledge. However, this set would need to be augmented due to historical changes—like the invention of the computer.

Another attempt to reduce language into a set of common vocabulary was made by Gernot Wersig (Wersig & Neveling, 1976). Wersig addressed the need for a set of common vocabulary defined over several languages—English, French, German, Spanish, and Russian. Wersig's taxonomy provides a standardized vocabulary for use in documentation. This work presents the vocabulary with definitions in all the languages. Like Basic English, this taxonomy can be used to specify TMM user knowledge.

Both Basic English and Wersig's documentation taxonomies provide sub-languages that the TMM could standardize on to describe user knowledge. The use of these sub-languages can provide constraints to make descriptions more tractable; however, the range of problem domain vocabulary requires a much less rigid approach. The following guidelines should be used when defining a knowledge representation language:

- Each vocabulary word is defined within a single glossary.
- Each vocabulary word has one and only one definition (reduces ambiguity).
- Words selected for the vocabulary are commonplace terms.
- All user/system actions are defined for a specific interaction style/interface artifact/inputs.
- All user/system actions are defined by the set of user tasks.
- Grammars used for knowledge representation should be clear, concise, and specified formally.
Interface specialists using the TMM should employ English reduced in both grammar and vocabulary based on these guidelines. The definition of a language for knowledge representation, apart from these guidelines, is outside the scope of this research.

**Manipulating Knowledge Representations**

The TMM uses knowledge in two ways: (1) to describe the scope of users’ knowledge within the user class profile, and (2) to represent knowledge requirements for task performance associated with mappings in TMM task descriptions—e.g., see Table III.5. User class profiles characterize the knowledge which users bring to the interaction, while knowledge requirements represent what users need to know to perform their tasks. In an ideal situation, users would have all necessary knowledge to perform their tasks, i.e., knowledge identified within user class profile covers knowledge requirements identified within TMM task descriptions. However, in many cases these knowledge sets have disjoint elements which imply users do not have and cannot get the knowledge necessary for task accomplishment. Both uses of knowledge are critical to TMM analyses.

Analysts use TMM task descriptions to identify user knowledge requirements. These knowledge requirements are categorized as supported or unsupported. Knowledge requirements are supported if (1) users have the knowledge through experience or training, or (2) the interface provides direct support for user acquisition of the knowledge. If both these conditions fail, the knowledge requirement is unsupported. Hence, it is imperative that analysts be able to deduce and infer information about user knowledge from the user class profile to compare against identified knowledge requirements. This ability allows analysts to categorize user knowledge requirements with relation to the user class profiles.

Determining support is equivalent to judging the degree of commonality between elements across the two knowledge sets. An analogy from mathematics is a characteristic function, say:

\[ X = \{ x \mid x \text{ is even } \land x \text{ is integer} \} \]

The function \( X \) defines a set of elements based on two predicates. Testing for membership, say whether 6 is a member of \( X \), is the logical result of the predicates—in this case yes. This type of notation formally specifies membership. Less formally (and less precisely), analysts must do the same with knowledge sets. Since it is less formal, deduction and inference are augmented with analyst judgment. Returning to the example in Table III.5, the analyst must be able to deduce or
infer knowledge requirement support. For example, to determine if users know the July inventory file name, the analyst compares FK$_{21}$ against CK$_{11}$. Most analysts, using their judgment, would say FK$_{21}$ is covered by CK$_{11}$—i.e., supported. All analyst judgments must be documented as part of the TMM analysis. Documentation allows analysts to revisit judgments that might be questionable (and sometimes wrong).

<table>
<thead>
<tr>
<th>Example User Class Profile</th>
<th>Example User Knowledge Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Domain: ACCOUNTING</td>
<td>FK$_{21}$: ACCOUNTING for July INVENTORY is stored in the ACCT.7.INV.DATA file.</td>
</tr>
<tr>
<td>Task: RECONCILE the JULY INVENTORY and ACCOUNTS</td>
<td>FK$_{22}$: DISKS are SINGLE or DOUBLE SIDED.</td>
</tr>
<tr>
<td>Users: Equal Sex distribution</td>
<td>FK$_{23}$: There are 5 standard Motif$^{\text{TM}}$ SELECTION MODELS.</td>
</tr>
<tr>
<td>Average Age: 32 years old.</td>
<td>CK$_{21}$: DOUBLE-ENTRY BOOKKEEPING SYSTEMS</td>
</tr>
<tr>
<td>Experience: Minimum training in direct manipulation interaction style; advanced (college B.S. level) training in ACCOUNTING</td>
<td>CK$_{22}$: How to use LINE-EDITOR COMMAND STRUCTURES</td>
</tr>
<tr>
<td>Knowledge:</td>
<td>PK$_{21}$: INVENTORY is CATALOGUED and then ENTERED INTO THE SYSTEM.</td>
</tr>
<tr>
<td>FK$_{11}$: Names of all ACCOUNT DATA FILES</td>
<td>PK$_{22}$: FILES are MODIFIED through EDITING.</td>
</tr>
<tr>
<td>CK$_{11}$: File naming conventions for INVENTORY DATA FILES</td>
<td>PK$_{23}$: DOUBLE-CLICK on a &lt;FILENAME&gt; to LAUNCH Application.</td>
</tr>
<tr>
<td>CK$_{12}$: Direct manipulation interaction style</td>
<td>and so on...</td>
</tr>
<tr>
<td>CK$_{13}$: DOUBLE-ENTRY ACCOUNTING</td>
<td>and so on...</td>
</tr>
<tr>
<td>FK$_{11}$: How to reconcile QUARTILE INVENTORY REPORTS.</td>
<td>and so on...</td>
</tr>
</tbody>
</table>

Chapter III: The Task Mapping Model — 73 —
III.3 OTHER TASK DESCRIPTION ISSUES IN THE TMM

This section is devoted to discussion of several research issues concerning task description using the TMM.

III.3.1 DOMAIN SKIPPING IN TASK PERFORMANCE

Earlier, task performance was characterized as requiring users to map each task through a series of domains or abstraction levels. There are some cases, however, where a user might map a task directly from the problem domain to, say, the articulation domain. In other words, while performing tasks, users reconceptualize items from one domain to another non-adjacent domain, omitting the intervening mappings. We refer to this as domain skipping. This section discusses domain skipping in reference to two different user classes: novice and expert users.

**Novice User Domain Skipping**

E.g. Consider a user that has been told how to perform a task (format a series of disks) at the articulation level (i.e., what to do and type) without being given much (or any) understanding of the task at the computer semantic or syntactic levels. The user knows only to insert the disk into the machine and press a pre-defined sequence of keys. For this user there is no understanding (or need) of the intervening domains and mappings for this task.

In this case, the novice user reconceptualizes an item from the problem domain to the computer syntactic or articulation domains and skips intervening domains. The user knows what to do, but there is no understanding of the underlying process of the task apart from the articulatory actions. Also, with repeated rehearsal this kind of novice behavior becomes rote performance.

While TMM notation allows analysts to describe and analyze novice user domain skipping, it is advisable that, when this phenomenon is observed, designers take more serious measures to inform users about their actions. Our example is described using TMM in Figures III.6 and III.7, both with and without domain skipping. This comparison shows there is much that can be overlooked by users.
Comparing these two descriptions, it can be seen that the second description implies a much richer knowledge base than the first. The first description paints a picture of users unaware of interaction styles and underlying system actions.
There are several alternatives that designers can employ in order to assist novice users in understanding underlying task mappings. For example, interfaces can expressly output stages of interaction/processing or artificially pace the user in the task. Also, designers can consider the possibility of redesigning the interaction, task, and interface design such that there are fewer (or no) mappings for the user to consider. An example of reducing user mappings is seen in the Macintosh operating system. The interface is modeled after a desktop metaphor that is more cognitively direct (requiring fewer user task mappings) than a command-line interaction style.

**Expert User Domain Skipping**

Domain skipping by expert users is an altogether different situation representing expert automated behavior moreso than rote performance (as with novice user domain skipping). In this case, expert users, who have the knowledge required to make the mappings, may skip across domains without consciously making the corresponding mappings.

Consider, for example, the user class consisting of Unix system programmers—most are expert users of vi (visual editor). These expert users can initiate an **EDITOR-INSERT COMMAND** without consciously thinking about necessary mappings—performance without thinking. However, long ago these users had to learn the necessary mappings for this task, and through continual use of this task and interface have automated that behavior—also referred to as operationalization as in Bodker (1989).

As another example, consider the fast-paced Nintendo™ video game Super Mario Bros. 3™. Children all over America play this game daily and have become experts in it. Yet, their thoughts about what action to do next has become automated. The ability to jump the virtual player over a **KILLER CONGO GREY** has long since been pushed into automated behavior.

### III.3.2 TASK CONTRACTIONS

Another issue addressed during this research is associated with task contractions. In the English language contractions exist to speed up communication—*do not* becomes *don't*, *we will* becomes *we'll*, and so on. Contractions can extend to task performance for the same reasons—i.e., tasks can have contractions. A task contraction is an alternative task method that folds several user actions/commands into a single action/command. For example, task A composed of actions

---

3Nintendo and Nintendo Entertainment Systems are Trademarks of Nintendo of America, Inc. Super Mario Bros. 3 is a Trademark and Copyright of Nintendo of America, Inc.
XYZ might have a task contraction $A'$ made up of actions $WZ$—where $W$ is a contraction of the $X$ and $Y$ user actions. As an example, consider moving an icon within a direct manipulation interface. MOVING is actually a contraction of the tasks SELECT and MOVE, with the SELECT task contracted into the MOVE task.

Consider, as another example, the task DELETE FILE(S) within the Macintosh OS. Roughly speaking, the task is broken down into several sub-tasks: \textsc{select the file(s)}, \textsc{move the file(s) to the trashcan}, \textsc{execute the delete operation} (i.e., drop the icons into the trash). This task is illustrated with UAN in Figure III.8.

<table>
<thead>
<tr>
<th>USER ACTIONS</th>
<th>INTERFACE FEEDBACK</th>
<th>INTERFACE STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{select file(s)}</td>
<td>HIGHLIGHT ALL SELECTED FILES</td>
<td>SELECTED = ALL FILES SELECTED</td>
</tr>
<tr>
<td>\textsc{move the file}</td>
<td>OUTLINE(SELECTED FILES) &gt;</td>
<td></td>
</tr>
<tr>
<td>\textsc{[file_icon]}' ignore MV</td>
<td>OUTLINE(SELECTED FILES) &gt;</td>
<td></td>
</tr>
<tr>
<td>\textsc{[x,y]}' ignore MV</td>
<td>OUTLINE(SELECTED FILES) &gt;</td>
<td></td>
</tr>
<tr>
<td>\textsc{[trashcan_icon]}</td>
<td>TRASHCAN_ICON!</td>
<td></td>
</tr>
<tr>
<td>\textsc{execute the delete}</td>
<td>ERASE (SELECTED FILES ICONS)</td>
<td></td>
</tr>
<tr>
<td>\textsc{mark selected files for deletion}</td>
<td>ERASE (OUTLINE(SELECTED FILES))</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE III.8. TASK: DELETE A FILE V.1**

This task description illustrates how to delete one or more files. Figure III.9 describes an alternative contracted version of this task in Figure III.8, i.e., the FILE SELECTION sub-task is contracted into the MOVE sub-task.

<table>
<thead>
<tr>
<th>USER ACTIONS</th>
<th>INTERFACE FEEDBACK</th>
<th>INTERFACE STATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textsc{move the file}</td>
<td>FILE_ICON!</td>
<td>SELECTED = FILE_ICON</td>
</tr>
<tr>
<td>\textsc{[file_icon]}' ignore MV</td>
<td>OUTLINE(FILE_ICON) &gt;</td>
<td></td>
</tr>
<tr>
<td>\textsc{[x,y]}' ignore MV</td>
<td>OUTLINE(FILE_ICON) &gt;</td>
<td></td>
</tr>
<tr>
<td>\textsc{[trashcan_icon]}</td>
<td>TRASHCAN_ICON!</td>
<td></td>
</tr>
<tr>
<td>\textsc{execute the delete}</td>
<td>ERASE (FILE_ICON)</td>
<td></td>
</tr>
<tr>
<td>\textsc{mark selected files for deletion}</td>
<td>ERASE (OUTLINE(FILE_ICON))</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE III.9. TASK: DELETE A FILE V.2 (WITH TASK CONTRACTION)**

---

\textsuperscript{4}This task is not defined here in order to reduce space and increase description simplicity. However, it is effectively the task \textsc{select file_icon} using a selection model (e.g., single selection, browse selection, multiple selection, range selection, and discontinuous selection—see (OSF, 1991, section 3.1)).
These two UAN task descriptions, from the TMM articulation domain, illustrate alternative methods for the task DELETE A FILE. Version 2 is a task contraction of Version 1—i.e., Version 2 performs the same task (as Version 1) with by folding the select sub-task into the move sub-task.

For TMM analyses purposes, task contractions represent different methods of task completion, and as such, should be described as alternative task methods. Interface designers that build in undocumented contractions can cause problems for task analysis and description; hence, contractions must be recognizable by analysts so as to insure proper analysis for user needs.

III.2.3 SUPER-TASKS

Another interesting point about the two previous task descriptions is that Version 1 is a super-task of Version 2. In other words, Version 2 is a special case (DELETE ONE FILE) encompassed within Version 1 (DELETE FILE(S)). A super-task is a generalized version of a task that may include several different task methods.

Designers who incorporate super-tasks (and task contractions) in interface designs must provide consistent support of user actions and commands (e.g., using a common metaphor). For example, users would find it confusing if methods to delete one file were radically different from deleting several files.

III.4 TMM METHODOLOGY

This section discusses methodologies associated with each phase of the TMM life cycle (refer back to Figure III.2). Each phase is outlined and includes a description of its activities. Recalling that many of the currently available analytical evaluation techniques are not cost-effective and not used, the TMM methodology strives to be practical, economical, and easy-to-use within the context of iterative user interface development.

In order to illustrate the TMM methodology, this discussion is coupled with a small running example:

USER: ACCOUNTANT
SITUATION: An accountant, while using a common WORD PROCESSING SYSTEM, has called up a DOCUMENT for reference only. After reviewing the DOCUMENT extensively, the accountant tries to quit the WORD PROCESSING SYSTEM. Before the system terminates the user's session, it responds with a "SAVE DOCUMENT CHANGES? [Y/N] : " dialogue.

TASK: EXIT DOCUMENT PREPARATION SYSTEM.

The task description and analyses are curtailed due to space and narration concerns. A larger and more complete example illustrating the TMM methodology is contained in Mayo (1993a).

III.4.1 TASK DESCRIPTION

The first phase of the TMM analysis process is generating a task definition and description—see Figure III.10. The TMM provides a framework for describing tasks which helps to facilitate analyses of user tasks for knowledge requirements. The description framework is discussed in a previous section (refer to Section III.2: TMM Task Description Framework).

![Diagram of Task Description Process within TMM](image)

FIGURE III.10. TASK DESCRIPTION PROCESS WITHIN TMM

The process for generating TMM task description is much like any other task description methodology. In our case, candidate tasks are selected for analysis by examining the users'
overall task structure along with formative evaluation results.\textsuperscript{5} The candidate task is decomposed into the various TMM abstract domains. Connections, mappings, are made between associated task abstraction elements, i.e., links are made between related domain items across domain boundaries. Each mapping is then analyzed for user knowledge requirements. I.e., analysts list user knowledge required to reconceptualize one domain item to another.

The task descriptions generated in this phase are used throughout the subsequent phases of the TMM life cycle. Therefore, it is important that task descriptions be complete because thoroughness characterizes the quality of the TMM analyses. It's obvious that a fully defined task yields better results than a poorly defined task. To this end, several of the steps within this phase are iterated, and the process continues until a comfortable level of description is attained—i.e., until analysts are convinced that further decomposition and description is not cost-effective.

The first steps in creating the task description is to decompose the task into the TMM domains (using specific domain grammars and vocabularies) and adding corresponding mappings. For the example task, \texttt{EXIT DOCUMENT PREPARATION SYSTEM}, a partial decomposition is shown in Figure III.11.

![Diagram of task decomposition and description]

\textbf{FIGURE III.11. TASK DESCRIPTION FOR TASK: EXIT DOCUMENT PREPARATION SYSTEM}

Following the task decomposition and description, the analyst examines each of the numbered mappings to determine the user knowledge requirements. At this point, the analyst must rely on

\textsuperscript{5}Formative evaluation results identify and isolate problematic user tasks, i.e., tasks which are identified as having usability problems based on user performance. Of course, if we are doing initial analysis/design there will be no formative evaluation results to draw upon.
empirical evidence (i.e., user testing, user profiles) and experienced (i.e., personal intuition). A small sub-set of the knowledge requirements for the example are listed in Table III.6.

**TABLE III.6. KNOWLEDGE REQUIREMENTS FOR TASK: EXIT DOCUMENT PREPARATION SYSTEM**

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Type</th>
<th>Knowledge Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FK</td>
<td>Termination will stop the current documentation preparation system</td>
</tr>
<tr>
<td></td>
<td>FK</td>
<td>Documentation preparation system are commonly called word processing and desk-top publishing systems</td>
</tr>
<tr>
<td>2</td>
<td>FK</td>
<td>Noun-Verb interaction style. However, for this task there is no Noun (object). So select the function and execute it.</td>
</tr>
<tr>
<td>3</td>
<td>FK</td>
<td>The terminate function is &quot;QUIT&quot;.</td>
</tr>
<tr>
<td></td>
<td>FK</td>
<td>Hitting return will execute any typed input.</td>
</tr>
<tr>
<td>4</td>
<td>FK</td>
<td>The set of changes made to the original document.</td>
</tr>
</tbody>
</table>

**III.4.2 KNOWLEDGE ANALYSIS**

The next phase of the TMM analysis life cycle gleanes and categorizes user knowledge requirements from task descriptions generated in phase one—see Figure III.12. Each TMM mapping, within the task description, identifies when users reconceptualize domain items; further, each mapping has a set of knowledge requirements identifying what the users must know to perform the reconceptualization.
FIGURE III.12. KNOWLEDGE ANALYSIS WITHIN TMM

The union of all the sets of knowledge requirements (from all of the mappings) represents the sum-total knowledge users need to accomplish the described task. Each knowledge requirement from the total set is categorized as supported or unsupported. Supported knowledge requirements represent knowledge that either the user possesses (i.e., it is indicated within the user class profile) or it is present within the interface (e.g., through visual cues). Conversely, if the knowledge is not supported by either of these two cases then it is categorized as unsupported. Unsupported knowledge requirements represent what users do not know and cannot divine by using the interface. It is clear that unsupported knowledge requirements represent critical impasses, and when faced with unsupported knowledge requirements, users must rely on exploratory learning or user manuals—both of which may be problematic.

Table III.17 shows the results of analyzing the knowledge requirements found for our example. Each knowledge requirement is categorized (with a ‘/’ as: supported by user class profile (S1), supported by interface (S2), or unsupported (US). Even in this small example we see unsupported knowledge requirements. Mapping 4 identifies the users' need to know the changes to the document in order to determine if the changes should be saved. Designers cannot expect users to
remember all changes, and the interface currently does not display a history of changes; therefore, mapping 4 is unsupported.

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Type</th>
<th>Knowledge Requirement</th>
<th>S1</th>
<th>S2</th>
<th>US</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FK</td>
<td>Termination will stop the current documentation preparation system</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>FK</td>
<td>Documentation preparation system are commonly called word processing and desk-top publishing systems</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>PK</td>
<td>Noun-Verb interaction style. However, for this task there is no Noun (object). So the user must select only the function and execute it.</td>
<td>√</td>
<td>√</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FK</td>
<td>The terminate function is “Q”.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>FK</td>
<td>Hitting return will execute any typed input.</td>
<td></td>
<td></td>
<td>√</td>
</tr>
<tr>
<td>4</td>
<td>CK</td>
<td>The set of changes made to the original document.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### III.4.3 UIDR SYNTHESIS

In the final phase of the TMM analysis life cycle, unsupported knowledge requirements from phase two are translated into new user interface design requirements (UIDRs)—see Figure III.13.
In the most simple cases, knowledge requirements are recast into UIDs based on the type of knowledge and using templates such as:

**FACTUAL KNOWLEDGE UIDR**

UIDR: COMMUNICATE <FACT> TO USER

(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

**CONCEPTUAL KNOWLEDGE UIDR**

UIDR: PROVIDE UNDERSTANDING OF <CONCEPT> TO USER

(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

**PROCEDURAL KNOWLEDGE UIDR**

UIDR: IDENTIFY POSSIBLE PATHS OF EXECUTION TO USER

(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

The following example for a fetal-monitoring system shows how a template is used to translate an unsupported factual knowledge requirement (**A HEART BEAT OF 15 OR LESS IMPLIES DISTRESS**) into a new UIDR (**COMMUNICATE <A HEART BEAT OF 15 OR LESS IMPLIES DISTRESS> TO THE USER (AT THE APPROPRIATE TIME DURING TASK PERFORMANCE)**)—see Figure III.14.
Unsupported Knowledge Requirement:
A HEART RATE OF ≤15 IMPLIES DISTRESS
AS DISPLAYED ON A FETAL-MONITOR SYSTEM INTERFACE

User Interface Design Requirement Template:
COMMUNICATE <UNSUPPORTED KNOWLEDGE REQUIREMENT> TO THE USER
(AT THE APPROPRIATE TIME DURING TASK PERFORMANCE).

User Interface Design Requirement:
COMMUNICATE <A HEART RATE OF ≤15 IMPLIES DISTRESS> TO THE USER
(AT THE APPROPRIATE TIME DURING TASK PERFORMANCE).

FIGURE III.14. EXAMPLE TRANSLATION OF AN UNSUPPORTED KNOWLEDGE REQUIREMENT INTO A NEW USER INTERFACE DESIGN REQUIREMENT (UIDR)

For our previous example task, EXIT DOCUMENT PREPARATION SYSTEM, we have described the task and analyzed it for unsupported knowledge needs. Users, in order to decide whether or not to save changes, must know what changes have occurred—mapping 4. Using the factual knowledge UIDR template, this knowledge requirement translates into

UIDR: COMMUNICATE <THE SET OF CHANGES MADE TO THE ORIGINAL DOCUMENT.> TO USER (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

At this point, the TMM has accomplished its goal, and the new UIDR is passed along to interface designers who will redesign and effect changes to the interface design. TMM is not involved, currently, in specifying a design solution—the solution is still in the bailiwick of the designer.6

Of course, it is not always straightforward to generate user interface design requirements. The analyst should always consider the context of the users' task structure and current interface design(s) when formulating new user interface design requirements.

---

6 As a side note, this UIDR can be addressed with techniques discussed in Hill, et al. (1992). This work presents and evaluations systems that incorporate file read and edit accesses into histories associated with the file.
III.5 CONCLUSIONS

This chapter discusses some of the rationale used while developing the TMM. This chapter does not capture all of the discussions and issues addressed during the entirety of the research, but it does focuses on some of the primary decisions we made.

To review, the TMM uses abstraction levels (domains) for task description and analysis of user knowledge requirements. We argue for closed-loop analysis of user tasks, supporting this argument with Norman’s Theory of Action. Using the TMM is a three phase process, and the steps of the analyses are outlined and reviewed as well as provided within the context of user interface development. The model outlined in this chapter is also discussed in (Mayo, 1993a; Mayo, 1993c; Mayo & Hartson, 1993).
Science is nothing but perception
—Plato

Experience is a hard teacher
because she gives the test first,
the lesson afterwards.
—Vernon Saunders Law

IV COMPARISON OF TMM & DGL

The task mapping model (TMM) uses systems analysis and formative evaluation results with task description and analysis to synthesize user interface design requirements (UIDRs). There are many aspects of TMM that can be evaluated to justify TMM’s usefulness. This research focused on comparing the TMM method against using design guidelines (DGL) for synthesizing UIDRs. By examining and comparing UIDRs against a set of seven quality-rating criteria, this approach characterizes the analyses of both TMM and DGL methods.

In order to compare TMM and DGL, a three phase experiment was performed—see Figure IV.1. This evaluation experiment was performed within a cycle of

- INTERFACE DESIGN ANALYSIS (Phase One)
  A series of user interface designs and benchmark tasks were studied to identify usability problems.

- SYNTHESIS OF UIDRs BASED ON USABILITY PROBLEMS (Phase Two)
  Usability problems (from phase one) were analyzed using two methods (TMM and DGL) to produce new UIDRs.
• EVALUATION OF UIDR QUALITY (Phase Three)
  E evaluators judged quality of synthesized UIDRs.

This chapter discusses the evaluation experiment and reports results in several sections: Experiment Objectives, Experimental Design Rationale, Experiment Pre-Test, the three Experiment Phases, and a summary.

![Diagram](image)

FIGURE IV.1. THREE PHASE EXPERIMENT TO EVALUATE THE TMM

IV.1 EXPERIMENT OBJECTIVES

The TMM experiment is designed to compare the TMM against DGL to characterize their quality of analyses and synthesized UIDRs. Phases two and three generated data used for the comparisons. Specifically, in phase two data were collected on analysts' reactions to each method's learnability and usability. Phase three data consists of evaluator comparisons of UIDRs.
against a set of seven quality criteria. These two sets of data help to address the following questions:

1. How do analysts characterize TMM and DGL ...
   ... in terms of defining and controlling the analysis process?
   ... in terms of using analysis notations?
   ... in terms of learnability?

2. How does the quality of UIDs synthesized using TMM compare with UIDs synthesized with DGL?
   ... in terms of how well they address the identified usability problems?
   ... in terms of how user-centered the UIDs are?
   ... in terms of introducing new usability problems inherent to the UIDs?

Answers to these questions support the validity and importance of this research and provide a frame of reference for defining future research activities with TMM.

**IV.2 EXPERIMENTAL DESIGN RATIONALE**

This section identifies the arguments and decisions that form design rationale for the TMM evaluation experiment.

**IV.2.1 WHY NOT COMPARE TMM WITH GOMS, CLG, OR ANOTHER HCI MODELING TECHNIQUE?**

The TMM is not a performance prediction model, nor is it an interface modeling technique. TMM is a task-/user-centered analysis tool for synthesizing new UIDs. Other HCI modeling techniques do not attempt to synthesize new UIDs, so the number of appropriate techniques for comparison was small. The extended comparison provided in Chapter II outlines many differences between TMM and other HCI modeling techniques. Using design guidelines as a synthesis technique is possible because design guidelines are phrased in the language of design; thus design guidelines can be used to specify new design requirements.
IV.2.2 Analyst Team Size

In early experience with the TMM, user interface designers felt that TMM analysis and synthesis should be performed by teams of analysts (Mayo, 1993a). Designers were concerned about overlooking important task information, or getting mentally stuck in a particular method or solution. Designers indicated the need for several different views to serve as catalysts for analysis, e.g., a user perspective, a designer perspective, a systems developer perspective. While these experiences showed that TMM should be used by teams, it is still unclear what size team would be best for TMM analyses. For the TMM experiment, analysts worked in one of two groups (TMM or DGL) divided into teams of two—the minimum size that satisfies the need to perform analyses within teams. Using teams also reflects how design is performed in the computer industry as a team-oriented process—design is very rarely an individual effort. However, identifying the ideal analysis team size and examining team dynamics are factors outside the scope of this research and considered areas for future research.

Ideally, the TMM experiment would have had many teams within each analysis group (both TMM and DGL), thus increasing the power of the experiment. However, the experiment was performed with ten teams of two analysts—totaling twenty participants. Twenty subjects fit experimental limitations based on cost, time, and availability of qualified subjects.

IV.3 Pre-Test

An extensive pre-test was performed addressing several underlying issues that affect experimental design. The pre-test followed the overall experiment focusing on issues related to training analysis methods, interface design prototypes and task selection, and handling of experimental data.

IV.3.1 Participant Training in Analysis Methods

There are two analysis/synthesis methods employed in this experiment—TMM and DGL. Participants in phase two are trained in one of these methods based on their role within the experiment. In order to reduce training variability, the training materials were reviewed by experienced HCI specialists and educators checking accuracy and consistent depth of coverage.
The pre-test localized ambiguities and difficulties in training and training materials. The pre-test also identified the required amount of training for each of the methods.

### IV.3.2 Interface Design Prototypes

Selection of user tasks and development of user interface design prototypes can have a profound effect on studies of user performance (Held & Biers, 1992). UIDR synthesis methods may be indistinguishable if user tasks and prototypes are either too elementary or too intricate. During the pre-test, tasks and interface designs were evaluated for appropriateness, i.e., whether the interface design can be analyzed in a given amount of time, whether the interface designs have rich sets of usability problems, and whether the interface designs represent a broad range of interaction—purposefully excluding command line interfaces.

Interface designs were gathered from external sources to avoid any bias that could have been incorporated into an in-house software system. Also, an initial set of fifteen interface designs were selected of which six were used in the pre-test. Each interface had a set of $N$ usability problems, where $N > 4$. Pre-test results identified four interface designs and four usability problems per interface design that met the above mentioned appropriateness criteria. Much of this work overlapped with phase one of the TMM evaluation experiment.

Interface design prototypes selected for this study are publicly available HyperCard shareware stacks. These interfaces design prototypes were gathered from the InterNET using anonymous FTP (file transfer protocol) sites maintained at the University of Michigan (specifically, mac.archive.umich.edu with IP address 141.211.165.41).

### IV.4 Phase 1: Determine Usability Problems

#### IV.4.1 Method

The first phase of the experiment employs heuristic evaluation on several different user interface design prototypes to find usability problems with regards to several user tasks—see Figure IV.2. This phase required user interface design prototypes, user tasks, and evaluators. Results of this phase were a collection of usability problems associated with each user interface design.
Usability problems, associated with each interface, were generated using heuristic evaluation (Molich & Nielsen, 1990; Nielsen & Molich, 1990; Nielsen, 1992). Two independent sources analyzed the interface design prototypes, and results were compared. Four usability problems were selected from analysis results.

The usability problems required further qualification. In other words, the identified usability problems were checked (1) to verify they were indeed usability problems, and (2) to be sure they were severe enough to be detectable by users and not just by expert interface specialists. In order to check usability problems against these two criteria, usability problems were correlated against a very limited round of user testing. This testing served to qualify usability problems as genuine hindrances—and was not intended to generate or identify new usability problems. Nevertheless, a few new usability problems were identified and later incorporated into the usability problem list. Also, the usability problems were examined by an experienced HCI researcher/practitioner.

Tables IV.1-IV.4 list the interface design prototypes, user tasks, and usability problems.
### Table IV.1. Interface Design Prototype—IMTS II 4.01 (Thomas, 1992)

<table>
<thead>
<tr>
<th>Overview</th>
<th>The IMTS II software system catalogues music media such as compact discs, albums, and tapes. Each entry into the system represents one collection of music—where a collection is defined as a single purchasable entity (e.g., CD single, CD, CD box set). The system stores information pertaining to the artist, manufacturer, songs, and media format.</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Task (1).</td>
<td>“Sort contents of Media Database by Artist.”</td>
</tr>
<tr>
<td>Usability Problem (1).</td>
<td>“User unable to perform task within usability performance specification ... difficulty selecting buttons ... USER ERROR: user selected Sort ICON (Not the button Artist)”</td>
</tr>
<tr>
<td>User Task (2).</td>
<td>“Enter Don McLean’s Single American Pie into system”</td>
</tr>
<tr>
<td>Usability Problem (2).</td>
<td>“User unable to perform task within usability performance specification ... confusion in terminology ... (multiple use of term Single)”</td>
</tr>
<tr>
<td>User Task (3).</td>
<td>“Enter Rod Stewart’s Storyteller Triple CD Box Set into system”</td>
</tr>
<tr>
<td>Usability Problem (3).</td>
<td>“User unable to perform task within usability performance specification ... difficulty adding new media ... USER ERROR: user entered objects before (New Media) action”</td>
</tr>
<tr>
<td>User Task (4).</td>
<td>“Exploratory usage”</td>
</tr>
<tr>
<td>Usability Problem (4).</td>
<td>“User had difficulty with Button objects: Hide and Show Buttons and palette buttons for Compact, Sort (Album, Artist, Label), About, Media (New, Delete), and Search ... inconsistent button usage.”</td>
</tr>
</tbody>
</table>

### Table IV.2. Interface Design Prototype—Periodic Table (Weber, 1992)

<table>
<thead>
<tr>
<th>Overview</th>
<th>This system provides users the ability to examine information on the periodic table of elements as well as each individual element.</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Task (1).</td>
<td>“Identify (list) all Noble Gasses”</td>
</tr>
<tr>
<td>Usability Problem (1).</td>
<td>“User unable to perform task within usability performance specification ... User missed highlight (too brief) of Noble Gasses after following correct path through General Inf. Button”</td>
</tr>
<tr>
<td>User Task (2).</td>
<td>“Identify (list) all Lanthanides”</td>
</tr>
<tr>
<td>Usability Problem (2).</td>
<td>“User unable to perform task within usability performance specification ... USER ERROR: User Identified Elements %58-#71 as Lanthanides (thus missing Element #57)”</td>
</tr>
<tr>
<td>User Task (3).</td>
<td>“Change Boiling Point and Melting Point (of any Element) to degrees Kelvin”</td>
</tr>
<tr>
<td>Usability Problem (3).</td>
<td>“User unable to perform task within usability performance specification ... difficulty getting help on changing units ... difficulty selecting buttons”</td>
</tr>
<tr>
<td>User Task (4).</td>
<td>“List all elements that exist as a GAS at 26 degrees C”</td>
</tr>
<tr>
<td>Usability Problem (4).</td>
<td>“User unable to perform task within usability performance specification of 3 minutes”</td>
</tr>
</tbody>
</table>
**TABLE IV.3. INTERFACE DESIGN PROTOTYPE—TITUS ANDRONICUS (ZIMMERMAN, 1992)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview.</strong> This software system presents Shakespeare's play Titus Andronicus. The system provides the users the ability to examine the play scene by scene, search through the play, maintain scene notes, and consult a glossary.</td>
</tr>
<tr>
<td><strong>User Task (1).</strong> &quot;Using Glossary to find granercy&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (1).</strong> &quot;User unable to perform task within usability performance specification of 30 seconds . . . user could not even get to glossary&quot;</td>
</tr>
<tr>
<td><strong>User Task (2).</strong> &quot;Answer the question: What news did AEmilius bring in his first appearance?&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (2).</strong> &quot;User could not locate AEmilius within the usability performance specification of 1 minute (e.g., using the Find operation)&quot;</td>
</tr>
<tr>
<td><strong>User Task (3).</strong> &quot;Free exploration of Play text&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (3).</strong> &quot;User indicated that the function of the buttons associated with traversing through the play text was not clear&quot;</td>
</tr>
<tr>
<td><strong>User Task (4).</strong> &quot;Using the Find operation (via the eyeball ICON)&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (4).</strong> &quot;User experienced difficulty with functional ambiguity associated with the Find Button's operation&quot;</td>
</tr>
</tbody>
</table>

**TABLE IV.4. INTERFACE DESIGN PROTOTYPE—CELTIC MUSEUM (NEWTON, 1991)**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overview.</strong> This software system represents Celtic information and traditions (e.g., art, religion, history, language) within the context of a museum metaphor. The system employs a navigational model based on museum hallways and hypertext links (referred to as transportation within the system). The user is left to free exploration and learning—much like within a real museum.</td>
</tr>
<tr>
<td><strong>User Task (1).</strong> &quot;Use of transporter: E.g., task: Follow a hyper link and get back (i.e., return)&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (1).</strong> &quot;User got lost in museum . . . e.g., within Hall of Religion: Druids card, the user followed the Ogam hyper link and lost way/context/place within museum&quot;</td>
</tr>
<tr>
<td><strong>User Task (2).</strong> &quot;Trace bibliographic reference and get back (i.e., return)&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (2).</strong> &quot;User could not trace a bibliographic reference and return—e.g., within Marriage in Ancient Ireland card, the user could not trace the reference [56] and reliability get back to the marriage card&quot;</td>
</tr>
<tr>
<td><strong>User Task (3).</strong> &quot;Window and card scrolling . . . task: exploratory learning&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (3).</strong> &quot;User confused text-field scrolling with card scrolling—i.e., cards with more information on a topic allow/feature scrolling, while text-fields within the cards also scrolled—thus confusing users&quot;</td>
</tr>
<tr>
<td><strong>User Task (4).</strong> &quot;Entering and exiting the hallways via the Hallway view&quot;</td>
</tr>
<tr>
<td><strong>Usability Problem (4).</strong> &quot;User confused with arrows in Hallway view&quot;</td>
</tr>
</tbody>
</table>
IV.5 PHASE 2: SYNTHESIZE NEW UIDRS

IV.5.1 METHOD

The second phase of the experiment involved two distinct analysis groups—one group used TMM and another group used DGL to synthesize new UIDRs. Each analysis group was further divided into five teams of two analysts. Each team examined and analyzed four usability problems for each of the four interface designs (from phase one) to produce sixteen sets of new UIDRs. After all analyses were finished, each analyst completed a survey about the method they used for synthesizing UIDRs. The survey contains questions about using the method and its notations. Analysts' survey responses help characterize their experiences with analysis methods, and allows comparisons between the two methods. Figure IV.3 outlines phase 2.

![Figure IV.3: Phase 2 of the TMM Evaluation Experiment (Excerpt from Figure IV.1)](image)

Participants. Twenty participants served as human-computer user interface analysts. Criteria for each analyst were (at a minimum): a B.S. in computer science (or related field) and a graduate level introduction to issues and practices of human-computer interface analysis and design. These criteria served as a baseline helping to reduce variability through homogenous participants.

Twenty analysts were randomly divided into two equal sized analysis groups—one for TMM analysis and another for DGL analysis. These two analysis groups were then randomly divided again into five equally sized teams. Therefore, the TMM analysis group consisted of five teams of two analysts, and the DGL analysis group also had five teams of two analysts. Some practical
consideration was given to team membership for one team—this was to avoid detrimental team
dynamics.

*Apparatus.* This phase required a Macintosh II computer equipped with prototyping software
for each of the ten analysis teams. This hardware was required to provide access to the user
interface design prototype during analysis sessions.

During training, TMM analysts were provided with the TMM Analyst's Guide (Mayo, 1993a), the
TMM tutorial (Mayo, 1993c), a copy of the training transparencies, and all necessary TMM forms.
Each DGL analyst was provided with a fully cross-referenced paper-copy of Smith and Mosier's
design guideline set. Also, the DGL teams were given an automated tool (with training and trial
time) to search Smith and Mosier's collection of design guidelines (Iannella, 1991; Smith &
Mosier, 1986). Training was thorough, however, there was no way to compare the levels of
mastery between the TMM and DGL analysts—it would be like comparing apples to oranges. It
is impossible to relate a 70% mastery of TMM to a 70% mastery of DGL. If any bias existed, it was
to favor DGL—all analysts had previous experience with DGL, while none had experience with
TMM.

*Procedure.* This experimental phase was performed in the following manner:

- Experimenter provided a 120 minute lecture on appropriate analysis and reporting
techniques.

- Two days later, each analysis team analyzed each user interface design. Teams were
  sequestered from each other and worked concurrently in the following manner:

  - Analysis teams were given copies of actual interface prototype, and a list of four
    usability problems identified for this interface design (identified in phase one).

  - Analysis teams evaluated/analyzed the user interface design with respect to each of
    the four usability problems. (Both interface design treatment order and usability
    problem treatment order were randomized.)

  - Each analysis team produced a set of UIDRs for each usability problem based on
    their analysis method. UIDR sets are reported on special forms supplied by the
    experimenter (see Appendix C: USER INTERFACE DESIGN REQUIREMENT
    REPORT FORM).
• After all analyses were performed, analysts were individually surveyed about their experiences with the analysis method and notation. (see Appendix D: USER INTERFACE ANALYST SURVEY)

IV.5.2 RESULTS AND DISCUSSION

Data generated in this experimental phase are (1) new UIDRs and (2) analyst survey responses. UIDRs are reproduced in Appendix G: UIDRS. Data and analyses of analysts' reactions are reproduced in Appendix F: PHASE TWO DATA. Results are discussed in this section.

**Analyst Survey Results**

Each analyst responded to a survey about her/his particular analysis method—either TMM or DGL. Each survey question is composed of several satisfaction scales by which analysts can report reactions to the analysis method they used. An average of these scales across each analyst was also calculated to provide a characterization of the overall question. The scales are 7-point Likert scales ranging from -3 to +3 and anchored by satisfaction criteria taken from QUIS (Chin, Diehl, & Norman, 1988). A copy of each survey is contained within Appendix D: ANALYST SURVEYS.

The first concern with respect to data analysis for this phase of the experiment was whether to use parametric or non-parametric analyses on the scale data. To address this concern, a comparison of parametric and non-parametric data analyses was performed, and summary tables can be found in Appendix F. Differences between parametric and non-parametric results were as expected—both produced very similar results, but parametric tests were more sensitive. These results, coupled with the assumption that the scale data is interval, led to our selecting parametric data analyses. Likert scales were examined using ANOVA tests to determine differences between the TMM and DGL analysis methods, and t-tests were used to identify specific differences in individual scales pooled across analysis sub-groups for each analysis method.

**Direct Comparison between TMM and DGL (ANOVA Results)**

Analyses of the scaled responses were the same for all scales—a 2 factor, between-subject, complete hierarchical design. Each scale has twenty responses corresponding to the analysts nested within five teams nested within two analysis methods groups. A hierarchical design was
used to test significance of the main effect (A= Analysis Method) and effects of teams within each analysis method (T/A).

I.e.: for each scaled response the analysis was performed as:

<table>
<thead>
<tr>
<th>Factor</th>
<th>df</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>a-1 ≥ 1</td>
</tr>
<tr>
<td>T/A</td>
<td>a(t-1) = 8</td>
</tr>
<tr>
<td>S/AT</td>
<td>at(n-1) = 10</td>
</tr>
<tr>
<td>Total</td>
<td>atn-1 = 19</td>
</tr>
</tbody>
</table>

Team interactions within analysis methods were not expected for the scaled responses because of group homogeneity. A summary of the ANOVA results is presented in Tables IV.5-IV.8. These tables identify each significant scale variables, scale anchors, and the ANOVA results testing Analysis Method—a full summary of significant and non-significant results are given in Tables A.F.3-A.F.6. Grayed-out effects are not significant. The Teams within Analysis Method effect was significant for only one variable, so it is not reported within Tables IV.5-IV.8. TMM and DGL means are also given for comparison purposes.

**TABLE IV.5. SUMMARY OF SIGNIFICANT RESULTS COMPARING TMM & DGL ANALYSTS’ REACTIONS TO ANALYSIS METHOD**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>Analysis Method F and Pr &gt; F</th>
<th>TMM &amp; DGL Means</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>The analysis METHOD:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>difficult to use</td>
<td>easy to use</td>
<td>6.75 0.0266</td>
<td>TMM: -0.100</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +1.700</td>
</tr>
<tr>
<td>M4</td>
<td>not usable</td>
<td>very usable</td>
<td>7.12 0.0236</td>
<td>TMM: +0.700</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +1.800</td>
</tr>
<tr>
<td>M5</td>
<td>frustrating</td>
<td>satisfying</td>
<td>5.26 0.0447</td>
<td>TMM: -0.200</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +0.900</td>
</tr>
</tbody>
</table>
### TABLE IV.6. SUMMARY OF SIGNIFICANT RESULTS COMPARING TMM & DGL ANALYSTS’ REACTIONS TO ANALYSIS METHOD’S PROCESS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>Analysis Method ( F \text{ and } Pr &gt; F )</th>
<th>TMM &amp; DGL Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>ambiguous</td>
<td>defined</td>
<td>5.25 0.0448</td>
<td>TMM: -0.900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +0.800</td>
</tr>
<tr>
<td>P4</td>
<td>too complex</td>
<td>too simplistic</td>
<td>11.84 0.0063</td>
<td>TMM: -0.900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +0.600</td>
</tr>
</tbody>
</table>

### TABLE IV.7. SUMMARY OF SIGNIFICANT RESULTS COMPARING TMM & DGL ANALYSTS’ REACTIONS TO ANALYSIS METHOD’S NOTATION AND FORMALISMS

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>Analysis Method ( F \text{ and } Pr &gt; F )</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>N2</td>
<td>inadequate power</td>
<td>adequate power</td>
<td>8.39 0.0177</td>
<td>TMM: +0.111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +1.800</td>
</tr>
<tr>
<td>N5</td>
<td>frustrating</td>
<td>satisfying</td>
<td>6.23 0.0341</td>
<td>TMM: -0.111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +0.900</td>
</tr>
<tr>
<td>Mean N</td>
<td>(average of N1 to N7)</td>
<td></td>
<td>5.73 0.0403</td>
<td>TMM: +0.524</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +1.300</td>
</tr>
</tbody>
</table>

### TABLE IV.8. SUMMARY OF SIGNIFICANT RESULTS COMPARING TMM & DGL ANALYSTS’ REACTIONS TO ANALYSIS METHOD’S LEARNABILITY (FOR BOTH PROCESS AND NOTATION)

**Learnability of the METHOD’S PROCESS:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>Analysis Method ( F \text{ and } Pr &gt; F )</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>difficult to learn</td>
<td>easy to learn</td>
<td>12.00 0.0061</td>
<td>TMM: -0.400</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.000</td>
</tr>
<tr>
<td>L2</td>
<td>long time to learn</td>
<td>short time to learn</td>
<td>48.76 0.0001</td>
<td>TMM: -1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
</tr>
<tr>
<td>L3</td>
<td>difficult concepts</td>
<td>easy concepts</td>
<td>36.00 0.0001</td>
<td>TMM: -0.400</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.000</td>
</tr>
<tr>
<td>Mean L1</td>
<td>(average of L1 to L3)</td>
<td></td>
<td>34.48 0.0002</td>
<td>TMM: -0.600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.033</td>
</tr>
</tbody>
</table>

**Learnability of the METHOD’S NOTATION:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>Analysis Method ( F \text{ and } Pr &gt; F )</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4</td>
<td>too complex</td>
<td>too simplistic</td>
<td>15.03 0.0037</td>
<td>TMM: +0.222</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
</tr>
<tr>
<td>L5</td>
<td>unclear starting point</td>
<td>clear starting point</td>
<td>10.20 0.0109</td>
<td>TMM: -0.667</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
</tr>
<tr>
<td>L6</td>
<td>unclear ending point</td>
<td>clear ending point</td>
<td>9.58 0.0128</td>
<td>TMM: +0.333</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.200</td>
</tr>
<tr>
<td>Mean L2</td>
<td>(average of L5 to L6)</td>
<td></td>
<td>12.17 0.0068</td>
<td>TMM: -0.037</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.133</td>
</tr>
</tbody>
</table>
TMM and DGL, comparison results shown in Tables IV.5-IV.8 can be interpreted and restated as follows:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>[M1, M4]</td>
<td>Using the TMM Method is more difficult than using the DGL Method.</td>
</tr>
<tr>
<td>[M5]</td>
<td>Using the DGL Method is more satisfying than using the TMM Method.</td>
</tr>
<tr>
<td>[P1]</td>
<td>The DGL Process is more defined than the TMM Process.</td>
</tr>
<tr>
<td>[P2]</td>
<td>The TMM Process has more steps than the DGL Process.</td>
</tr>
<tr>
<td>[N2,N5,Mean_N]</td>
<td>Overall, the DGL Notations and Formalisms were easier to work with than the TMM Notations and Formalisms.</td>
</tr>
<tr>
<td>[L1-L3,Mean_L1]</td>
<td>Overall, the Learnability of the DGL Method was easier, shorter, and less conceptually difficulty than the TMM Method.</td>
</tr>
<tr>
<td>[L4-L6,Mean_L2]</td>
<td>Overall, the Learnability of the DGL Notation and Formalisms was easier, shorter, and less conceptually difficulty than the TMM Notation and Formalisms.</td>
</tr>
</tbody>
</table>

These results support that analysts found DGL easier to learn and use than TMM (cf. Tables IV.5-IV.8, variables M1, Mean_N, L1-L6, Mean_L1, Mean_L2). These results are not surprising because TMM has richer task description and knowledge analysis methodologies, and due to this richness it is more complicated to use (as compared to DGL) in terms of method, notation, and learnability.

There was only one case, M7, in which group differences occurred. On further inspection of the data, TMM analysis team T05 differed from the rest. Subjective responses from T05 indicated they experienced difficulty in determining and classifying domain items and knowledge requirements. While this was echoed by other analysts, apparently T05 had more problems than other teams and it was reflected by a significant difference of T/A for the M7 variable.
T-test on Likert Scales

Since there were minimal significant team differences, t-tests were performed on values pooled across each analysis method per variable. These tests help characterize the analysis methods against individual scales. Analysts’ responses to each scale are reproduced in Appendix F as bubble histograms with associated t-test results.

T-test Results on Individual TMM Specific Scales

Table IV.9(a-b) summarizes significant t-tests (two-tailed tests on means against zero) for scales responded to by TMM analysts. These results characterize analyst reactions to using the TMM.

| Variable | Left Anchor (-3) | Mean & STD Error | T | Prob.>|T| |
|----------|------------------|------------------|---|--------|
| TMM Method: | | | | |
| M2       | Inadequate Power | +1.300 0.472     | 2.75 | 0.0224 |
|          | Adequate Power   |                  |    |        |
| M7       | Terrible         | +0.900 0.314     | 2.86 | 0.0187 |
|          | Wonderful        |                  |    |        |
| TMM Process: | | | | |
| P2       | Too Many Steps   | -1.100 0.433     | -2.53 | 0.0318 |
|          | Too Few Steps    |                  |    |        |
| P4       | Too Complex      | -0.900 0.378     | -2.37 | 0.0414 |
|          | Too Simplistic   |                  |    |        |
| TMM Notation and Formalisms: | | | | |
| N3       | Inadequate Rep. Power | 1.333 0.372 | 3.57 | 0.0072 |
|          | Adequate Rep. Power |                  |    |        |
| TMM Learnability (Process): | | | | |
| L2       | Long Time to Learn | -1.000 0.421 | -2.37 | 0.0418 |
|          | Short Time to Learn |                  |    |        |
### TABLE IV.9.B. SUMMARY OF SIGNIFICANT T-TESTS ON TMM-SPECIFIC SCALES

| Variable | Left Anchor (-3) | Mean & STD Error | T | Prob.>|T| |
|----------|------------------|------------------|---|------|
|          | Right Anchor (+3) |                  |   |      |
| **Using TMM Domains:** | | | | |
| D1       | Difficult to Use | Easy to Use | -1.050 0.368 | -2.84 0.0191 |
| **Determining if Knowledge Requirements are Supported or Unsupported:** | | | | |
| S1       | Difficult to Determine Easy to Determine | 1.500 0.401 | 3.73 0.0046 |
| S2       | Inadequate Support Adequate Support | 1.000 0.365 | 2.73 0.0229 |
| S4       | Dull Stimulating | 0.900 0.378 | 2.37 0.0414 |
| S5       | Frustrating Satisfying | 1.100 0.378 | 2.90 0.0174 |
| **Mean_S (average of S1 to S5)** | | | | |
|         | 1.050 0.218 | 4.80 0.0010 |
| **Synthesizing U1DRs using the TMM:** | | | | |
| U1       | Difficult to Synthesize Easy to Synthesize | 1.400 0.400 | 3.50 0.0067 |
| **Mean_U (average of U1 to U15)** | | | | |
|         | 0.880 0.360 | 2.44 0.0373 |

TMM evaluation results shown in Table IV.9(a-b) can be interpreted and restated as follows:

Variable | Interpretation |
----------|----------------|
[M2]      | TMM's method has adequate power. |
[iM7]     | TMM's method was more wonderful than terrible to use. |
[P2]      | TMM's process has many steps. |
[P4]      | TMM's process is complex. |
[N3]      | TMM has adequate representational power. |
[L2]      | TMM takes a considerable time to learn. |
[D1]      | TMM abstraction domains are difficult to use. |
[S1, S2, S4, S5] | Determining if knowledge requirements are supported or unsupported was easy, TMM provided adequate support, and it was stimulating and satisfying. |
[Mean_S]  | Overall, analysts reported favorable responses to S1-S5 for TMM. |
Using TMM to synthesize UIDRs was easy.

Overall, analysts reported favorable responses to U1-U5 for TMM.

In summarizing results of individual t-test results specific to TMM analysts, it is fair to say that results show several favorable findings; however, there was reported difficulty learning and using the TMM. Again, this is because the TMM is more complicated and has a richer set of methods and notations. One possible solution is to increase training for TMM. Because of experimental constraints (e.g., time, cost, subject availability) the training was limited to a two hour session, but one TMM analyst reported:

\[ S10 \] "Kevin, this material is more appropriate for a three-day workshop. Not a two hour seminar."

More training would help reduce analysts' trepidation about using TMM; however, a three day workshop was not feasible considering our budget. Yet, it is interesting to note that all analysts used TMM correctly. This suggests that their anxiety can be attributed to their lack of experience with the model and task description/decomposition—not just TMM's complexity.

**T-test Results on Individual DGL Specific Scales**

While the focus of this research was not evaluation of DGL, the DGL analysts were also polled for reactions concerning their analysis method. Table IV.10(a-c) summarizes significant t-tests (two-tailed tests on means against zero) for scales responded to by DGL analysts—full analyses are contained in Appendix F.
| Variable                      | Left Anchor (-3)       | Right Anchor (+3) | Mean & STD Error | T     | Prob.>|T| |
|------------------------------|------------------------|-------------------|------------------|-------|------|
| **DGL, Method:**             |                        |                   |                  |       |      |
| M1                           | Difficult to Use       | Easy to Use       | 1.700 0.422      | 4.01  | 0.0030 |
| M2                           | Inadequate Power       | Adequate Power    | 1.400 0.339      | 4.11  | 0.0026 |
| M3                           | Rigid                  | Flexible          | 1.500 0.401      | 3.73  | 0.0046 |
| M4                           | Not Usable             | Very Usable       | 1.800 0.200      | 9.00  | 0.0001 |
| M5                           | Frustrating            | Satisfying        | 0.900 0.314      | 2.86  | 0.0187 |
| **Mean_M**                   | Average of M1 to M7    |                   | 1.214 0.289      | 4.18  | 0.0023 |
| **DGL Process:**             |                        |                   |                  |       |      |
| P4                           | Too Complex            | Too Simplistic    | 0.600 0.163      | 3.67  | 0.0051 |
| P5                           | Unclear Starting Point | Clear Starting Point | 1.300 0.423    | 3.07  | 0.0133 |
| **Mean_P**                   | Average of P1 to P6    |                   | 0.533 0.161      | 3.32  | 0.0089 |
| **DGL Notation and Formalisms:** |                       |                   |                  |       |      |
| N1                           | Difficult              | Easy              | 1.700 0.396      | 4.29  | 0.0020 |
| N2                           | Complex                | Simplistic        | 1.800 0.200      | 9.00  | 0.0001 |
| N3                           | Inadequate Rep. Power  | Adequate Rep. Power | 1.700 0.396    | 4.29  | 0.0020 |
| N4                           | Rigid                  | Flexible          | 1.400 0.340      | 4.12  | 0.0026 |
| N5                           | Frustrating            | Satisfying        | 0.900 0.233      | 3.86  | 0.0039 |
| N7                           | Terrible               | Wonderful         | 0.800 0.200      | 4.00  | 0.0031 |
| **Mean_N**                   | Average of N1 to N7    |                   | 1.300 0.183      | 7.12  | 0.0001 |
## TABLE IV.10.B. SUMMARY OF SIGNIFICANT T-TESTS ON DGL-SPECIFIC SCALES

| Variable          | Left Anchor (-3) | Mean & STD Error | T    | Prob.>|T| |
|-------------------|------------------|------------------|------|-----|---|
|                   | Right Anchor (+3)|                  |      |     |   |
| DGL Learnability (Process): |                  |                  |      |     |   |
| L1                | Difficult to Learn| Easy to Learn    | 2.000| 0.298| 6.71 | 0.0001|
|                   |                  |                  |      |     |   |
| L2                | Long Time to Learn |              | 2.100| 0.277| 7.58 | 0.0001|
|                   | Short Time to Learn |          |      |     |   |
| L3                | Difficult Concepts | Easy Concepts | 2.000| 0.258| 7.75 | 0.0001|
| Mean_L1 (average of L1 to L3) |                  |                  | 2.033| 0.265| 7.67 | 0.0001|
| DGL Learnability (Notation): |                  |                  |      |     |   |
| L4                | Difficult to Learn | Easy to Learn    | 2.100| 0.277| 7.58 | 0.0001|
|                   |                  |                  |      |     |   |
| L5                | Long Time to Learn |              | 2.100| 0.379| 5.58 | 0.0004|
|                   | Short Time to Learn |          |      |     |   |
| L6                | Difficult Concepts | Easy Concepts | 2.200| 0.389| 5.66 | 0.0003|
| Mean_L2 (average of L3 to L6) |                  |                  | 2.133| 0.341| 6.25 | 0.0001|
TABLE IV.10.C. SUMMARY OF SIGNIFICANT T-TESTS ON DGL-SPECIFIC SCALES

| Variable | Left Anchor (-3) | Mean & STD Error | T | Prob.>|T1 |
|----------|------------------|------------------|---|--------|
|          | Right Anchor (+3) |                  |   |        |
| **Using DGL:** |                  | (test against centeredness) |   |        |
| U1       | Difficult to Use  | 1.700 0.335      | 5.07 | 0.0007 |
|          | Easy to Use      |                  |   |        |
| U2       | Inadequate Power | 0.900 0.277      | 3.25 | 0.0100 |
|          | Adequate Power   |                  |   |        |
| Mean_U  | (average of U1 to U15) | 0.940 0.270      | 3.48 | 0.0069 |
| **Selecting Necessary DGLs from Applicable DGLs:** |                  |                  |   |        |
| S1       | Difficult to Select | 1.000 0.394      | 2.54 | 0.0319 |
|          | Easy to Select    |                  |   |        |
| S2       | Inadequate Support | 0.800 0.327      | 2.45 | 0.0368 |
|          | Adequate Support  |                  |   |        |
| S3       | Rigid             | 0.900 0.277      | 3.25 | 0.0100 |
|          | Flexible          |                  |   |        |
| Mean_S  | (average of S1 to S5) | 0.760 0.260      | 2.93 | 0.0168 |
| **Synthesizing new UIDRs using DGLs (i.e., Tailoring):** |                  |                  |   |        |
| SU1      | Difficult to Synthesize | 2.100 0.277      | 7.58 | 0.0001 |
|          | Easy to Synthesize |                  |   |        |
| SU2      | Inadequate Capability | 1.700 0.300      | 5.67 | 0.0003 |
|          | Adequate Capability|                  |   |        |
| SU3      | Rigid             | 1.300 0.335      | 3.88 | 0.0037 |
|          | Flexible          |                  |   |        |
| SU5      | Frustrating       | 1.000 0.422      | 2.37 | 0.0418 |
|          | Satisfying        |                  |   |        |
| Mean_SU | (average of SU1 to SU5) | 1.420 0.272      | 5.21 | 0.0006 |

DGL evaluation results in Table IV.10(a-c) can be interpreted as follows:

Variable | Interpretation
----------|-----------------------------------
[M1-M5]   | The DGL method was easy to use, has adequate power, is flexible and satisfying to use.
[Mean_M]  | Overall, the analysts reported favorable responses to M1-M7 for DGL.
[P4, P5]  | The DGL method was simple and had a clear starting point.
Overall, analysts reported favorable responses to P1-P6 for DGL.

DGL method has easy and flexible notations and formalisms while still having adequate representational power.

Overall, analysts reported favorable responses to N1-N7 for DGL.

DGL process learnability is easy, takes a short time, and is composed of easy concepts.

DGL notation learnability is easy, takes a short time, and is composed of easy concepts.

Design guidelines are easy to use and have adequate power.

Overall, analysts reported favorable responses to U1-U5 for DGL.

Selecting design guidelines is easy and flexible, DGL methods provide adequate support for selection.

Overall, analysts reported favorable responses to S1-S5 for DGL.

DGL is flexible, stimulating, satisfying and has adequate capability to tailor design guidelines into UIDs. It is easy to synthesize UIDs using DGL.

Overall, analysts reported favorable responses to SU1-SU5 for DGL.

In summarizing results of individual t-test results specific to the DGL analysts, it is fair to say there are many favorable findings. Analysts had few problems using design guidelines. However, analysts did report a lack of support for knowing when to stop searching for applicable guidelines.

Results presented here for DGL are not surprising. The process of identifying violated guidelines for specific usability problems is (at the surface) inherently elementary: scan and compare the list of guidelines against an interface design. This process is easily understood and learned. In fact guides are a familiar concept, used in many areas in addition to interface design (e.g., plumbing guidelines, electrical guidelines, voting guidelines) and people are used to working with guidelines.
IV.6 PHASE 3: EVALUATE SYNTHESIZED UIDRS

IV.6.1 METHODS

Phase three of the TMM evaluation experiment involved quantitatively rating UIDRs generated in phase two—see Figure IV.4. All one hundred sixty UIDR sets (= two Analysis Methods x five Analysis Teams x four Interface Designs x four Usability Problems) were grouped by method and usability problem yielding thirty-two different UIDR sets—one for each usability problem per interface per analysis method. Other than removing duplicates, the experimenter faithfully transcribed the wording and structure of the synthesized UIDRs. Four evaluators were then asked to perform head-to-head comparisons of corresponding UIDR sets against several criteria.

![Phase 3: Evaluate the synthesized UIDRs](image)

FIGURE IV.4: PHASE 3 OF THE TMM EVALUATION EXPERIMENT (EXCERPT FROM FIGURE IV.1)

Criteria selected for the head-to-head comparisons of UIDR sets are separated into three classes: *USABILITY PROBLEM RELATED CRITERIA, USER RELATED CRITERIA, and ARTIFACT RELATED CRITERION*. Each of these classes of criteria represents a different facet of UIDR quality as determined by two HCI researchers/practitioners and the experimenter.
USABILITY PROBLEM RELATED CRITERIA

• ADDRESSES USABILITY PROBLEM
  UIDs address usability problems identified during formative evaluation. Therefore, the primary criterion for evaluating UIDs is based on the question: In comparison, how well does each UID set identify requirements for possible solution(s) to the particular usability problems?

• COHESIVENESS
  Cohesiveness is defined, for this purpose, to characterize UIDs that focus on one (or a select related few) concepts/ideas/interface features. That is, all UIDs in a set should be about the same thing. This criterion is concerned with cohesiveness within a single UID set—NOT between UID sets.

• SPECIFICITY
  Specificity is defined to characterize the level of detail and precision of the UID sets—ranging from ambiguous (or vague) to fully and precisely specified. In other words, this criterion focuses on the wording and completeness of description of the UIDs.

USER RELATED CRITERIA

• USER CENTEREDNESS
  The well known concept and practices of user centeredness means that design should focus on the user (Norman, 1986). The sets of UIDs should, therefore, reflect user-centeredness—NOT system-centeredness.

• ADDRESSES USERS’ NEEDS
  More specific than the previous user-centered criterion, this criterion compares how well each set of UIDs addresses users’ needs associated with task performance. The emphasis on users’ mental and physical needs is what makes this criterion different from the criterion for addressing the usability problem.

• ADDRESSES USERS’ TASK STRUCTURE
  Task structure is addressed in UIDs by support for the conceptualization of how tasks fit together to accomplish user work goal(s). This criterion is related to the overall task structure rather than simple tasks in isolation.
ARTIFACT RELATED CRITERION

- AVOIDS INTRODUCTION OF NEW USABILITY PROBLEMS
  The function of UIDRs is to provide designers requirements that address usability problems (identified in formative evaluation). In some cases, UIDRs can spur a designer in a direction that may solve the initial usability problem, but introduces additional usability problems.

The evaluators were trained in the use of these criteria for UIDR comparison purposes.

Participants. Four participants served as UIDR evaluators. Criteria for each evaluator was (at a minimum): a B.S. in computer science (or related field) and a graduate level introduction to issues and practices of human-computer interface analysis and design. These criteria served as a baseline helping to reduce variability through homogenous participants. The four evaluators worked independently.

Apparatus. Each evaluator required a Macintosh II computer equipped with prototyping software. This hardware was required to provide access to the user interface design prototype during evaluation sessions.

Procedure. This experimental phase was performed in the following manner:

- Experimenter normalized (i.e., sanitized) UIDRs to remove hints of design analysis technique, and merged each requirement set within each usability problem within each interface design. This merging was done separately for each of the both analysis methods. The result was a TMM set of requirements that addresses each usability problem for each interface design, as well as a DGL set.

- Experimenter provided a sixty minute lecture on appropriate evaluation and reporting techniques.

- Each evaluator examined each of the four interface designs (randomly ordered treatments) in turn. For each interface design, the evaluator compared and rated both UIDR sets associated with each of the four usability problems (also randomly ordered). Presentation order of analysis method associated with the UIDR sets, i.e., polarity of the rating scale, was also randomized. (Each evaluator performed sixteen analyses.)
• Each evaluator performed comparisons of UIDR sets corresponding to sixteen different usability problems. Comparisons were reported using 7-point Likert scales based on the seven quality criteria. The Likert scale provided for the comparison was:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is slightly better than Set 1</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
<td>[ ]</td>
</tr>
</tbody>
</table>

• Each box indicates the criterion's level of support in relation to both UIDR sets. The scale ranges from Set 1 being much better than Set 2 to Set 2 being much better than Set 1.

IV.6.2 RESULTS AND DISCUSSION

After all data were gathered, evaluators' reactions were translated and encoded based on presentation polarity of analysis methods. This had the effect of transcribing all results into the same polarity, i.e., all Set 1 UIDRs were generated using the DGL, and all Set 2 UIDRs were generated using TMM. Once this was done by the experimenter, the data was ready for further analysis. At this point, all positive scaled responses implied favorable results for TMM UIDR sets, while negative data implied favorable results for DGL UIDR sets. All raw data used in these analyses are reproduced in Appendix H: PHASE THREE DATA.

Results were first analyzed using a hierarchical 2-factor model (I = Interface; U/I = Usability Problem within Interface). Analysis showed no differences between interfaces for any of the criteria or the usability problems within the interfaces (i.e., I and U/I were not found to be significant). Analyses are reproduced in Appendix H: PHASE THREE DATA.

The next analysis examined the means of the reported UIDR quality criterion scaled responses. UIDR quality criterion scales ranged from -3 to +3, where

-3 indicates DGL UIDRs were much better for given criterion than TMM UIDRs, and

+3 indicates TMM UIDRs were much better for given criterion than DGL UIDRs.
Therefore, positive means indicate that TMM UIDRs were judged generally more favorable over the DGL synthesized UIDRs, and visa versa. Results for analyses that examined grand means for each UIDR quality criterion are presented in Table IV.11.

<table>
<thead>
<tr>
<th>TABLE IV.11. RESULTS OF COMPARING TMM AGAINST DGL</th>
</tr>
</thead>
<tbody>
<tr>
<td>UIDR Quality Analysis</td>
</tr>
<tr>
<td>Addresses Usability Problem (UP)</td>
</tr>
<tr>
<td>Cohesive (COH)</td>
</tr>
<tr>
<td>Specificity Level (SPEC)</td>
</tr>
<tr>
<td>Average Responses (of UP, COH, SPEC)</td>
</tr>
<tr>
<td>User Centered (UC)</td>
</tr>
<tr>
<td>Addresses User Needs (UN)</td>
</tr>
<tr>
<td>Addresses User Task Structure (UT)</td>
</tr>
<tr>
<td>Average Responses (of UC, UN, UT)</td>
</tr>
<tr>
<td>Avoids Introducing Usability Problems (AUP)</td>
</tr>
</tbody>
</table>

* = Significant to $\alpha \leq 0.05$  ** = Significant to $\alpha \leq 0.01$  *** = Significant to $\alpha \leq 0.001$

Evaluation results shown in Table IV.11 can be interpreted as follows:

**Variable** | **Interpretation**
--- | ---
[UP] | TMM produces UIDRs that address the usability problem better than DGL.
[SPEC] | TMM produces more specific UIDRs than DGL.
[R_UP] | On average, TMM produces UIDRs that
(1) better address usability problems,
(2) are more cohesive, and
(3) are more specific than UIDRs generated using DGL.
[UC] | UIDRs produced using TMM are more user centered than UIDRs produced using DGL.
[UN] | TMM produces UIDRs that address users' needs better than DGL.
[UT] | TMM produces UIDRs that address users' task structures better than DGL.
[R_USER] On average, TMM produces UIDs that
(1) are more user centered,
(2) better address user needs, and
(3) better address user task structure
than UIDs generated using DGL.

These results statistically support the general claim that TMM produces higher quality UIDs than DGL based on the defined criteria. Responses for all UID quality criteria (except COH and AUP) were all positive and significantly different from the median (i.e., different from the response Set 1 (DGL UIDs) is equivalent to Set 2 (TMM UIDs)). It is unclear why the COH and AUP criteria stand out. While both lean favorably toward the TMM, neither shows significant differences between TMM and DGL.

The TMM focuses on users and users’ tasks, whereas DGL focuses more on artifacts and less on users. Therefore, it was expected that TMM would rate more favorable in the UC, UN, UT, and R_USER, and the p-values (all less than 0.0012) support this contention.

IV.7 CONCLUSIONS

This chapter described rationale, procedures, and results of a three phase evaluation experiment which compared using TMM against using DGL to synthesize UIDs. In phase one, interfaces were selected and usability problem identified. Analysts, in phase two, examined the usability problems using either TMM or DGL to produce new UIDs. Evaluators, in phase three, compared TMM and DGL UIDs in relation to quality criteria.

Phase two results characterize learnability and usability of TMM and DGL. Specifically, analysts found DGL easier to learn and use than TMM (cf. Tables IV.5-IV.8, variables M1, Mean_N, L1-L6, Mean_L1, Mean_L2). Method specific learnability scales also point to DGL being easier than TMM. These are not surprising results, TMM has richer task description and knowledge analysis methodologies than DGL. Also, identifying violated guidelines for specific usability problems is (at the surface) inherently elementary.

Results from phase three statistically support the general claim that TMM produces higher quality UIDs than DGL as rated on seven quality criteria. Responses for five of the seven UIDR quality criteria showed evaluators rated TMM synthesized UIDs better than DGL synthesized
UIDRs. However, means of the UIDR quality criteria, while statistically significant, are not very different from the center point (i.e., zero). This raises a question of practicality: How much better is TMM over DGL? The evaluation performed here does not address this question. These positive results, although not as positive as we would have liked, give us reason to continue TMM research. We expect as TMM research grows and matures, its differences and analysis power will also increase.

Experimental findings are clear—TMM, while difficult to learn and use, produces higher quality UIDRs than DGL. These results point out that analysts subjectively rate TMM low in learnability and usability in comparison to DGL. However, objective results show that TMM’s UIDRs are better. Simply put, DGL is easier to understand and use, but it is less powerful than TMM.
V CONCLUSIONS

This dissertation presents definition and evaluation of a technique for improving user interface development. Primary importance of this work is the definition of the Task Mapping Model (TMM)—a synthesis-oriented, user-centered task analysis technique. By using situational analysis to synthesize user interface design requirements (UIDRs), TMM bridges the gap between formative evaluation and user interface design. This method is different from other methods because of its synthesis-orientation, and its fast-track focus on user mappings and knowledge requirements for task performance.

This chapter concludes the dissertation by providing discussion on (1) the TMM evaluation experiment, (2) contributions of the research, (3) limitations of the research, and (4) future research directions.

V.1 EVALUATION EXPERIMENT OVERVIEW

To date, our experience using the TMM has been limited to exploring its strengths, weaknesses, and applicability to user interface development. The TMM evaluation experiment explored using the model in comparison to using design guidelines (DGL) to synthesize new UIDRs for interface usability problems. In the experiment, both TMM and DGL produced UIDRs. The results of the study showed:
1. TMM produces UIDRs that address usability problems better than DGL.
2. TMM produces more specific UIDRs than DGL.
3. On average, TMM produces UIDRs that
   (1) better address usability problems,
   (2) are more cohesive, and
   (3) are more specific
   than UIDRs generated using DGL.
4. UIDRs produced using TMM are more user centered than UIDRs produced using DGL.
5. TMM produces UIDRs that address users’ needs better than DGL.
6. TMM produces UIDRs that address users’ task structures better than DGL.
7. On average, TMM produces UIDRs that
   (1) are more user centered,
   (2) better address user needs, and
   (3) better address user task structure
   than UIDRs generated using DGL.

These results illustrate the TMM’s utility for interface development (by synthesizing UIDRs) in comparison to the common DGL approach.

On the other hand, when the TMM analysts were asked, “What did you like best about using the [TMM] analysis method?” selected designers reported:

S2  “Once learned, it’s clear to follow and use. The [design] specification can be derived easily from the use of TMM.”

S8  “[TMM] Seems to be a good concrete way of identifying interface shortcomings. [We] found some problems we didn’t see at the beginning. [TMM] made formulating the problem easier.”
S13  "TMM's structure gave me confidence that I was addressing all aspects of the problem. The method allowed 'general' requirements to be formed in places where our team could not agree on a specific solution. We were able to apply the basic method partially and derive requirements."

However, when the TMM analysts were asked, "What did you like least about using the [TMM] analysis method?" selected designers replied:

S1  "Sometimes the distinction between computer domains and knowledge domains is not very clear. There is a 'steep' learning curve for the method."

S16  "The amount of time that is required to achieve an in-depth analysis. Despite the end-result centering, the detail that is required in the specification is at too low a level (in fairness, this has been true of every design analysis method I have used.)"

These subjective comments support the contention that the TMM training needs modification: accounting for analyst concerns, and experience in HCI and task analysis.
V.2 CONTRIBUTIONS OF RESEARCH

Contributions of the research reported here include ...

- conceptualizing and defining the TMM by:
  - providing a base for discussing TMM in relation to other HCI modeling techniques
  - identifying how TMM fits in and compares to other HCI modeling techniques
  - providing a *users guide, tutorial, and training materials* for the TMM
- showing that the TMM synthesizes significantly different UIDRs than DGL by:
  - experimentally comparing TMM and DGL synthesized UIDRs against a set of seven criteria
- identifying shortcomings in current use of DGL.

V.3 LIMITATIONS OF THIS RESEARCH

A limitation of this research which circumscribes generalizations is the subject population. Criteria were established to ensure a basic understanding of computer science and human-computer interaction. However, the population pool was limited to graduate students (M.S. and Ph.D.) from the Computer Science (CS) and Industrial and Systems Engineering (ISE) departments. Ideally, this evaluation experiment would have been performed on user interface designers and analysts in an industrial setting—but our budget could not support a venture of that size. Therefore, the next step is to repeat and validate the findings reported here using an industrial population and setting.

Another weakness is the evaluation looked at a single instance of using TMM within user interface development. Ideally, it would have been interesting to examine effects of using TMM while iterating within the interface development process.
V.4 FUTURE TMM RESEARCH DIRECTIONS

There are several interesting avenues of research that opened during our research on the TMM. These include, but are not limited to:

- Reevaluating TMM in light of research findings
  The evaluation experiment identified several shortcomings of TMM. These results need to be reexamined with an eye towards modifying and enhancing TMM analyses. While the objective results are positive, the qualitative results point out that TMM needs improvement. Critical to this effort is inclusion of more real-world data.

- Specification/Verification of TMM domain languages
  TMM domain languages (i.e., grammars and vocabularies) used to describe tasks need further specification and verification. Languages presented here are only initial approximations for task description languages. Also, further work must be done on integrating languages across the domain boundaries—this includes modifying the UAN.

- Investigating impact of analyst group size on TMM UIDR quality
  In initial experiences with TMM, designers felt more secure working in teams to perform analyses. While analysts worked in teams during the evaluation, the experiment did not address impact of analyst group size. Perhaps working in teams is beneficial to TMM analyses, but several questions are still unanswered such as: What is the optimum size for TMM analysis teams (i.e., the effects of group dynamics)? What is the ideal composition (i.e., team member roles) for TMM analysis teams?

- Investigating impact of expertise on TMM UIDR quality
  Expertise of analysts is a factor that should be explored. For example, analysts can have varying levels of expertise in phases of user interface development (e.g., needs assessment; systems-, user-, and functional-analyses; user interface design, implementation, and evaluation), or in the problem (i.e., application) domain. Impact of expertise on analyses can help determine appropriate and cost-effective training.
• Investigating appropriateness/feasibility/usability of TMM walk- and jog-throughs

TMM is an extensive modeling technique with several underlying methodologies for
task description and analysis. Using TMM can be labor-/time-intensive. Therefore,
it is important to explore possibilities of using the TMM in a walk- or jog-through
setting. These approaches trade-off analysis time for increased throughput. A study
is necessary to examine these approaches in terms of cost-effectiveness of analysis
quality to analysis time. In the study, I envision an approach using similar (but
simpler) characteristics and methodologies derived from TMM. Intuitively I believe
these approaches would be best applied to less critical usability problems.

• Verifying/Evaluating use of TMM in industrial settings

Initial research on the TMM has be limited to academic settings. An industrial study
is the next logical step in validation/evaluation of TMM.

• Verifying/Evaluating use of TMM for task analysis

TMM provides a framework for task description and analysis; however, in this initial
research the focus has been on formulating new UIDRs from identified usability
problems. However, the task description framework within TMM can be used for
initial task analysis—helping analysts identify user tasks, user task structures and
goals, user physical and mental needs for tasks.

• Verifying/Evaluating use of TMM for initial user interface design

Again, our focus on using TMM has been for deriving UIDRs; however, I feel that
TMM can also be used for initial user interface design by defining initial UIDRs and
providing user task structures for consideration by user interfaced designers. A
study is needed to support this belief.

• Incorporating Requirements Traceability into TMM

Using TMM to generate new UIDRs within the iterative user interface development
life cycle poses a problem: How do we trace requirements? This problem is common to
software engineering and referred to as Requirements Traceability. A mechanism
should be incorporated into TMM to include requirements traceability.

• Reevaluating Tailoring with respect to design guideline usage

The TMM evaluation experiment employed TMM and DGL to synthesize UIDRs.
These UIDRs were compared using a set of seven criteria. These criteria showed
favorable results for TMM; however, these criteria also identified shortcomings in us-
ing design guidelines for UIDS synthesis. In terms of design guideline research, these shortcomings should be addressed and resolved.
The profoundest thought or passion sleeps as in a mine, until an equal mind and heart finds and publishes it.
—Ralph Waldo Emerson

BIBLIOGRAPHY


APPENDIX A: TMM ANALYSIS MANUAL

The following guide is a reprint of:


It was given to the TMM analysts in phase two of the TMM evaluation experiment.
Task Mapping Model (TMM) Analysis Manual

— A Reference —

Version 1.6

Kevin A. Mayo
Department of Computer Science, Virginia Tech (VPI&SU)
Blacksburg, VA 24060
mayo@cs.vt.edu

I. Introduction
II. Overview of TMM Task Descriptions
III. TMM Task Description Domain Framework
   III.1. Problem Domain
   III.2. Computer Semantic Domain
   III.3. Computer Syntactic Domain
   III.4. Articulation Domain
   III.5. Domain Skipping
IV. TMM’s Description of User Knowledge Requirements
   IV.1. Factual Knowledge
   IV.2. Conceptual Knowledge
   IV.3. Procedural Knowledge
V. TMM Task Ordering Issues
VI. TMM View of User Classes
   VI.1. Dealing with Different User Classes
   VI.2. User Class Profiles
VII. Using TMM in User Interface Design
   VII.1. Generating TMM Task Descriptions (Methodology)
   VII.2. Unsupported-Knowledge Analysis (Methodology)
   VII.3. Interface Design Support Analysis (Methodology)
   VII.4. TMM Metrics
   VII.5. TMM as a Teaching/Documentation Device
VIII. Common Questions
IX. Example
Analysis Templates
Glossary
References

Appendix A: TMM Analysis Manual
1. Introduction

This guide presents the task mapping model (TMM), a synthesis/analysis methodology for aiding the interface specialist in designing interfaces with better usability. This guide does not motivate the necessity or impact of this model with regards to user interface development, that is presented elsewhere (Mayo & Hartson, 1993).

**What is the Task Mapping Model (TMM)?**

Briefly, TMM describes necessary knowledge for user task completion, and analyzes if this knowledge is supported by the user or interface. In order to examine user task knowledge, a model of the users' task is needed. TMM provides a framework for describing and analyzing tasks. This framework consists of various abstract levels of task decomposition and description. Each level of abstraction contains level-specific objects, operations, and sub-tasks for task description and analysis. TMM focuses on the mappings users make among domains (during task performance) to provide designers with specific information about task structure and user knowledge requirements. This model allows descriptions of tasks using hierarchical decomposition coupled with abstraction of user knowledge requirements.

**How does TMM effect the user interface development process?**

The overall goal of TMM is to support the user interface development process in relation to both initial design and redesign by deriving new interface design requirements. Because TMM is task oriented, analysis for initial design and redesign support are implemented in two ways: **global analysis**, **situational analysis**. Global analysis techniques examine whole systems—including all user tasks, input-output devices, and user interfaces. Global analysis examines all aspects of systems and is very costly in terms of time and effort. Situational analysis is involved with examination and analysis of a few specific user tasks and interface usability problems found by evaluators during formative evaluation cycles. During the initial design process both global and situational analysis can help the interface specialist focus on user- and task-centered issues. The redesign process has the benefit of formative evaluation findings, and thus, situational analysis is very cost-effective. TMM employs both analyses types for initial design and redesign during user interface development.

TMM may also be used as a synthesis or analysis method. As a synthesis method, TMM supports interface design throughout the design process by focusing interface specialists on user-centered needs. Using TMM descriptions of tasks that users had problems with, interface specialists can direct their attention on improving interface usability and human performance. Through analysis of specific situations (situational analysis) during formative evaluation, TMM synthesizes new interface design requirements.

As an analysis method, TMM helps interface specialists investigate what users need to know during task performance. With a TMM task description, user knowledge requirements can be analyzed in detail, based on task structure and relationships among domain contents. Further, TMM analysis points out interface deficiencies and provides
a synthesis of missing knowledge that should be made available to the users, and new interface design requirements are then derived to support that missing knowledge.

Where does TMM fit in the overall picture?

Human-computer interaction specialists often conceive their work two different viewpoints: the *constructional* view and the *behavioral* view. The constructional outlook is focused on implementation issues in HCI—where to put button, how to open a window, how can anyone program using X? While the behavioral view focuses on user and system behavior—how did the user do this, how did the system respond? Because of the user- and task-centered approach of TMM, it fits into the behavioral view.

Who uses TMM?

TMM can be used by interface specialists for both synthesis and analysis during human-computer interface design. Formally, TMM is designed for human-computer interface specialists; however, the simplicity and readability of the model’s descriptions allow more general use, including ‘walkthrough’-style evaluations by non-specialists.

The following sections provide a general introduction to TMM, and outline the domain structures, knowledge elements, and task description formats used in TMM. A section on TMM usage outlines associated methodologies, and finally, an example is provided to illustrate the model’s use. This guide also has a glossary defining commonly used terms.

II. Overview of TMM Task Descriptions

This section gives a general overview of describing tasks with TMM.

Domains

The abstract levels of TMM task descriptions are referred to as *domains*. A domain contains entities specific to the abstraction level. These entities are collectively referred to as *domain items*. A domain item is either an *object*, *operations*, or *sub-tasks*. Domain objects are manifestations of physical or conceptual *things* related to the task. Operations are actions defined within the domain’s level of abstraction. Sub-tasks represent subordinate execution sequences. All three components are (examples are given in the following section).

Domains range from ‘higher’ to ‘lower’ levels of task abstraction. The domains reflect different levels of abstraction at which users perceive task artifacts (domain items). (Shneiderman, 1979; Shneiderman, 1987; Moran, 1981; Nielsen, 1986) Simply put, a user works at several levels when performing a task on a computer—from the highly abstract problem domain down to the low abstraction level of physical manipulation.

The domains used in TMM task descriptions are: *problem domain*, *computer semantic domain*, *computer syntactic domain*, and *articulation domain*. These domains are discussed in more detail, with examples, in the following section. The following figure depicts the domains of task abstraction.
Pictorial Representation of Relationship among Domain Levels

Ideally, users wish to work in the problem domain, where all domain items are problem related; however, once a task is computerized, users must understand the translations between their understanding of the task and the computers representation of the task.

For example, when a user attempts the task of **DUPLICATE A DOCUMENT** (problem domain task), the user must reconceptualize it into **COPY A FILE** (computer semantic domain). Many of these tasks can be viewed as action-object pairs that exist within different domains, and are both representations of the task at hand. Also, **COPY A FILE** can be reconceptualized further into the ‘computer’ grammatical components (computer syntactic domain items representing the ‘copy command’ and the actual filename) and input sequence (articulation domain specification of users’ actions).

Example Task (**DUPLICATE A DOCUMENT**) within Domain Levels
**Mappings**

Using these domains to represent the different abstract levels that users perceive tasks, mappings are formed to relate items across domain boundaries. A **mapping** refers to a specific reconceptualization or translation of a domain item from one domain to another. Mappings can exist from higher to lower or lower to higher abstract domains levels. The sequence of mappings through these domains are **task paths**, and may be either **execution** or **evaluation**. Mappings from higher to lower levels of abstraction represent execution paths. Here the user maps conceptual task goals into actual physical actions. The converse, examining physical results to determine is a goal has been achieved, is an evaluation path—users map from lower to higher levels of abstraction.

In the previous example, **DUPLICATE A DOCUMENT** was reconceptualized (translated) to **COPY A FILE**. During task performance the user maps **DUPLICATE** to **COPY** and **DOCUMENT** to **FILE**. TMM notation uses an arrow, \( \rightarrow \), to indicate a 'maps to' relationship, e.g., **DUPLICATE** \( \rightarrow \)** **COPY** and **DOCUMENT** \( \rightarrow \)** **FILE**. These are two of the necessary mappings for task translation in this example.

In general, mappings link domain items across domain boundaries. The following figure depicts the relationships of mappings and domain items among domains:

![Diagram of mappings among domains](image)

**Pictorial Representation of Mappings within Domain Structure**

This example shows mappings among different domains. Mappings within domain boundaries are further decompositions of domain items; while, mappings among items from different domains boundaries represent changes in abstraction levels.

This example illustrates a 'one-to-many' mapping from the single problem domain item to two computer semantic domain items. This could represent a possible alternative method for performing the same task, or these mappings can indicate a second possible computer semantic representation of the problem domain item. 'One-to-many' mappings can occur between any two domains.

'Many-to-one' mappings are also possible. Our example shows two computer semantic domain items mapping to a single computer syntactic domain item. This could represent a collapsing of the task into a common set of grammatical components or actions.
example, consider error conditions where once an error is identified a sequence of actions common to all errors must be taken for error resolution. Observe the errors associated with reading-from or writing-to a file—once the error is determined the user must check the media and then remount it. 'Many-to-one' mappings can occur between any two domains.

With mappings, our previous example task of duplicate a document might look like:

![Diagram showing problem domain, computer semantic domain, computer syntactic domain, and articulation domain (UAN)]

*Higher Abstraction*  

**Example Task (duplicate a document) with Mappings**

In this example, duplicate ⇒ copy ⇒ actual system 'Copy' command (e.g., 'cp', 'Duplicate' menu item) ⇒ users' physical actions. This series of mappings illustrates the user's reconceptualization of a problem domain task down into physical user actions.

Mapping one domain item into another item in a different domain is not altogether straightforward. For instance, what is involved when the user performs document ⇒ file? The user requires certain knowledge to perform this mapping. Therefore, sets of necessary knowledge elements are associated with each mapping, representing the knowledge requirements for the mapping. Also, a mapping may have alternative knowledge sets. The following figure depicts this concept with example generic mappings:
In the previous figure notice that mappings connect domain items across domain boundaries. The knowledge necessary for each mapping is 'external' to TMM's domains. In other words, domains contain domain items, while knowledge requirements are generally defined outside or between the domains. In the **DUPLICATE A DOCUMENT** example, consider the mapping **DOCUMENT ⇒ FILE**, one possible knowledge requirement could be the factual knowledge that **documents are stored within files.**, as shown in the following figure:

This example illustrates that users must understand that documents are stored in system files in order to map document to file. This may seem trivial, but a user without this understanding is doomed.
This section has presented a brief overview of TMM. The following sections depict the domains, knowledge elements, and task description issues.

III. TMM Task Description Domain Framework

This section further defines the domains introduced in the previous section: problem domain, computer semantic domain, computer syntactic domain, and articulation domain.

III.1. Problem Domain

The problem domain contains items (objects, operations, and sub-tasks) directly related to user task performance described in specific real-world terminology. Thus, the scope of this domain is defined by the users' task or job. Domain items are the physical and conceptual real-world entities that comprise the users' task, and are often not related to computer systems. The problem domain does not contain anything defined outside this scope.

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th>Objects</th>
<th>Operations</th>
<th>Sub-Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENT PREP.</td>
<td>• LETTER</td>
<td>• EDIT</td>
<td>• MOVE LINE 1 TO 5</td>
</tr>
<tr>
<td></td>
<td>• DOCUMENT</td>
<td>• READ</td>
<td>• EDIT DOCUMENT</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• DISCARD LETTER</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CHECK SPELLING</td>
</tr>
<tr>
<td>PERSONAL FINANCES</td>
<td>• CREDIT CARDS</td>
<td>• DEPOSIT</td>
<td>• TRANSFER 320 TO SAVINGS</td>
</tr>
<tr>
<td></td>
<td>• DRAFT ACCT.</td>
<td>• WITHDRAW</td>
<td>• BALANCE CHECKBOOK</td>
</tr>
<tr>
<td></td>
<td>• SAVINGS ACCT.</td>
<td>• TRANSFER</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• DEPOSIT SLIP</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• BALANCE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This table shows two different problem domains with representative domain items. Obviously, this is not a complete table—e.g., documents can also be printed, reformatted, etc. This table does illustrate that all domain items are problem defined, and there is no hint of possible computer implementations or solutions. The objects defined are within the problem domain—a letter is a physical world 'letter' and not a sequence of ones and zeros in a computer somewhere.

As can be seen, there are no items concerned with computer hardware or software in this domain. This level of abstraction requires that only the problem domain and tasks within the problem domain need to be understood.
III.2. Computer Semantic Domain

The computer semantic domain contains items (objects, operations, and sub-tasks) that build a representation of the users' tasks with abstract computer concepts. This domain is a layer of abstraction between the problem domain task representation and the specific computer syntactic domain task representation. I.e., this domain serves as an generic computer "middle-ground" in the users' translation of tasks between the real world and a specific computer system.

<table>
<thead>
<tr>
<th>Objects</th>
<th>Operations</th>
<th>Sub-Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>• FILE</td>
<td>• MOVE</td>
<td>• MOVE A DATA FILE FROM</td>
</tr>
<tr>
<td>• DISK</td>
<td>• DELETE</td>
<td>APPLICATION FOLDER TO</td>
</tr>
<tr>
<td>• KEYBOARD</td>
<td>• RENAME</td>
<td>DATA DISK</td>
</tr>
<tr>
<td>• WINDOWS</td>
<td></td>
<td>• INSERT A DISK</td>
</tr>
</tbody>
</table>

Items within this domain are strictly computer related, but their definition is not limited to any particular hardware or software platform. Items in this domain do not include the actual interface features, but this domain does include the generalized concepts represented by these features. For example, FILES are common to most systems but their physical representations among these systems vary considerably.

The TMM has chosen the taxonomy reported by Lenorovitz as the standard language within the domains. (Lenorovitz, Phillips, Ardrey, & Kloster, 1984) By adopting this taxonomy as standard, the TMM helps eliminate ambiguities from designers' natural language task descriptions—while not relying on constrictive grammars. This also gives a common language by which designers and analysts can communicate.

The taxonomy is divided into four sub-taxonomies of the User-System Interface (USI): computer-output, computer-internal, user-input, and user-internal taxonomies. The user-input and user-internal taxonomies are of interest to TMM—these form the basic language used in TMM task descriptions. This language is used throughout the problem-, computer-semantic-, and computer-syntactic domains. These two taxonomies are outlined hierarchically:
The definitions are given in the following table (still showing the hierarchy):
| PERCEIVE | ACQUIRE | DETECT | Discover or notice an occurrence (usually unsolicited) |
| SEARCH | Purposeful exploration or looking for specified item(s). |
| SCAN | Glance over quickly, usually looking for overall patterns or anomalous occurrences (not details). |
| EXTRACT | Directed, attentive reading, observing, or listening with the purpose of gleaning the meaning or contents thereof. |
| CROSS-REFERENCE | Accessing or looking up related information usually by means of an indexing or organized structuring scheme set up for that purpose. |
| IDENTIFY | DISCRIMINATE | Roughly classify or differentiate an entity in terms of a gross level grouping or set membership—frequently on the basis of only a limited number of attributes. |
| RECOGNIZE | Specific, positive identification of an entity. |

| MEDIATE | ANALYZE | CATEGORIZE | Classify or sort one or more entities into specific sets or groupings, usually on the basis of a well-defined classification scheme. |
| CALCULATE | Reckon, mentally compute, or computationally determine. |
| ITEMIZE | List or specify the various components of a grouping. |
| TABULATE | Tally or enumerate the frequencies or values of the members of an itemized list or table. |
| SYNTHESIZE | ESTIMATE | Mentally gauge, judge, or approximate, often on the basis of incomplete data. |
| INTERPOLATE | Assign an approximate value to an interim point based upon knowledge of values of two or more bracketing reference points. |
| TRANSLATE | Convert or change from one form or representational system to another according to some consistent mapping scheme. |
| INTEGRATE | Pull together, and mentally organize a variety of data elements so as to extract the information contained therein. |
| FORMULATE | Generate and put together a set of ideas so as to produce an integrated concept or plan. |
| PROJECT OR EXTRAPOLATE | Assign an approximate value to a future point based upon the value(s) of preceding point(s). |

| ASSESS | COMPARE | Consider two or more entities in parallel so as to note relative similarities and differences. |
| EVALUATE | Determine the value, amount, or worth of an entity, often on the basis of a standard rating scale or metric. |
| DECIDE | Arrive at an answer, choice, or conclusion. |

| COMMUNICATE | TRANSMIT | CALL | Signal to a specific recipient or set of recipients that a message is forthcoming. |
| ACKNOWLEDGE | Confirm that a call or message has been received. |
| RESPOND | Answer or reply in reaction to an input. |
| SUGGEST | Offer for consideration. |
| DIRECT | Provide explicitly authoritative instructions. |
| INFORM | Pass on or relay new knowledge or data. |
| INSTRUCT | Teach, educate, or provide remedial data. |
| REQUEST | Solicit, query, or ask for. |
| RECEIVE | Get, obtain, or acquire an incoming message. |

_user-internal taxonomy_

Definitions for Lenorovitz Taxonomies (Lenorovitz et al., 1984)
<table>
<thead>
<tr>
<th>CREATE</th>
<th>ASSOCIATE</th>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GROUP</td>
<td>Link together or associate for purposes of identification.</td>
<td></td>
</tr>
<tr>
<td>INTRODUCE</td>
<td>INSERT</td>
<td>Make space for and place an entity at a selected location within the bounds of another such that the latter wholly encompasses the former, and the former becomes an integral component of the latter.</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLE</td>
<td>AGGREGATE</td>
<td>Combine two or more components so as to form a new composite entity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OVERLAY</td>
<td>Superimpose one entity on top of another so as to affect a composite appearance while still retaining the separability of each component layer.</td>
<td></td>
</tr>
<tr>
<td>REPPLICATE</td>
<td>COPY</td>
<td>Reproduce one or more duplicated of an entity (no links to master).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSTANCE</td>
<td>Reproduce an original (master) entity in such a way as to retain a definitional link to the master—i.e., such that any subsequent changes or modifications made to the master will automatically be reflected in each and every instance created therefrom.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>INDICATE</th>
<th>SELECT</th>
<th>POS/OBJ</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SELECT</td>
<td></td>
<td>Opt for or choose an entity (e.g., a position or an object) by pointing to it [and possibly other user actions].</td>
</tr>
<tr>
<td></td>
<td>REFERENCE</td>
<td></td>
<td>Opt for or choose an entity by invoking its name.</td>
</tr>
</tbody>
</table>

| ELIMINATE | REMOVE | CUT | Remove a designated portion of an entity and place it in a special purpose buffer (residual components of the original entity usually close in around hole left by cut-out portion). |
|          | REMOVE | DELETE | Remove and (irrevocably) destroy a designated portion of an entity. |
|          | SUSPEND |        | Stop a process and temporarily hold in abeyance for future restoration. |
|          | TERMINATE |        | Conclude a process such that it cannot be restarted from the point of interruption, only by complete re-initiation. |

| DISASSOCIATE | RENAME | Change an entity’s title or label, without changing the entity itself. |
| DISASSEMBLE | UN-GROUP | Eliminate the common bond or reference linkage of a group of entities. |
| DISASSEMBLE | SEGREGATE | Partition and separate an entity into one or more component parts such that the structure and identity of the original is lost. |
| DISASSEMBLE | FILTER | Selectively eliminate one or more layers of an overlayed composite. |
| DISASSEMBLE | SUPPRESS | Conceal or keep back certain aspects or products of a process without affecting the process itself (i.e., affects appearance only). |
| DISASSEMBLE | SET-ASIDE | Remove entire contents of current (active) work area and store in a readily accessible buffer (for future recall). |

| MANIPULATE | TRANSFORM | Manipulate or change one or more of an entity’s attributes (e.g., color, line type, character font, size, orientation) without changing the essential content of the entity itself. |

| ACTIVATE | EXECUTE | FMT | Initiate or activate any of a set of predefined utility or special purpose functions (e.g., sort, merge, calculate, update, extract, search, replace). |

**User-Input Taxonomy**

Definitions for Lenorovitz Taxonomies (Lenorovitz, et al., 1984)
### III.3. Computer Syntactic Domain

Computer syntactic domain items (objects, operations, and sub-tasks) represent actual syntactical components related to both computer semantic and articulation domain items. These items represent user interface software entities (e.g., specific filenames, interface objects) and actions (e.g., user physical actions, interface commands) employed during task performance with particular computer or software packages.

This domain outlines the syntax—the components of articulation and their order. In other words, the computer semantic domain describes *what* to express to the computer, whereas the computer syntactic domain describes *how* to express to the computer. Thus, items in the computer syntactic domain are somewhat specific to general interaction styles and techniques.

Many components are not fully specified at this level of task description because of ambiguities in users' actions. (E.g., the interface specialist may not know the exact filename or command.) Therefore, variables may be used at this level of task description. Domain items with brackets, `< >`, are variables in TMM task descriptions that represent actual system equivalents. For example, `<FILENAME>` would reduce to the actual system filename like `exp.dat`, and `<FILE ICON>` is the system-displayed icon for a particular file used within the task. With variables a task description still maintains adequate expressive power, while the 'noise' of variable qualification is left to user actions.

It is important to understand that computer syntactic domain items represent syntactical items for articulation—not the manipulation or articulation of the syntactical items. By the vary nature of command line interfaces (CLI), the computer syntactic domain items will fully specify the commands; however, in direct manipulation (DM) interfaces only the components of the commands are specified. To illustrate the differences between a CLI and DM style interfaces, consider the task *ERASE FILE*:

<table>
<thead>
<tr>
<th>Example computer semantic items</th>
<th>Computer Semantic Items</th>
<th>Interaction Style</th>
<th>Computer Syntactic Items</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ERASE</code></td>
<td><code>FILE</code></td>
<td><strong>COMMAND, LINE, INTERFACE</strong></td>
<td><code>INVOKER 'RM'</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>SELECT &lt;FILENAME&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>INVOKER 'DEL'</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>SELECT &lt;FILENAME&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DIRECT, MANIPULATION</strong></td>
<td><code>SELECT &lt;FILEICON&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>MOVE &lt;FILE ICON&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>TO &lt;TRASHCAN ICON&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>Deselect &lt;FILEICON&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>MICROSOFT WINDOWS</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>INVOKER &lt;DELETE&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>MENU ITEM FROM &lt;FILE&gt;</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>PULLDOWN MENU</code></td>
</tr>
</tbody>
</table>
These example syntactic domain items are, in effect, a meta-language for the articulation. The items specify the articulation in terms of actual commands and arguments (albeit through variables), but they do not specify the articulation. For instance, the UNIX-CLI syntactic items are `INVOKED 'RM'` and `SELECT <FILENAME>`, but the articulation of these items is not specified (in fact, it may be either by keyboard, voice, or some other form). The syntax for the syntactical domain items is fully defined and described by the interface specialist.

Syntactical representations described within this domain map into the articulation domain. The specification for the articulation needs only be described once. For example, the tasks `ERASE FILE` and `COPY FILE share a common computer syntactic domain item: SELECT <FILENAME>`. The articulation for this syntactic domain item needs only be described once, and then referenced in the future. These representations of syntactical items can be kept in a dictionary of articulation descriptions.

III.4. Articulation Domain

The articulation domain contains the specification of the users' physical interaction while communicating computer syntactic items to and from the interface. This domain models the users' actions with the syntax of the interaction defined within the computer syntactic domain. Because of its ability to specify users' actions and system feedback, the User Action Notation (UAN) (Hartson, Siochi, & Hix, 1990) is used within this domain.

The articulation domain contains both system feedback to users and user inputs to the system. The system's interpretation of user inputs is a purely constructional issue, and therefore not important to this research. However, the users' ability to understand system feedback is a behavioral issue very important within TMM.

Users must understand system feedback because it usually affects subsequent actions. Hence, evaluation paths in TMM task descriptions mostly originate from system feedback. TMM helps determine the usefulness of the feedback by identifying ambiguous, misleading, or nonexistent feedback based upon the evaluation paths and users' knowledge requirements. For example, the feedback `FILE ID=0092KAM REMOVED` could be understood better as `LETTER TO JOE SCHMALTZ DISCARDED`. In the first case the feedback is in cryptic computer-syntactic domain terms; while in the second case, feedback is provided in problem domain specific terms. The TMM task description for this task identifies several mappings users must perform to relate `ID=0092KAM` back to `LETTER TO JOE SCHMALTZ`. The second feedback example (problem-domain-specific) removes many of the file name mappings identified by TMM, and hence makes the feedback easier for the user to understand. Of course, feedback cannot always be represented within the problem domain—a DISK FULL error is a good example.

The articulation level can also help in identifying interface characteristics that could inhibit task performance for certain classes of users. For example, how would a seeing person deal with a Braille screen, or a speech impaired individual with a voice activated system? Examining user class profiles in conjunction with the articulation domain could identify usability problems, this is discussed in the section TMM View of User Classes.
III.5. Domain Skipping

TMM task descriptions capture the flow of user actions in terms of tasks and sub-tasks, as well as the necessary knowledge for these actions. It is possible, given the TMM domain framework, that users can 'skip' a domain. Skipping a domain implies that the user does not have or want an understanding of the task within the domain. For example, it is possible for a user to have memorized a keystroke sequence for a particular task without an understanding of the underlying commands that are being issued. TMM analyses supports this style of translation; however, a user with a fragile task understanding represents usability problems waiting to happen.

IV. TMM’s Description of User Knowledge Requirements

Each mapping is associated with a set of knowledge requirements. This set of knowledge requirements represents the necessary information that users must possess to perform that mapping. Knowledge takes several forms within TMM task descriptions: factual knowledge, conceptual knowledge, and procedural knowledge (much like schemas—see (Evans, 1988)). Knowledge from these categories may be necessary for successful task completion. It is not the intention of this research to define mental models defining what users believe, how they store/retrieve information, or the relationship structure among user knowledge. These categories serve to classify for knowledge, but not the ability to model knowledge manipulation or relationships.

IV.1. Factual Knowledge

Definition: Factual knowledge is comprised of single or collections of declarative facts users need for mapping items from one domain to another.

E.g:

- problem-computer semantic mapping that requires the factual knowledge: FAMILY TREE INFORMATION IS STORED IN A DATABASE.
- computer semantic-computer syntactic mapping that requires the factual knowledge: COLUMN DELETION COMMAND IS 'COL-DEL'.
- computer syntactic-articulation mapping that requires the factual knowledge: KEYBOARD IS QWERTY STYLE.

The first and most straightforward category of knowledge is factual knowledge. Factual knowledge is information necessary to the user. The information can be a single
statement or a series of statements taken as a whole. Both files are stored on disk and the first column is monthly sales are examples of factual knowledge elements. These examples help to illustrate that factual knowledge is necessary for task completion, and also that the requirement for specific factual knowledge can be associated with a mapping between any domain levels.

The notation that TMM uses for factual knowledge is FK: <STATEMENT>. The knowledge, <STATEMENT>, is a natural language representation of a single fact. As an example of a mapping requirement with corresponding factual knowledge, consider the task: SEND EMAIL TO JOHN DOE:

<table>
<thead>
<tr>
<th>Problem Domain</th>
<th>Computer Semantic Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>Send EMail to John Doe</td>
<td>Edit &lt;Message&gt; with &lt;Editor&gt;</td>
</tr>
<tr>
<td>Compose Message</td>
<td>Message has been created</td>
</tr>
<tr>
<td>Mail Message</td>
<td>Mail &lt;Message&gt; to &lt;J.Doe Mail Address&gt; with &lt;Mail System&gt;</td>
</tr>
</tbody>
</table>

Key:

1: FK: Messages are contained within files
2: FK: Files are changed through editing
3: FK: Message is prepared
4: FK: All EMail must have an address
5: FK: John Doe has an EMail address
6: FK: EMail is sent through mail system

Example of Factual Knowledge

In this superficial example the user must know about system files, edit and mail systems, and user email addressing. These represent a few of the necessary factual knowledge elements for this task. This example also shows factual knowledge used in both execution and evaluation paths.
IV.2. Conceptual Knowledge

Definition: **Conceptual knowledge** is the understanding of relationships between collections of factual knowledge, ideas, and other conceptual knowledge.

*E.g.*

- problem-computer semantic mapping that requires conceptual knowledge about BIRDS (including: general characteristics, color, shape, wings, etc.)
- computer semantic-computer syntactic mapping that requires conceptual knowledge about FILES (including: general characteristics, commands, contents, etc.)
- computer syntactic-articulation mapping that requires conceptual knowledge about VOICE ACTIVATED SYSTEMS (including: general characteristics, commands, initialization, etc.)

The second category of knowledge is conceptual knowledge. A concept is defined to be a collection of ideas, characteristics, and/or other concepts. For example, a user has a conceptual understanding of FILES, and therefore, the user may understand operations and general characteristics of FILES but not necessarily the underlying system representation and operations.

Conceptual knowledge can also be used when the definition of the users' factual knowledge is too cumbersome. For example, it would be very difficult to generate the entire list of facts associated with FILES in a TMM task description, so conceptual knowledge can be employed as a shorthand notation. It also is used to indicate general knowledge of a topic without reference to specifics. If a user is known to possess given conceptual knowledge, then sometimes possession of certain factual knowledge can be deduced or inferred. Like factual knowledge, conceptual knowledge can be used throughout a TMM task description.
The previous figure depicts an example of using a conceptual knowledge element within a mapping. Here the task is to delete a file from a Macintosh-like interface. This involves moving the file icon to a deletion (trashcan) icon. The conceptual knowledge element represents knowledge of direct manipulation interfaces. This knowledge encompasses movement, selection, and the direct manipulation paradigm of object-action pairs.
IV.3. Procedural Knowledge

Definition: Procedural knowledge represents possible courses of action, user goals, task ordering and structure, needed during task performance.

E.g.
- problem-computer semantic mapping that requires procedural knowledge: TASK: LOCATE DATABASE ASSOCIATED WITH QUARTERLY REPORT.
- computer semantic-computer syntactic mapping that requires procedural knowledge: TASK: PERFORM VISUAL SEARCH FOR PRINT COMMAND.
- computer syntactic-articulation mapping that requires procedural knowledge: TASK: MOVE POINTER/CURSOR TO RIGHT-HAND SIDE OF BOX OUTLINE (SCREEN POSITION: 655, 231).

The final knowledge category is procedural knowledge. Procedural knowledge represents possible courses of action, user goals, task ordering and structure, needed during task performance. In simpler terms, this knowledge represents ‘where-am-I’, ‘how-did-I-get-here’ and ‘what-do-I-do-next’ task knowledge. There are two different types of procedural knowledge: knowledge of possible task paths, and knowledge to correctly choose a path.

The first type of procedural knowledge outlines the possible paths users may take. Much of this knowledge is embedded into the task decomposition. In particular, users must be aware of tasks that have multiple orderings in time. For example, task A requires both task B and C to be completed (in either order)—there are two paths for task A: B then C, or C then B. This type of procedural knowledge identifies timing relationships among tasks—which defines possible task paths.

Knowledge about task paths also includes methods that help the users’ mapping process among task abstraction domains. Consider the problem of locating an interface artifact, such as a menu item, the associated procedural knowledge identifies the need for information about searching menus. Said another way, procedural knowledge also describes auxiliary tasks and information that users employ to assist their own task mapping needs.

The second type of procedural knowledge is concerned with conditional task execution and evaluation. Simply put, a task may conditionally execute. A user should know and understand the condition that controls task execution. For example, if the condition is RECORD RECENT DATA ONLY for the task STORE DATA, then the user must be aware of the condition and be able to evaluate it.

Tasks can have several outcomes, and users faced with evaluating the results. Procedural knowledge includes the conditions that control the task outcomes. For example, the task FIND CORRECT FILE has both a positive (CORRECT FILE FOUND) and negative (FILE NOT FOUND) outcomes. Users must know the correct path based on the condition to follow (i.e., where-do-I-go-next). Procedural knowledge for this task includes both outcomes and the
paths the users take. Conditions associated with the outcomes are specified using natural language, and can indicate either an execution or evaluation paths.

Users can maintain multiple methods for the same task. However, this modeling does not attempt to place conditionals on equally likely method selection based on user class profiles. This is because TMM does not model mental processes but only identifies knowledge necessary for task accomplishment.

As an example, consider the problem domain task of **Locate a known movie in movie store computer**, this maps into the computer semantic domain items: **movie name** and **search**. (For this task, it is assumed the user knows the movie name.)

Key:

1: FK: The movie name, <Movie>, is a * criteria for search
2: FK: <Search> is the command to retrieve data from database
3: PK: <Results> are examined by user visual search

Also,

PK: **Structure of task**: Locate a known movie in video store computer
PK: **Where to go after task**: Examine Data from search

### Example of Procedural Knowledge

In this task, the user must specify the criteria (the movie name) and invoke the search. Factual knowledge represents the actual command names, but procedural knowledge describes the necessary task structure. Further, once output is generated from the search, the user must examine the data to check for accuracy in the search. Either the movie data was found or it wasn’t (notice the condition within the task description). This is another example of procedural knowledge. The user must know to verify the output, and also know the next action base on the verification.

## V. TMM Task Ordering Issues

Correct and accurate task description also involves modeling the ordering constraints among tasks. Tasks may be ordered in many different ways. In command line interfaces a sequential task ordering is standard; however, with direct manipulation windowing systems and the increase of desk-top computer power, users are now challenged with performing several tasks at once, possibly sequentially, concurrently, or interleaved.
User interface designs have taken these new interaction possibilities to users, and in many cases, seriously changed the complexity of user tasks.

If a user may perform a given task in several different ways, then TMM should capture that ability in the task descriptions. TMM provides several notations to model timing relationships.

TMM task descriptions use brackets to delineate tasks and sub-tasks into related aggregates. The timing relationships defined in this section can be applied to either single tasks or aggregates of tasks grouped within brackets.

**Sequential tasks**

If tasks are serially ordered in time, then they are **sequential tasks**. This timing relationship among tasks is very common. For example, consider using your car’s computer to SET CRUISE CONTROL SPEED TO 55 MPH. The sub-tasks are (1) GET CAR TO 55 MPH, (2) SET CRUISE CONTROL SPEED. These tasks are ordered in time. Setting the cruise control before the correct speed is attained leads to a task execution error. TMM task description notation for this example is:

```
[Task: Get Car to 55 mph.
Task: Set Cruise Control Speed]
```

**Example of Sequential Tasks**

**Conditional task performance**

Often a task is performed based on a condition that the user evaluates. For example, consider the task EXAMINE DATASET FOR ERRORS. The sub-tasks could include: Evaluate Dataset, and Correct Dataset. However, the execution of correct dataset is **dependent on whether an error exists in the dataset**—a user evaluated conditional. The TMM notation for this uses ‘IF <condition>’ as in:

```
Task: Examine dataset for errors

  Task: Evaluate Dataset

  IF an error exists in the dataset
  Task: Correct Dataset
```

**Example of Conditional Task Performance**

Conditionals can also be associated with task mappings—an alternative notation for conditional task performance. These notations allow designers to build up conditional execution sequences (using nesting) based on user evaluation of conditionals.

**Repeated tasks**

If a task is to be performed several times, then it is defined to be a **repeating task**. For example, if the task is CHANGE FONT TYPE FOR CITY NAMES which are dispersed throughout a document, this could be defined as the grouped tasks LOCATE CITY NAME and CHANGE FONT TYPE repeated zero or more times. The TMM notation for this uses ‘DO *’ as in:
DO *
  Task: Locate City Name
  Task: Change Font Type

Example of Performing a Task Zero or More Times

The "*" represents zero or more occurrences of the tasks within the brackets. Sometimes it is known that the task(s) will be executed at least once, and in this case, the '+' notation represents one or more executions of the tasks within the brackets. The following example provides the same task timing relationships but with two different notations; one uses the 'DO +' operator, while the other uses the 'DO *' operator.

Notation Alternative 1:

- DO +
  Task: Locate City Name
  Task: Change Font Type

Notation Alternative 2:

- Task: Locate City Name
- Task: Change Font Type
- DO *
  Task: Locate City Name
  Task: Change Font Type

Example of Performing a Task One or More Times

Of course it is also possible that a task is repeated a specific number of times. In this case, a numeric count, or a range of counts, is applied to the bracket. For example, imagine three new boxes of diskettes and the task FORMAT DISKETTES. There are ten diskettes to a box, so the task is described as:

DO 3
- DO 10
  Task: Insert Diskette
  Task: Format Diskette
  Task: Name Diskette
  Task: Name Box of Diskettes

Example of Performing Specific Repetitions of Tasks

This example also shows how task groupings can be nested.

- Conditional looping of tasks

The previous notation is used when the number of repetitions is known, however, in many cases the number of repetitions is controlled by a condition. Conditional task repetition can be represented by two different task description structures: a test-before loop and a test-after loop (much like computer programming looping constructs). A test-before looping construct describes tasks that are performed only if a condition is evaluated first, whereas, a test-after looping construct describes tasks that are performed
first and then the condition is evaluated. These task description structures differ from the previous notations in that the repetition is conditionally controlled.

A test-before loop structure that TMM uses in task description is ‘DO-WHILE’. Natural language statements represent the conditions associated with the looping structure. The conditions are evaluated by users during task performance, and the user determines if the tasks are to be executed again. For example, if the task is to update files the description could be:

```
[DO WHILE more <File_Icon> to Update
 Task: Locate next file, <File_Icon>
 Task: Update <File_Icon>
]
```

**Example of Test-Before Looping of Tasks**

The test-after loop description structure used by TMM is ‘DO-UNTIL’. Like the test-before loop condition, the test-after loop condition is specified in natural language which the user should be able to evaluate. One difference between the test-before loop and the test-after loop is that the tasks will be performed at least once in a test-after loop. Our previous example could be restated with a test-after loop, if and only if, there is an assumption that there is always at least one <File_Icon> to update:

```
[DO
 Task: Locate next file, <File_Icon>
 Task: Update <File_Icon>
 UNTIL no more <File_Icon> to Update
]
```

**Example of Test-After Looping of Tasks**

Notice in both of these examples, the test-before and test-after looping constructs, that the conditions are evaluated by the user. The user determines if the loop is executed or re-executed. This implies that the user must know the condition and how to evaluate it. These conditions are included in procedural knowledge.

The previous examples all involve sequential tasks, but many higher-level tasks provide time independent sub-tasks where the selection order does not matter. In this case, the task is composed of disjoint sub-tasks—non-concurrent tasks that are not ordered in time.

For example, if the task were SET VCR CLOCK TO WEDNESDAY 7:21PM, then the sub-tasks represent setting the hour, minute, and weekday. However, their execution order is not important—the user may set the weekday before setting the hour, or whatever order is desired. If the order of sub-task execution is not important (or left to user discretion) then the ‘&’ operator is used:

---

1 While natural language (English, or other human-spoken language) is used to specify conditions, an imposed syntax would be helpful. This syntax is left to the interface specialist.
Task: Set VCR Weekday to Wednesday
Task: Set VCR Hour to 7 pm
Task: Set VCR Minute to 21

Example of Non-Sequential, Non-Concurrent Tasks

Alternative method selection

Many higher-level tasks may be performed by any one of several different methods. Each method should be captured within a TMM’s task description. TMM’s notation for alternative method selection is ‘I’ or ‘OR’. As an example, consider the task DUPLICATE FILE in the Macintosh operating system. One of two duplication alternatives can be used, either the ‘⌘D’ short-cut or the ‘DUPLICATE’FILE-menu item is selected:

Notation Alternative 1:
- Task: Select File
- Task: Key ⌘D
- Task: Select File
- Task: Select ‘Duplicate’ item from ‘File’ menu

Notation Alternative 2:
- Task: Select File
- Task: Key ⌘D
- OR
- Task: Select ‘Duplicate’ item from ‘File’ menu

Example of Alternative Method Selection

In this example, it can be seen that the user must first select the file for duplication. After this selection, however, the user may duplicate the file in one of two ways, either through the ‘fast-key command access’ or through the ‘File’ menu. This example shows both methods, but it also demonstrates how the common prefix task SELECT FILE can be factored out with task grouping for readability and simplicity.

Concurrent tasks

There can also be concurrent tasks—performance of tasks is perceived to happen at the physical same time. These can be much more difficult to describe notationally, but they too must be captured and modeled. For example, consider the automotive task SHIFT FROM 3RD TO 4TH GEAR. To perform this task the user must first depress the clutch while releasing the accelerator concurrently. These two sub-tasks are marked as concurrent using the ‘||’ operator:

Task: Depress the Clutch
Task: Release the Accelerator

Example of Concurrent Tasks
Interleaved tasks

The final timing relationship involves interleaved tasks. In this case, a task might be suspended and then resumed after some intervening actions or tasks. Like concurrency, this is also very difficult to represent within a task description. Consider the tasks of writing and using a thesaurus—at any moment the task of writing could be suspended to look up a word, and then resumed. The notation for interleaved tasks is \( \Leftrightarrow \) as in the example:

\[
\begin{align*}
\text{Task: Write} & \quad \Leftrightarrow \quad \text{Task: Look up a word in Thesaurus}
\end{align*}
\]

Example of Interleaved Tasks

Combinations of timing operators

Of course, combinations and nesting of timing operators over task groups is possible. Let's return to the example task SET CRUISE CONTROL SPEED TO 55 MPH. Consider the following description:

\[
\begin{align*}
\text{Task: Turn Cruise Control on} & \quad || \quad \text{Task: Release Accelerator Pedal} \\
\text{Task: Scan Dash for Speed} & \quad + \quad \text{Task: Press Accelerator Pedal Down} \\
& \quad \text{Task: Set Cruise Control Speed}
\end{align*}
\]

Example of a Combination of Timing Relationships

This example has a double nested task timing relationship. The first action is to turn cruise control on. Then, at least once, the speed is assessed and the proper action is chosen among three alternatives. That example shows how timing relationships can be combined in a task described using TMM notation.

VI. TMM View of User Classes

The previous discussions assume that a single user (or single class of users) performs a given task. However, this is not always the case; there may be several distinct user classes for a given system or task. These various user classes can have different abilities and characteristics which are collectively referred to as user class profiles. For example, an interface may need to accommodate both novice and frequent users, yet the user class profiles are different which implies different approaches to the tasks and interfaces.
VI.1. Dealing with Different User Classes

A designer faced with a user population consisting of several distinct user classes must create a design that addresses the needs of all user classes. TMM incorporates support for different user classes into a design in several ways:

- switch among several interfaces depending on the user’s class,
- provide an intelligent interface that anticipates needs and addresses them, or
- create a single composite interface for all the user classes.

The first method, switch among several interfaces, is the simplest to design. In this case, several interfaces are created. TMM analysis is performed for each interface for each salient user task, and a specific user class profile is coupled with the corresponding interface for the analyses. Also, an analysis of the user task of switching from one interface to another may be important.

The second method, an intelligent interface, is far more complicated, and a complete discussion is beyond the scope of this guide. Basically, a meta-theory of user knowledge based on the different user classes is necessary for the interface to anticipate the users’ class and actual knowledge requirements.

The third and final method, a composite interface that provides support for all the various user classes, involves a TMM analysis of the interface with an intersection of all user class profiles. The intersection gives the analysts and designers a description of the “lowest-common users’” knowledge.

VI.2. User Class Profiles

Currently, this research does not provide a mechanism for generating user class profiles. However, research in knowledge elicitation in both cognitive psychology and artificial intelligence provides methods that can be used. There are several different methods that can be employed in generating the user class profile, they include: experimental methods to infer cognitive processes, interviewing/self-reporting techniques, repertory grids,, and verbal protocol. A critique of these methods can be found in (Evans, 1988).

Once a method is selected to gather user class profile information, it is important that this information be organized to aid TMM analyses. Mappings are analyzed by examining items being mapped and the domains in which they reside. Because of this, it is beneficial to isolate user profile information into groups identified by TMM domain-pairs (e.g., problem/computer semantic domain pair). This grouping will reduce the analysts time searching the profile definition.
VII. Using TMM in User Interface Design

The basic TMM analysis life cycle has three stages. The first stage generates TMM descriptions for user problematic tasks identified within formative evaluation. These candidate tasks are broken down hierarchically into sub-tasks. The task description is based on the task, sub-tasks, current interface (if any), and user observables (both user requirements and task performance). In stage two, the descriptions are analyzed with the user class profile(s) to produce a list of unsupported-knowledge elements. And in the final stage, a list of new user interface design requirements are synthesized from the unsupported-knowledge elements. If the interface is redesigned then this process can be repeated. The following figure represents the TMM process life cycle:

![TMM Analysis Life Cycle Diagram]

This section describes methods used within the TMM life cycle. Also, this section discusses metrics collectable over TMM descriptions and analyses.

VII.1. Generating TMM Task Descriptions (Methodology)

The first, and most important, step in performing any analysis with TMM is generating a task description. Task descriptions can be generated for tasks with or without existing computer interfaces—both of which are related to the design processes (initial design or redesign). It is wise to examine, through description and analysis, the most common user tasks and highly complex user tasks. Usability of an interface can be affected most when these classes of tasks are examined.

When generating a task description for a task that is not yet automated (a computer interface does not exist), it is important for analysts to have a good understanding of the problem domain. The analyst must examine the problem domain and user requirements, gleaning the most important (common and complex) tasks for description and analysis. After an initial design, further analysis, prototyping, and usability evaluation can drive the redesign process.
Generating task descriptions for already existing interfaces is more straightforward. Qualitative and quantitative formative evaluation results give analysts basic structure of user tasks. From this analysis extrapolate domain items and mappings in order to create full task descriptions.

Following is a general outline for creating task descriptions using the TMM Task Description Form (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only one possible approach to generating TMM task descriptions.

---

**Generating Task Descriptions using TMM**

Select the task

1. Perform task decomposition to generate suitable candidates for task description.
2. Select candidate for description.
3. With a new TMM Task Description Form—decompose task within problem domain (if necessary) and for each of the sub-tasks perform steps 4 through 10.
Define the domain items

4. For the problem domain items define all related computer semantic domain items. Decompose tasks within computer semantic domain if necessary. Don't be concerned with their relationships at this point—just define all necessary items.

5. For the derived computer semantic domain items (step 4) define all related computer syntactic domain items. Decompose tasks within computer syntactic domain if necessary. Don't be concerned with their relationships at this point—just define all necessary items.

6. For the derived computer syntactic domain items (step 5) define input sequences and user articulatory actions within the articulation domain. Use LIAN (or any other articulation notation) in the articulation domain.

7. Iterate steps 4 through 6 until a comfortable degree of certainty is attained as to the completion of the domain item identifications.

Identify the mappings

8. For each domain item create mappings (arcs) to the corresponding adjacent domain items.

9. For each mapping list all the knowledge requirements.

10. Iterate steps 8 through 9 until a comfortable degree of certainty is attained as to the completion of the translation and mapping identifications.

Repeat process for the evaluation paths

11. Each system feedback creates an evaluation path. Perform the same process in 4-7 in reverse to capture the users' needs for each evaluation path. And then, perform the same process in 8-10 for each evaluation path.

Steps 8-11 are critical for the success of TMM analysis. These steps identify the mappings and translations among domain items. A finer task description granularity produces a better identification and analysis of user knowledge requirements.

Step 10 begs the question: What is 'A COMFORTABLE DEGREE OF CERTAINTY'? Here the analysts must examine the description's mappings and knowledge requirements determining if the they adequately capture of the underlying user needs. The analyst gathers task mappings and knowledge requirements by brainstorming, asking colleagues, and observing user performance. Analysts' experience will guide this process, failing that, observing users is the best method for capturing user needs. As with other task analysis methods, the TMM analyst is warned against analyzing arcane task methods—methods that very few users employ.

VII.2. Unsupported-Knowledge Analysis (Methodology)

At this point in the TMM analysis life-cycle, the analyst has a description of a task is within the TMM framework. This description is replete with mappings and necessary user knowledge requirements. An analysis is performed on each of the knowledge requirements within the mappings to ascertain whether the knowledge requirement is supported. User knowledge requirements are supported if:
- the user possess the knowledge (i.e., the user class profile identifies the knowledge as possessed or understood by the user class), or
- the interface models or gives reasonable access to this knowledge (i.e., the interface provides the knowledge in a symbolic/metaphor/iconic representation or as textual prompts).

If either of these two conditions hold, the knowledge requirement is supported, otherwise it is an unsupported user knowledge requirement. This stage of analysis focuses on generating a list of unsupported user needs in terms of knowledge. Generating this list is referred to as an unsupported knowledge analysis.

Following is a general outline for generating an unsupported knowledge list using the completed TMM Task Description Form and filling in the TMM Identified Knowledge Element Form (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only one possible approach to analyzing a task description for unsupported knowledge requirements.

---

**Performing a TMM Knowledge Analysis**

1. Generate a task description for the task in question.
2. Gather user class profile(s).
3. From mappings within the task description, compile a list of all knowledge requirements separated into three lists: factual, conceptual,
and procedural knowledge. Use the TMM Identified Knowledge requirements Form.

4. For each of the three lists (from step 3) do steps 5-9.
5. For each knowledge requirement on the list do steps 6-9.
6. Determine if user possesses the knowledge requirement though comparison of the knowledge requirement and user class profile.
   - If the user has the knowledge then mark the knowledge requirement as a supported knowledge requirement and proceed to next item on the list (goto step 5).
   - If the user does not possess this knowledge requirement continue (goto step 7).
7. Determine if interface adequately provides access to the necessary knowledge.
   - If the interface provides the knowledge then mark the knowledge requirement as a supported knowledge requirement and proceed to next item on the list (goto step 5).
   - If the interface does not provide this knowledge continue (goto step 8).
8. At this point, the knowledge requirement is unknown to the user and the interface does not provide access to necessary information. Therefore, that knowledge requirement is marked as an unsupported knowledge requirement.

Determining if knowledge requirements are supported, steps 6 and 7, is the crux of this analysis stage. Often there are ambiguities as to knowledge support, and in these cases, the analysts' experience and knowledge play an integral part. For example, consider an interface where a knowledge element is supported from the analysts' view (it is displayed), but not from the user's view (no logical connection can be made). However in this case, the results of usability testing would indicate the flaw in the designers' assumption of knowledge requirement support.

VII.3. Interface Design Support Analysis (Methodology)

Finally, once a TMM task description and a list of unsupported knowledge requirements are generated, a final analysis turns this information into design requirements. Following is a general outline for generating new user interface design requirements using the TMM Translation Analysis Form (see Analysis Templates section). This does not represent a hard-and-fast dictum, but only a possible approach.
TMM Synthesis of Design Requirements

Translate
Unsupported-Knowledge
Requirement
Into New User Interface
Design Requirements

⇒ Repeat for each Unsupported Knowledge Requirement ⇒

Third Stage in TMM Analysis Lifecycle

Interface Design Support Analysis

1. Perform a unsupported knowledge analysis for the task in question.
2. For each unsupported knowledge requirements identified in step 1, do steps 3-8.
3. Transfer the unsupported knowledge requirement(s), mappings, and associated related domain items to a new TMM Translation Analysis Form.
4. With relation to the unsupported knowledge requirement, examine the current interface (if appropriate) and distinguish and record related characteristics.
5. With relation to the unsupported-knowledge element, examine the user class profile(s) and distinguish and record related deductions and inferences.
6. With regards to steps 4 and 5, define the user interface design requirement(s).
7. (Optional) Translate the User Interface design requirement into an actual user interface design specification.

The quality of this analysis relies on the abilities and experience of analysts and interface designers—steps 5-7 in particular. However, the following templates of user interface design requirements can aid analysts and designers when faced with unsupported knowledge elements and user needs.
Supporting Factual Knowledge

An example user interface design requirement for an unsupported factual knowledge, `<STATEMENT>`, could be:

**UIDR:** COMMUNICATE `<STATEMENT>` TO USER
(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

For example, suppose a factual knowledge requirement of *third column is debt*. If this requirement is unsupported, then both the user must be unaware and the interface doesn't give any cue that the third column represents debt. This requirement is cast into a user interface design requirement like:

**UIDR:** COMMUNICATE `<THIRD COLUMN IS DEBT>` TO USER
(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

Of course, the user interface design specification could outline an interface solution for this requirement. Within interfaces, requirements for factual knowledge can be dealt with by displaying the knowledge requirement either textually or graphically, and for a short duration or continuous. So a possible user interface design specification could be:

**UIDS:** ALL DISPLAYED COLUMNAR DATA IS TO BE CLEARLY LABELED.

Supporting Conceptual Knowledge

With conceptual knowledge, `<CONCEPT>`, the UIDR could be:

**UIDR:** PROVIDE UNDERSTANDING OF `<CONCEPT>` TO USER
(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

For example, Apple Macintosh-OS helps users conceptualize files by using icons. These file representations can be used with other icons that represent actions (e.g., trashcan represents deletion). The use of icons and software widgets to support conceptual information is not new; however, it is very complicated.

For example, consider the user conceptual knowledge requirement of *chemical process temperature control* within an interface that controls some chemical reaction. Temperature is a common physical phenomena; however, in this case the temperature has direct bearing on a chemical process, where too hot or too cold can corrupt the process. If the user class profile does not indicate that the user class is intimately familiar with the ranges, settings, and outcomes based on temperature, and the interface is poor at conveying this information (found through usability testing)—then a UIDR could be cast as:

**UIDR:** PROVIDE UNDERSTANDING OF `<CHEMICAL PROCESS TEMPERATURE CONTROL>` TO USER
(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

The redesign associated with this UIDR is not as straightforward as that of the factual knowledge example. In this case, an entire concept must be conveyed. A user interface specification defines a possible solution:
UIDS: CONTINUOUSLY DISPLAY A Widget THAT ENCAPSULATES ALL NECESSARY OPERATOR INFORMATION BASED ON A 'SCALE' METAPHOR.

There are many possible design solutions for this interface specification, including:

Possible solutions to UNDERSTANDING OF <CHEMICAL PROCESS TEMPERATURE CONTROL>

In this example, the critical temperatures are identified for the user, as well as, buttons for changing the reaction temperature (either increasing or decreasing the temperature).

Supporting Procedural Knowledge

Procedural knowledge of actions or courses of actions could generate the following UIDR:

UIDR: IDENTIFY POSSIBLE PATHS OF EXECUTION TO USER (AT THE APPROPRIATE TIME IN TASK PERFORMANCE).

Informing users about possible courses of action is difficult. Sometimes performing a task requires the user be aware of several sub-tasks and their outcomes. Also, alternative task methods may be available to the user. Yet, with the increasing complexity of user interfaces and user tasks, identification of procedural knowledge becomes more important.

For example, consider a bank teller's task of DEPOSIT MONEY INTO BANK ACCOUNT using the bank's computer system. This task is made up of several different tasks: IDENTIFY CURRENCY TYPE (cash, check, or another account), LOCATE DEPOSIT ACCOUNT, PERFORM TRANSACTION, and VERIFY TRANSACTION. These tasks may also have sub-tasks (e.g., identify currency type for a check may involve verifying the check against an account, etc.) So the procedural knowledge necessary for the user (bank teller) would generate the following UIDR:

UIDR: INFORM THE USERS THAT THE <DEPOSIT MONEY INTO BANK ACCOUNT> TASK REQUIRES THE FOLLOWING TASKS TO BE PERFORMED: IDENTIFY CURRENCY TYPE (CASH, CHECK, OR ANOTHER ACCOUNT), LOCATE DEPOSIT ACCOUNT, PERFORM TRANSACTION, AND VERIFY TRANSACTION.
Each sub-task would also have a UIDR associated with it to outline the tasks required. However, for this example we will focus on this UIDR.

The question comes up, how can this UIDR be converted into an UIDS or user interface design solution. Specifications that try to communicate procedural knowledge to users include: task tree (a flow diagram of possible courses of actions), user prompting, etc. In our example the UIDS could be:

**UIDS:** PROVIDE A TASK-TREE TO THE USER IDENTIFYING SUB-TASKS AND ORDERING FOR THE TASK: DEPOSIT MONEY INTO BANK ACCOUNT.

with the possible solution:

![Possible task-tree solution for the task: Deposit money into bank account](image)

In this example, a task-tree is implemented so that users (bank tellers) know at all times their location within the task (identified here as rectangular border). Users also know what they have done and what is left to do. For this example the tasks are expanded as necessary to show possible alternative methods and task timing. This can be very useful in training novice users, but may be an impediment to expert users.

**TMM and design requirements**

It is worth restating that TMM is used to synthesize new user interface design requirement, and not new interface designs. The examples provided here show the utility of transferring TMM design requirements into specifications and redesign; however, TMM does not directly support the designer's creative problem solving process.

**VII.4. TMM Metrics**

Another style of analysis that can be performed over TMM task descriptions and analyses is metrics collection. Metrics are measurements and functions of measurements that can indicate a strength or weakness within the measured object.
Currently two different metrics exist: **goodness of fit** and **task stacking**. TMM metrics are still growing and need validation.

**VII.4.i. Goodness of Fit**

The goodness of fit (GOF) metric measures the ‘fit’ of specific interface to a task. The ‘fit’ is defined as the ratio of number of supported knowledge requirements to the total number of knowledge requirements derived from TMM analysis. Logically, the goodness of fit metric is bounded between zero and one—worst and best respectively. A high ratio indicates that a large portion of the knowledge requirements were supported either by the users or the interface.

\[
\text{Number of Supported Knowledge Requirements} \div \left( \text{Number of Unsupported Knowledge Requirements} + \text{Number of Supported Knowledge Requirements} \right)
\]

**Goodness of Fit Metric**

We feel that these metrics can provide an empirical way to compare different tasks over alternate interface designs; however, currently this metric needs validation.

**VII.4.ii. Task Stacking**

Another metric that can be gathered over a TMM task description is task stacking. Task stacking is the user process of suspending the current task, performing another task to completion, and then returning to the previous task (that had been stacked). The user must maintain a stack of the current tasks that s/he is performing and that can be counted by examination of a TMM task description. Task stacking has been explored in other models to determine user cognitive workload, and our approach is similar. (Card, Moran, & Newell, 1983; Kieras & Polson, 1985; Lewis & Polson, 1992)

While it is useful to count the number of suspended tasks at any task level, this method does not attempt to access the complexity of each task stacked. Therefore, this metric can only serve to indicate the number of suspended tasks, but not the overall effect of suspending a particular task. If, however, you are examining short duration, low complexity, sequences of tasks then this metric can help to identify location of cognitive overload—where there are too many tasks stacked.

**VII.5. TMM as a Teaching/Documentation Device**

TMM task descriptions can also be used for documentation and teaching purposes. TMM is based on a domain structure where natural language specifies items, also the clarity of UAN at the articulation level helps. (Hartson, et al., 1990) The straightforward nature of the descriptions lends itself to non-expert interpretation, and thus, a description can, to a degree, be understood by users. If users can follow a task description and identify necessary knowledge, then their training needs are met.

In the arena of teaching, TMM task descriptions can support, through guidance, task performance. TMM descriptions outline necessary information that users must have—hence, users only need to parrot the actions within TMM descriptions.
VIII. Common Questions

Will a tool be implemented to support TMM analyses?
Yes, eventually. For any theoretical endeavor to make its way to the practical world there must be some physical manifestation. Currently, there are preliminary thoughts towards creating a shareware C/X11 application that will provide ability to create and analyze task descriptions using the TMM framework and methods.

Should TMM be used for global analysis?
If time and money permitted than a full analysis could be done; however, in most cases the time to analyze an entire system is too great. We feel that a strength of TMM is its ability to be used in *situational analysis*—examination of a particular situation or task. This focuses the designer on real problems instead of allowing them to wander through other unfruitful analyses. Using situational analysis, TMM can be more easily incorporated into the iterative design process, during initial design as well as between formative evaluation and redesign.

Why all this work for design requirements?
Often, the interface design requirements gathered from users are incomplete and vague. TMM task modeling and analysis can help further define the requirements, and these interface design requirements are then used in interface specification and high level design. By providing a systematic method to derive new design requirements, as opposed to *ad hoc* methods, TMM can help reduce design time.

Does TMM tie the interface designers’ hands?
No. TMM modeling and analysis only derive interface design requirements and possibly interface design specifications. Using these requirements and specifications to create a usable interface is still the job of interface designers.

Should we always redesign based on TMM analyses?
As with most endeavors, the interface design process is limited by finances and time. Obviously, the goal of all interface designers is to produce a highly usable interface design, however, decisions must be made when confronted by deadlines. The decision to commit to a redesign, based on analysis, is a function of time, cost, and user impact. It is a context-dependent decision that cannot be addressed in this forum.
IX. Example

This section outlines an example of using TMM to derive/synthesize new design requirements. This example shows TMM analyses for a genealogy project prototype on a particular user task.

The first subsection outlines the prototype, while the second subsection provides a TMM situational analysis with analyst commentary. Each stage of the TMM analysis life-cycle is demonstrated, albeit condensed for space reasons.

X.1 Genealogy Project Description

Recently several users were asked to compile a list of requirements for a ‘simple’ genealogy interactive system. They produced three different types of requirements: data requirements, functional requirements, and interface requirements. Data requirements outline the project’s responsibilities with respect to data and data links (what to store and relationships among data). Functional requirements specify the basic functionality of the project—these form a minimum set of commands and actions available to users. Lastly, interface requirements were users’ requests concerning the project’s human-computer interface. An abbreviated list of these user defined requirements is presented to define characteristics of the genealogy project.

<table>
<thead>
<tr>
<th>User Defined Data Requirements</th>
<th>The interactive system must store information (personal datasets, PD) indexed by person—information includes, but is not limited to: Date, Time, and Place of Birth; Lifetime Achievements or Awards, short history, and any other free-format information. The interactive system must store relationships among PDs—these relationship links include: husband, wife, child, sibling, parent.</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>User Defined Functional Requirements</th>
<th>The interactive system must generate reports of the following formats (format given under separate cover). These formats are not important to this discussion, so they are not included. The interactive system must have the following functionality: Add/Remove/Edit a personal dataset Add/Remove/Edit a relationship link Ability to search the family tree Ability to traverse a tree ‘Cut-Away’ (section) Ability to get help</th>
</tr>
</thead>
</table>
The designer examined the three sets of requirements, performed a high-level task analysis and functional decomposition, created an essential (logical) model of the system, developed initial interface designs, and then developed a working prototype.

Functional decomposition and task analysis provides a list of important tasks that users commonly perform while using interactive systems. These tasks are generic in form, so they are translated into specific benchmark tasks for use during formative evaluation. A sub-set of tasks are chosen for this project design's formative evaluation, and they are:

<table>
<thead>
<tr>
<th>Generic Tasks</th>
<th>Benchmark Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task: Add a Personal Dataset to the database.</td>
<td>Task: Add George Alfred Mayo to the database.</td>
</tr>
<tr>
<td>Task: Change Relationship within a Personal Dataset. <em>(This includes the tasks of adding/removing children, spouses, and parents.)</em></td>
<td>Task: Change George Alfred Mayo's wife to Rebecca Rennick Johnson</td>
</tr>
<tr>
<td>Task: Change Relationship within a Personal Dataset. <em>(This includes the tasks of adding/removing children, spouses, and parents.)</em></td>
<td>Task: Add Gregory Thomas Mayo as a child of George Alfred Mayo</td>
</tr>
<tr>
<td>Task: Change Relationship within a Personal Dataset. <em>(This includes the tasks of adding/removing children, spouses, and parents.)</em></td>
<td>Task: Change George Alfred Mayo's mother to Inda Inskeep</td>
</tr>
<tr>
<td>Task: Traverse Tree Structure through Ancestor Chart</td>
<td>Task: Pull-up Ancestor Chart for George Alfred Mayo</td>
</tr>
<tr>
<td>Task: Traverse Tree Structure through Ancestor Chart</td>
<td>Task: Pull-up Ancestor Chart for Gregory Thomas Mayo</td>
</tr>
<tr>
<td>Task: Pull-up Personal Dataset for a given Person from the Ancestor Chart</td>
<td>Task: Pull-up Personal Dataset of Rebecca Rennick Johnson</td>
</tr>
</tbody>
</table>

A prototype was created to evaluate the interface design. This prototype was implemented in Hypercard™ and consisted of several (>10) 'cards' over various 'stacks'. There are several problems with this interface design. The problems range from simple consistency (e.g., card name position) to more detailed conceptual difficulties (e.g., how to navigate through the interface family tree structure). The following prototype screens show the basic user interface design.
This is the ‘start-up’ screen seen by users when the project software is launched. This provides a rough button-menu driven approach to system functionality. At this level the user may add/delete personal datasets from the database, search the database, or produce database reports. Help is also available for all commands.

This screen represents the general ‘name query’ sub-task common to many of the system’s functions. In this case, the query screen interacts with the user to gain a new name to add to the database. One of the users’ data requirements specifies the database be indexed by names, and this screen lists all names in the database helping users in name specification.

This screen shows the general form for displaying personal datasets, with an example dataset for Mayo, Gregory Thomas. The user may interact with this interface design to add/delete children, spouses, or parents. Any name may be selected to display the corresponding personal dataset (this functionality traverses the family tree structure). The user may also ‘pull-up’ an ancestor chart by selection the ‘Ancestor Chart’ button.

This final screen example shows an ancestor chart (card). The chart shown focuses on one person at a time—Mayo, Gregory Thomas—for this example screen. The chart displays all spouses, children, and siblings; there is also a partial display of father and mother lineage—only two levels deep (i.e., only to grandparents). There are two modes of searching for traversing ancestry charts, and name selection on this chart will invoke a search.

The formative evaluation on the prototype serves several functions, it (1) provides testing for functional completeness, (2) furnishes users access to working prototype for task analysis, and (3) grants users input into the user interface design process.
X.2 Situational Analysis of Genealogy Task

The discussion, to remain short, concentrates on a single problem that several novice users encountered while traversing the ancestor chart. To reiterate, the ancestor chart is traversed by name selection with two different search modes: Tree Trace (displays another ancestor chart for selected name), and Information Search (displays personal dataset for selected name). Users found it difficult to understand these modes, and to traverse the ancestor chart.

To explore this user problem, an analyst examined a task highlighting the use of search modes—the user must use the ancestor chart to locate a person and display the associated personal dataset. The benchmark task was: FROM THE ANCESTOR CARD (FOCUSED ON MAYO, GREGORY THOMAS) PULL-UP THE PERSONAL INFORMATION CARD FOR JOHNSON, REBECCA RENNICK.

The user starts the task at this screen—an ancestor chart focused on MAYO, GREGORY THOMAS.

This is the final screen. Getting to this screen indicates that the user successfully completed the task.

These two screens show the user’s dilemma. The user must interact with the interface in the first screen to get to the personal dataset shown in the second screen.

Following is an example analysis scenario with analyst/designer quotes in highlighted-italic text.

Analyst/Designer: “Okay, I’ve just finished some user testing, and the prototype interface has some problems. I will analyze the task: PULL-UP PERSONAL DATASET OF REBECCA RENNICK JOHNSON. I know the user’s performed two different methods to accomplish this task, either the user will traverse the ancestor card to find the necessary information, or the

---

1 Novice users are novice to the genealogy project software, not to direct manipulation computer system interfaces.
user will return to the main card and start a search from there. These are the representative task methods:

![Diagram of task decomposition]

**Example Figure 1: Task (Pull-up Personal Dataset of Rebecca Rennick Johnson) Decomposition**

Thinking through these two different methods, it becomes apparent that they represent two different styles of searches: the first method, if chosen, was accomplished easily with little error. In this case the user returned to the main screen and initiated a global search for Rebecca Rennick Johnson.

The second method, using the ancestor chart to traverse the family tree structure is based on a sub-task of locating a family member position with the family tree. This is a very difficult sub-task, and the location within the task that users had the most problems. See Example Figure 3 for a description of this sub-task. (Example Figures 3, 4, and 5 are at the end of this section due to their size.)

This task description gives an outline of the task Locate Family Member Position in Family Tree (LFMP-FT). In order to accomplish this task, the user must first examine the tree and change it's focus (possibly more than once). This represents the user traversing through the family tree within the 'Information Trace' search mode. Here the user must conceptualize personal pedigree charts as tree structures and that proper names index this tree structure. Decomposing this task further, to change the ancestor chart focus, the user must confirm the search mode and select a key name to search on. The knowledge required here includes information about the identifying and changing search modes, as well as, search outcomes based on these modes.

**Analyst/Designer:** "I feel pretty good about these descriptions. Of course, I realize that there are other ways a user might perform this task—but, these represent the methods that most users performed. I know (through user testing) users were having problems with search modes, but let's see what the analysis turns up."

Next the designer performed an Unsupported-Knowledge Analysis. Listing and examining the knowledge elements yielded several knowledge elements that were unsupported. (See Example Figure 4.)

**Analyst/Designer:** "Hmmm. The users complained about modes, and the TMM analysis showed modes were not supported! I wish I had done this analysis before paying for all that user testing! I hope I'm not fired for this."
The final analysis examined the unsupported-knowledge elements to synthesize design requirements—see Example Figure 5. This example uses the basic user interface design requirement (UIDR) template from the user's guide, and synthesizes four new UIDRs for the specified task. From these UIDRs the designers can create new interface design solutions.

In fact, from these UIDRs several different solutions can be posed, two are shown in Example Figure 2.

Example Figure 2: Possible Design Solutions

This scenario shows how TMM can be used in formative evaluation to support interface development. The new design requirements, once implemented, should go far to help support the user in this task. This example also shows how TMM can be used for situational analysis (analysis of a specific user problematic task) to derive interface design requirements.
TMM Task Description Form

Task: Locate Family Member Position in Family Tree

User Action | Feedback | Interface State
--- | --- | ---

Task Context/Notes:
Currently the screen displays an ancestor card. This task description is based on the user locating a name within ancestor card without returning to the main card.

Key:
- Fk: Procedural Knowledge
- Ck: Conceptual Knowledge
- K: Statements

Task: Locate Family Member Position in Family Tree
Designer(s): Joe Designer
Date: 12/31/95
Reviewed By: The Boss
# TMM Identified Knowledge Element Form

## Task/Design:
- Family Member Position in Family Tree
- Designer(s): Joe Designer
- Date: 12/25/95
- Reviewed By: The Boss

## Identified User Knowledge Needs

<table>
<thead>
<tr>
<th>Item</th>
<th>Notes</th>
<th>Supported</th>
<th>Partially Supported</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Family tree is indexed by person's proper name</td>
<td>This is supported by the user profile. A user defined data requirement is that the tree is indexed by the person's name.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two types of tree searching modes:</td>
<td>There is no indication as to what 'Tree Trace' or 'Information Trace' means. Also, there is no indication as to the number of searching modes. This information is not in the user profile.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Can search in only one mode</td>
<td>The interface does not provide this information, and it is not in the user profile.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Search mode is selected by toggling 'Model Buttons'</td>
<td>There is no indication that the mode button toggles between search types.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'Model Button' identifies search 'Model'</td>
<td>The mode button identifies the search type.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'Model' is 'Tree Trace'</td>
<td>The mode button identifies the search type.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>'Model' is 'Information Trace'</td>
<td>The mode buttons identify the search type.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Selecting a name field while in 'Information Trace' will generate a new response card with next-field form</td>
<td>There is no indication that selecting a field will generate any feedback based on search mode.</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Notes (e.g., justification of support or non-support)

- This information is in the user profile.

## Example Figure 4: Unsupported-Knowledge Element Analysis for LFMP-PT Task

Appendix A: TMM Analysis Manual - 177 -
TMM Transaction Analysis Form

Task/Design: 
Designer(s): 
Date: 
Reviewed By: 

FK: <Mode Button> identifies search <Mode>

FK: Selecting the <Mode Button> will toggle the search mode.

FK: Selecting a name field while modes='Information Search' will generate a new Ancestor Card with name field the focus.

User Articulation
Articulation Domain

User Needs
(Transformations/Mappings)

Users understand standard drag manipulation artifacts (windows, buttons, icons, pointing, etc.)

Inference: Currently employs a button that identifies the current search mode, and selecting (clicking on) the button will toggle between the two different search modes. There is no indication that the button will toggle among search modes, no indication as to the number of search modes, and there is no way that the user can get help.

Current Interface Characteristics

Design Requirement

UIDR: Communicate: <The two types of tree searching modes> to User (at the appropriate time in task performance).

UIDR: Communicate: <The tree can be searched in only mode> to User (at the appropriate time in task performance).

UIDR: Communicate: <The tree search mode can be easily identified, selected, or toggled with the <Mode Button> by User (at the appropriate time in task performance).

UIDR: Communicate: <The functionality of selecting a name field> to User (at the appropriate time in task performance).

User Interface Design Specification

Change current representation of tree search model selection to a more standard representation.

Provide help for the user.

Example Figure 5: UIDR Synthesis for EMP-FT Task.
Notes
—Notes—
Analysis Templates

This section provides several blank templates (Forms) for TMM users.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the author(s) copyright notice and the title of the publication and its data appear, and notice is given that copying is by permission of the author(s). To copy otherwise, or to republish, requires a fee and/or specific permission.
<table>
<thead>
<tr>
<th>Identified User Knowledge Needs</th>
<th>Notes (e.g., justification of support or non-support)</th>
<th>Supported by User Profile</th>
<th>Supported by Instructional Technology</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>Factual Knowledge Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptual Knowledge Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Procedural Knowledge Elements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Glossary

This section contains definitions of commonly used terms within this document and also within the task mapping model.

**Analysts**: A person concerned with the description and analysis of user tasks for performance and usability improvements of human-computer interfaces.

**Analysis Orientation Modeling Characteristic**: Models with this orientation generally attempt to analyze tasks and performance for currently existing interfaces. These analyses can be used within formative evaluation with limitations (because of size and complexity); these models are more suitable for final analysis within the summative evaluation phase.

**Articulation Domain**: The lowest level of abstraction in the domain structure of TMM. The actual physical user actions are recorded within this domain.

**Closed-Loop Modeling Characteristic**: Closed-loop modeling views tasks as cyclical involving a user action-feedback-reaction processes. Models that approach tasks in this manner have this characteristic.

**Computer Semantic Domain**: A domain of the TMM domain structure. This domain contains generic computer artifacts translated from the problem and computer syntactic domains.

**Computer Syntactic Domain**: A domain of the TMM domain structure. This domain contains abstract grammatical components for conveyance to the computer.

**Conceptual Knowledge**: The understanding of relationships between collections of factual knowledge, ideas, and other conceptual knowledge.

**Concurrent Tasks**: Tasks that are executed at the same time are concurrent.

**Design Support Modeling Characteristic**: This characteristic is represented in models whose results are directly applicable to designs or the design process. These models feed results directly into the iterative user interface design process.

**Domain**: A region populated with items characterized by a single feature. (E.g., all items are defined within the problem scope = problem domain.)

**Domain Items**: Any objects, operations, or sub-tasks defined within a specific domain.

**Error-Free Performance Modeling Characteristic**: Models with this characteristic can perform analysis with the assumption of error-free user behavior. This assumption is very common in prediction and interface models.

**Error Performance Modeling Characteristic**: Models that can analyze user behavior with errors have this characteristic. This characteristic is much more problematic for task analysis techniques, and requires an extended view of task performance and analysis.

**Essentially Analytical Modeling Characteristic**: Models that examine and manipulate representations of tasks and interfaces apart from empirical measurements/observations are analytical. This is a common characteristic among several models.

**Essentially Empirical Modeling Characteristic**: Models based on and incorporating empirical
Evidence have this characteristic. Empirical evidence could be verbal protocol, timing values, user satisfaction surveys, and critical incidents among others.

**Evaluation Path:** A sequence of translations and associated domain items from lower to higher levels of abstraction. These paths represent a reflective mode of the user, often spawned by system output.

**Execution Path:** A sequence of translations and associated domain items from higher to lower levels of abstraction. These paths represent the user taking higher level domain items into lower level items, often all the way to articulation.

**Expert Performance Modeling Characteristic:** The ability to model and analyze expert behavior and performance is represented by this characteristic. Many models have this characteristic.

**Factual Knowledge:** Single or collections of declarative facts; something said to be true.

**Global Modeling Characteristic:** This characteristic is present in models that analyze entire representations of the interface and task. Performance models are often of this type because the predictions need the entire system described to be accurate.

**Interface:** The collection of devices (hardware and software) that allows the communication between two different systems. (In this case, the communication is between humans and computers.)

**Interface Design:** The developed requirements and specifications of an interface.

**Interface Designer:** A person who performs the necessary chores for interface design. This could, under loose interpretation, also include implementers of the interface design.

**Interface Design Requirement:** A design statement of a goal that must be present in the interface. E.g., The interface must be Motif compliant.

**Interface Design Specification:** A design requirement translated into hardware or software terminology. E.g., There will be four menus: File, Edit, View, and Special.

**Interleaved Tasks:** A set of tasks of which only one can execute at a time, but they may be suspended and resumed at anytime and in any order.

**Knowledge:** Facts, concepts, and procedural knowledge. Also see conceptual knowledge, factual knowledge, and procedural knowledge.

**Open-Loop Modeling Characteristic:** Open-loop modeling views a task as performing actions to a desired result without benefit of feedback, or that feedback is irrelevant. Models with expert behaviors assumptions often ignore feedback.

**Performance Prediction Modeling Characteristic:** Models that calculate performance predictions are most commonly used in summative evaluation for comparisons against usability specifications. A complete (or greatly extended) design is necessary for these models to approach accuracy.

**Problem Domain:** The highest abstracted domain of the TMM domain structure. This domain contains items defined in, and only in, the user task (problem) domain.

**Procedural Knowledge:** Knowledge of a particular action, or course of action.

**Mapping:** A knowledge element crucial to a translation. One or more mappings constitute a translation.

**Method:** A plan or course of action.

**Method Selection:** The determination of which method from a list of alternative methods.

**Methodology:** A system of methods, principles and rules.

**Metric:** A measurement taken on an object. In this case, measurements taken to increase usability or performance from a software interface.

**Multiple User Classes Modeling Characteristic:** Users are no homogeneous. Methods that allow analysis and modeling of multiple user classes per task have this characteristic. Models that
center on expert error-free performance often cannot have this important characteristic.

Non-Sequential Non-Concurrent Tasks: Tasks that are not ordered in time and each task is atomic (non-divisible) in nature.

Novice Performance Modeling Characteristic: This characteristic is present in models that can model and analyze novice performance. Novice users comprise a growing class of users that must be addressed, yet, many models only partially support them.

Repeated Tasks: A set of tasks that are repeated zero or more times.

Sequential Tasks: A set of tasks where the ordering is linearly based on time.

Situational Modeling Characteristic: Models that allow analysis of single tasks or interface features are situational and have this characteristic. Often these models are employed during formative evaluation due to their utility.

Sub-task: A task subordinate to another task. Its completion or failure impacts the primary task(s).

Synthesis Orientation Modeling Characteristic: Models that derive new designs or requirements are synthesis oriented. The new design or requirements can be incorporated into a design revision within the iterative design process, and thus, aid the designer.

Task: “A task is an arrangement of behaviors (perceptual, cognitive, motor) related to each other in time and organized to satisfy both an immediate and a longer-range purpose.” (Meister, 1985)

Task Abstraction: To examine and redefine a task form and function away from the actual manifestation; to reconceptualize in high-level ideas and concepts.

Task Analysis: “Task analysis is a process of identifying and describing units of work, and analyzing the resources necessary for successful work performance. Resources in this context are both those brought by the worker (e.g., skills, knowledge, physical capabilities) and those which may be provided in the work environment (e.g., controls, displays, tools, procedures/aid).” (Drury, Paramore, van Cott, Grey, & Corlett, 1987)

Task Description: To examine and reiterate a task form and function within a pre-defined task description notation.

Task Mapping Model: A task/user-centered approach to synthesizing user interface design requirements from task analysis.

TMM: Task Mapping Model.

TMM Identified Knowledge Need Form: Template of a form that helps designers examine task descriptions to produce a list of unsupported knowledge needs.

TMM Task Description Form: Template of a user interface task design form that is crafted to work with TMM.

TMM Translation Analysis Form: Template of a form that helps designers examine unsupported knowledge elements in order to produce user interface design requirements (UIDR).

UAN: See User Action Notation.

UIDR: User Interface Design Requirement. See design requirement.

User: Any person that performs a task on a computer.

User Action Notation: A symbology and notation for describing user physical actions during human-computer interaction. (Hartson, et al., 1990)

User Class: A pre-defined set of similar users.

User Class Population: The set of all salient user classes defined for a particular application.

User Class Profile: A description of knowledge, skills, and attributes of a particular user class.

User Knowledge Modeling Characteristic: Models that have this characteristic attempt to capture necessary user knowledge. This differs from user mental modeling in that this characteristic does not in any way represent the mental state of user, but only necessary user knowledge.
User Mental Modeling Characteristic: Models that rely on a basis user mental model have this characteristic. A user mental model, as this research defines it, is a collection of user mental states, translations among the states, and cognitive operators to effect the state changes. This could include a representation of an interface or user knowledge within a context. Interface modeling techniques have an underlying user mental model of the system (interface).
References


APPENDIX B: TMM TUTORIAL

The following guide is a reprint of:


It was given to the TMM analysts in phase two of the TMM evaluation experiment.
Situational Analysis with the Task Mapping Model (TMM)

— A Tutorial —

Version 1.0

Kevin A. Mayo
Department of Computer Science, Virginia Tech (VPI&SU)
Blacksburg, VA 24060
mayo@cs.vt.edu

I Introduction
1.1 The Task Mapping Model (TMM)
1.2 User Interface Development Process

II Where to begin?
II.1 An Example

III Describing Tasks
III.1 TMM Domains
III.2 Finding Domain Items
III.3 TMM Mappings, Paths, and Knowledge Requirements
III.4 Measure twice, cut once

IV Analyzing Task Descriptions
IV.1 User Class Profiles
IV.2 Supported vs. Unsupported Knowledge Requirements
IV.3 Example Analysis

V Synthesizing New User Interface Design Requirements
V.1 Templates
V.2 Alternative Method: The Holistic Approach

VI What now?
VI.1 How do UIDRs Help the User Interface Design Process?

VII Discussion

Appendix A: Domain Language
Appendix B: Analysis Forms
Acknowledgments
References

Appendix B: TMM Tutorial
1 Introduction

"The product is nothing; the process is everything."

(Gause & Weinberg, 1989, p. xvi)

1.1 The Task Mapping Model (TMM)

The Task Mapping Model (TMM) is a user-centered analysis technique providing methods and notations for use in human-computer interaction (HCI) task analysis (Mayo & Hartson, 1993). The primary function of the TMM is to synthesize new design requirements based on usability problems, user task descriptions, user class profiles, user knowledge requirements, and current user interface designs. The TMM does not, however, derive new design specifications or high/low level interface designs.

This document is meant to jump-start the reader in using the TMM—i.e., this is a tutorial. The reader should have the Task Mapping Model Analysis Manual (Mayo, 1993) on hand for further explanation of terms, concepts, and methodologies.

1.2 User Interface Development Process

The user interface development process is an iterative process cycling between design, implementation or prototyping, and evaluation—see Figure 1. User interface specialists iterate within this cycle to develop usable systems. Usability, in this context, is a function of user satisfaction and efficiency of interactions.

This process begins with a Functional Analysis and Task Analysis for the system being designed. The functional analysis defines system capabilities in terms of objects and actions. For example, a Rolodex system's actions include SEARCH, PRINT, and SORT, while the objects are data associated with each persons' entry. Here the functional analysis defines the functional capabilities of the system—not what the user will do, but what the user can do. Task analysis identifies and analyzes representative tasks that users will do. Task analysis does not necessarily analyze all possible tasks—just meaningful ones. In terms of what the user will do, task analysis determines needs associated with physical and mental execution of each task, i.e.:

---

Apple, Macintosh, and MultiFinder are registered trademarks of Apple Computer, Inc.
Microsoft and Windows are registered trademarks of Microsoft Corporation, U.S. Patent No. 4,974,159.

1 By user interface specialists we mean: user interaction developer, user interface software developer, and the problem domain expert. Definitions of these roles and their associated duties can be found in (Hix & Hartson, 1993, pp. 8-11).
"Task analysis can be defined as the study of what an operator (or team of operators) is required to, in terms of actions and/or cognitive processes, to achieve a system goal. Task analysis methods can also document the information and control facilities used to carry out the task."

(Kirwan & Ainsworth, 1992, p. 1)

The functional and task analyses produce a set of initial user interface design requirements and/or specifications. Designers begin the transformation of this set of initial design requirements into an initial user interface design by defining user interface design specifications—"usually a formal set of statements about the design from which coding is to be done." (Hix, et al., 1993, p. 106) Specifications can range from high-level conceptualizations of the system to low-level description of user interface artifacts.

**Figure 1. Simplistic Representation of Iterative Nature of User Interface Design**

Next, user interface software developers implement (or prototype) the user interface design as specified. It is important to recognize the different roles that user interface
interaction specialists (who define the interaction and the interface design) and user interface software developers (who code the design) play in development.

The implementation is evaluated to identify usability problems. This is analogous to the acceptance-testing phase in standard software engineering product life cycles; however, the evaluation (or testing) of user interface designs takes into account user satisfaction (e.g., functional completeness, high degree of usability, efficient interactions, refer to (Chin, Diehl, & Norman, 1988)) and usability specifications (i.e., expected timings of task performance, refer to (Whiteside, Bennett, & Holtzblatt, 1988)).

At this point a set of usability problems have been identified during formative evaluation of the user interface design. Solutions to these usability problems must be incorporated back into the design. However, there is a methodological gap between identifying the usability problems and generating new interface design requirements for the following round of re-design, implementation, and evaluation. The TMM addresses this gap—see Figure 2.

We believe the translation of usability problems into new design requirements is an essential, and to-date unsupported, component of user interface development. The TMM takes usability problems, user tasks, user class profiles, and current user interface designs and synthesizes new user interface design requirements, thus supporting the translation. Specifically, the TMM is a three phase process: (1) TMM Task Description, (2) TMM Knowledge Analysis, and (3) TMM Synthesis of Design Requirements—see Figure 3. First, user interface specialists identify and select candidate tasks for analysis by examining results from formative evaluation. The candidate tasks are described using the TMM task description framework. In phase two, the TMM task description is analyzed for user knowledge requirements for task performance, and the knowledge requirements are further classified as either supported or unsupported within the task performance environment. In the final phase, unsupported user knowledge requirements are recast as new user interface design requirements for use in future redesigns.

---

2 Of course, this simplistic statement reflects the theory of what to do and not the reality of practice. Often usability problems are examined with respect to its impact on overall usability (i.e., impact analysis) and its estimated cost to fix. Only cost-effective solutions and re-designs are attempted.
Figure 2. Representation of User Interface Design with TMM Analyses in Context
The TMM focuses analyses on user tasks that formative evaluation has identified to have usability problems. This style of analysis, *situational analysis*, is conducive to the iterative nature of user interface development. By only modeling tasks with usability problems, the TMM helps increase efficiency by reducing analysis time.

II Where to begin?

"The trickiest part of certain problems is just recognizing their existence."

(Gause & Weinberg, 1990, p. 53)

Where to begin? is often a difficult question. In our context, the question of where to begin is really the question: *What tasks require analysis?* Luckily, this question is answered for user interface specialists within the evaluation phase. Specifically, empirical results from formative evaluation defines and isolates user interface usability problems. The TMM, for efficiency purposes, uses these results to drive the analyses—*situational analysis* of usability problems and tasks.

This tutorial focuses on using the TMM as a tool for re-design. This implies that at least one round of formative evaluation has taken place. It should be noted, however, that the
TMM can be used for initial design, but that ability of the TMM is not illustrated within this tutorial.

This section introduces a simple example that is used to facilitate discussion about the TMM.

II.1 An Example

Consider, as an example, a DICTIONARY APPLICATION. This application provides users the means to reference words and meanings from multiple dictionaries. Following is a short prose software requirement, a hierarchical functional decomposition, function definitions, and some representative tasks. A task analysis of specific tasks is provided in a following section.

<table>
<thead>
<tr>
<th>Software Requirement (Prose)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design an interactive Dictionary system that will automate finding words and their meanings across multiple, independent dictionaries. It is intended that this application will execute in the background.</td>
</tr>
</tbody>
</table>

This interactive system runs in the background, that is, the system can be invoked within other contexts (e.g., wordprocessors, mail systems). The system allows the user to specify dictionaries to use—multiple dictionaries are available for specific terminology domains (e.g., medicine, law, building standards).

The system performs the following system functions (SF) and supports the following generic user tasks (UT):

<table>
<thead>
<tr>
<th>SF: Check Spelling</th>
<th>This function correlates search criteria spelling against the selected dictionary(ies)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SF: Locate Word</td>
<td>This function locates a word in a dictionary, but also allows user to check spelling, selected different dictionary(ies), or customize a dictionary (i.e., add a dictionary term).</td>
</tr>
<tr>
<td>UT: Chose Dictionary</td>
<td>Allow user to view and select among several dictionaries.</td>
</tr>
<tr>
<td>UT: Chose Search Criteria</td>
<td>Allow user to enter/modify search criteria.</td>
</tr>
<tr>
<td>UT: Customize Dictionary</td>
<td>Allow user to change (add/delete/modify) terms within a dictionary.</td>
</tr>
</tbody>
</table>

**Table 1. System Functionality and Generic User Tasks**

Due to space considerations, this tutorial does not analyze system functions or user tasks identified by italics.

Consider the following representative tasks:

- SELECT A DICTIONARY AND LOCATE THE WORD AND MEANING: PHENYRAMIN
- SELECT A DICTIONARY AND LOCATE THE WORD AND MEANING: PHENOXINE
- SELECT A DICTIONARY AND LOCATE THE WORD AND MEANING: PSYCHOLOGY
- SELECT A DICTIONARY AND LOCATE THE WORD AND MEANING: PSYLOSIS
This tutorial illustrates the use of TMM as a situational analysis tool, therefore, an initial user interface design already exists. Consider the following initial user interface design:

![Diagram of dictionary interface]

**Figure 4. Initial User Interface Design for Dictionary System**

In this user interface design, the `<SEARCH COMMAND>` field will not accept input unless at least one dictionary is specified in the `<DICTIONARIES IN USE> field.

And finally, consider the following usability problem found during formative evaluation:

**Users were unable to perform satisfactorily (meet a usability specification)**

**The task:** Select a dictionary and locate the word and meaning: 'popinjay'.

This task is examined throughout our discussion. However, the example is purposefully kept short for space considerations.
III Describing Tasks

"As you wander along the weary path of problem definition, check back home once in a while to see if you haven’t lost your way."

(Gause, et al., 1990, p. 71)

“When the map and the territory don’t agree, always believe the territory.”

(Gause, et al., 1989, p. 9)

III.1 TMM Domains

Like other techniques, the TMM relies on task descriptions for its analyses. The more complete the description—the better the analyses. Task descriptions capture and reflect users performing tasks. In other words, a TMM task description outlines the steps that users take during task performance. Observations indicate there are many different abstraction levels in which users view tasks, e.g., high abstraction level (FIND MEANING OF ‘POPINJAY’ IN DICTIONARY SYSTEM) down to low abstraction level (TYPE ‘FIND’ WITH KEYBOARD). In order for users to perform these tasks, users must map the task from high to low levels of abstraction.

The TMM provides several levels in which tasks are described. We feel these levels more closely reflect the reality of users performing tasks. Using multiple levels of abstraction for description the TMM models users moving among these levels. For example, a user may need to perform the task USE THE DICTIONARY SYSTEM TO FIND MEANING OF THE WORD ‘POPINJAY’. Analysts can examine this task at a higher level, or perhaps may wish to examine the necessary articulation sequence, e.g., ENTER ‘FIND POPINJAY’ WITH KEYBOARD INTO <SEARCH-WORD> FIELD. Both of these views express the same task at different levels of abstraction. It is very important to analyze user performance at these various levels because user errors can occur at any of these levels.

In fact, there are more than two different domains in the TMM task descriptions. The user can conceptualize the task in other domains. Figure 5 illustrates the reconceptualization of the ‘USE’ task into the tasks: SELECT DICTIONARY DATABASE, and PERFORM SEARCH ON DICTIONARY DATABASE FOR <WORD>. The ‘PERFORM SEARCH’ task is further reconceptualized into a REFERENCE and EXECUTE pair of tasks and then into a series of keyboard typing tasks. (The SELECT DICTIONARY DATABASE task is not expanded due to space limitations.)
This figure shows how a task is conceptualized into various abstraction levels by users. Each of these abstraction levels is called a domain of TMM task description. Each domain represents a unique and disjoint abstraction level. The TMM has four domains: Problem, Computer Semantic, Computer Syntactic, and Articulation Domain. Each domain contains domain items—specifically these are objects, operations and sub-tasks defined within the domain's abstraction level.

The Problem Domain represents the users' real world task space, and as such, all of its domain items represent real world entities. E.g., the DOCUMENT PREPARATION problem domain includes items like: PARAGRAPH, WORD (objects); CUT, DELETE (actions); ADD TABLE OF CONTENTS, EDIT MANUSCRIPT (sub-tasks).

The Computer Semantic Domain contains computer representations of real world entities (from the problem domain). E.g., the DICTIONARY DATABASE in our running example is a computer representation of the problem domain DICTIONARY object.

The Computer Syntactic Domain differs from the computer semantic domain in that representations at this level use specific computer related interface entities with grammatical ordering of interactions. In other words, the computer semantic domain describes what to express to the computer, whereas the computer syntactic domain describes how to express to the computer. Thus, items in the computer syntactic domain are somewhat specific to general interaction styles and techniques. For example, the computer semantic domain task INDICATE DICTIONARY (what to express) is represented within the computer syntactic domain as RECOGNIZE <DICTIONARYFILENAME>, RECOGNIZE <SELECTCOMMANDNAME>, and EXECUTE SELECT DICTIONARY USING <DICTIONARYFILENAME> AND <SELECTCOMMANDNAME> (how to express).

Finally, the Articulation Domain is composed solely of user physical actions, hence it is the lowest level of abstraction. The TMM employs the User Action Notation (UAN, refer to (Siochi & Hartson, 1989; Hartson, Siochi, & Hix, 1990; Hartson & Gray, 1992)) to describe user and system behavior.

With these domain definitions our dictionary example is re-depicted in the following way:
The terminology used in the computer semantic and syntactic domains is derived from (Lenorovitz, Phillips, Ardrey, & Kloster, 1984) taxonomies of user interaction (both internal and external). These taxonomies are reproduced in Appendix A.

### III.2 Finding Domain Items

Finding domain items is a process of task decomposition and description. By decomposing a task into smaller and smaller sub-tasks analysts can create a task hierarchy (refer to HTA in Kirwan, et al., 1992). Sub-tasks within this hierarchy are then examined against domain definitions to determine placement within the TMM task description framework.

Each task identified by decomposition can be thought of as a action-object pair. Approaching tasks in this manner helps analysts model how actions and objects are modeled at the different abstraction levels (i.e., domains).

### III.3 TMM Mappings, Paths, and Knowledge Requirements

Decomposing a task and representing it at different levels leads to the observation that items can exist at several abstraction levels. E.g., the problem domain item Dictionary exists as Dictionary Database in the computer semantic domain, and also as <Dictionary Name> in the computer Syntactic domain. This existence begs the question: *What relationships exist among the various abstract incarnations of an item?* Answer: The relationships represents the users' reconceptualization of domain items across abstraction level boundaries. Analysts need to recognize and understand user reconceptualization, as well as, know *what is necessary* so that users can perform these reconceptualizations.

In the TMM task description framework, *mappings* identify users' reconceptualizations of items across various abstraction views. We believe that most usability problems are because of users' inability to perform these mappings. The following figure shows our example with mappings depicted as black arrows.
The mappings (arrows) form a path that models the users' performance of the task. However, this example is incomplete because it ignores that users perform tasks cyclically based on execution and evaluation. Users execute tasks and then evaluate their progress based on system feedback. (Refer to Norman's Theory of Action (Norman, 1986; Hutchins, Hollan, & Norman, 1986).) The TMM examines both user execution and evaluation paths. A path that moves from higher to lower abstraction represents an user execution path, while a path from lower to higher abstraction represents an evaluation path. Expanding our example, Figure 8 illustrates how TMM models both paths.

At this point, our task description depicts the decomposed task and the necessary user reconceptualizations. This description begs the next question: What do users require to perform the reconceptualization (mapping) from one domain item to the next? For instance, what do users need to map the computer syntactic domain item Reference <Search> Command into the articulation domain item K"FIND". The user must know that the <Search> Command is actually "FIND", and the way to Reference it is by typing its name (K"FIND" in UAN). Each mapping, like this one, has user knowledge requirements that must be met in order for users to make or perform the mapping.
Describing knowledge is a very tricky (and messy!) process. The TMM uses simplified natural language (English) to represent user knowledge requirements. The TMM only identifies necessary user knowledge; TMM does not represent any mental processing of knowledge (e.g., remembering, learning). There are three categories of knowledge used in the TMM analyses: **factual knowledge**, **conceptual knowledge**, and **procedural knowledge**.

**Factual Knowledge** is comprised of specific facts. E.g., DOCUMENTS are contained in FILES, the `<SEARCH>` command is "FIND".

**Conceptual Knowledge** are collections of knowledge that comprises a whole understanding or meaning. In other words, conceptual knowledge is the understanding of relationships among collections of factual knowledge, ideas, and other conceptual knowledge. E.g., MANUSCRIPT EDITING STANDARDS, FILES, GENERAL DICTIONARY USAGE AND TERMINOLOGY.

**Procedural Knowledge** represents possible courses of actions, alternative methods, and task ordering and structure. E.g., SELECTION IS VIA SINGLE-CLICK, there are TWO OUTCOMES TO SITUATION XYZZY.

Using these three knowledge categories, we re-examine our task description and associate user knowledge requirements with each mapping. Identifying knowledge requirements is often a process of examining mappings and their domain items from the user's perspective. Try answering the following example questions: What do I (the user) need to know to reconceptualize A into A'? What are the relationships between the components of A and A'? What are the possible outcomes of my current task or sub-task?

Returning to our example task, in Figure 9 each mapping (identified by number) is associated with knowledge requirements (outlined in the following table).

![Diagram of knowledge requirements and task mapping](image)

**Figure 9. Example Task with Knowledge Requirements**
<table>
<thead>
<tr>
<th>Mapping</th>
<th>Knowledge Type</th>
<th>Knowledge Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Procedural</td>
<td>A dictionary must be chosen before the search</td>
</tr>
<tr>
<td></td>
<td>Factual</td>
<td>Dictionaries are alphabetically indexed by words</td>
</tr>
<tr>
<td></td>
<td>Conceptual</td>
<td>Database Functions and Structures</td>
</tr>
<tr>
<td>2</td>
<td>Factual</td>
<td>Search criteria, <code>&lt;WORD&gt;</code>, is POPINJAY</td>
</tr>
<tr>
<td>3</td>
<td>Factual</td>
<td>Databases can be searched</td>
</tr>
<tr>
<td></td>
<td>Factual</td>
<td>A command, <code>&lt;SEARCH&gt;</code>, exists to locate data in database</td>
</tr>
<tr>
<td>4</td>
<td>Procedural</td>
<td>The search criteria must be specified before the search</td>
</tr>
<tr>
<td>5</td>
<td>Procedural</td>
<td>Explicit action is necessary to invoke search command</td>
</tr>
<tr>
<td>6</td>
<td>Procedural</td>
<td>Input of <code>&lt;SEARCH&gt;</code> command name is by keyboard entry</td>
</tr>
<tr>
<td></td>
<td>Factual</td>
<td>FIND is the <code>&lt;SEARCH&gt;</code> command</td>
</tr>
<tr>
<td>7</td>
<td>Procedural</td>
<td>Input of search criteria is by keyboard entry</td>
</tr>
<tr>
<td></td>
<td>Factual</td>
<td>Search criteria, <code>&lt;WORD&gt;</code>, is POPINJAY</td>
</tr>
<tr>
<td>8</td>
<td>Procedural</td>
<td><code>&lt;CR&gt;</code> Terminates input and executes command</td>
</tr>
<tr>
<td></td>
<td>Procedural</td>
<td>Input of <code>&lt;CR&gt;</code> is by keyboard entry</td>
</tr>
<tr>
<td>9</td>
<td>Factual</td>
<td>Search results are presented in <code>&lt;RESULTS&gt;</code> window</td>
</tr>
<tr>
<td></td>
<td>Conceptual</td>
<td>Format of search results</td>
</tr>
<tr>
<td>10</td>
<td>Procedural</td>
<td>When desired results are not located, the task is repeated</td>
</tr>
<tr>
<td>11</td>
<td>Procedural</td>
<td>When desired results are located, the task is completed</td>
</tr>
</tbody>
</table>

Table 2. User Knowledge Requirements X-Referenced with Mappings

A full TMM task description includes the decomposition of the task into TMM abstraction domains, the mappings, and the user knowledge requirements. This description, which is as complete as space allows, is representative of a TMM task description.

### 3.4 Measure twice, cut once

Task description quality dictates quality of both the analyses and synthesized user interface design requirements. Check over all descriptions. In fact, have a friend/colleague check the descriptions. Walkthrough and talk-through formats are highly recommended to check accuracy and completion. (Refer to (Freedman & Weinberg, 1990)).

Space limitations prohibit discussion of several topics concerned with task descriptions within the TMM framework. These topics include decomposition techniques and task timing issues. These topics are covered in more depth in (Mayo, 1993), and more motivation for the TMM can be found in (Mayo, et al., 1993).
IV Analyzing Task Descriptions

“All development starts with the assumption that a solution to a problem can exist.”

(Gause, et al., 1989, p. 55)

Once an adequate task description is generated for a problematic task, analyses generate a total set of user knowledge requirements for task performance. Analyzing the total set of user knowledge requirements yields two subsets of knowledge: supported, unsupported knowledge requirements. This process is explained in a subsequent section (IV.2), but first a user class profile must be generated.

IV.1 User Class Profiles

In order to perform knowledge analyses for our example, we need to know what knowledge needs are supported by users. Thus, it is imperative that we define our user class profile. User class profiles are demographic outlines of user groups that includes user physical and mental capabilities, as well as, what the user brings to the interactions in terms of knowledge (i.e., what the user is expected to know) and skills. (Refer to (Mayhew, 1992, Chapter 2).) For reasons that will become clear, user class profiles are necessary components of TMM knowledge analyses.

Generating user class profiles is a non-trivial process. There are infinite demographic dimensions by which users can be categorized and rated. It is contingent on analysts to select a set of meaningful discriminators to include within the profile. For example, is COLOR-BLINDNESS an important discriminate for an interface involving a BRAILLE SCREEN? (Not likely.) However, COLOR-BLINDNESS is can be very important for public INFORMATION ACCESS interfaces.

User knowledge must also be a component in user class profiles for TMM analyses. The knowledge is recorded using the same three categories already discussed (factual, conceptual, and procedural). Usually, however, there is only conceptual and procedural knowledge recorded within user class profile. Both of these knowledge categories can be thought of as short-hand compositions of factual knowledge, thus reducing the need to identify each piece of factual knowledge users possess. Also, it is nigh impossible to identify and record every piece of user factual knowledge.

There are several knowledge elicitation techniques that can gather user profile information along these categories. (Refer to (Evans, 1988).) However, again it is contingent on the analyst to gather and compose a meaningful user class profile.

Generating user class profiles are outside of the TMM’s scope of purpose. However, for purposes of discussion, assume the following user class profile holds for our example. In this example user class profile, each piece of information is preceded by a unique code which is used for traceability. Information from the user class profile is identified by its place within the hierarchical structure. For example, UCP. UD. 3 represents the USER
CLASS PROFILE. USER DEMOGRAPHICS: <THIRD ELEMENT>, i.e., SINGLE GEOGRAPHIC LOCATION (ANY TOWN, USA), and so on.

(UCP) User Class Profile

(UD) User Demographics:
UD.1: AGE: 15-45 YR.
UD.2: EQUAL SEX DISTRIBUTION
UD.3: SINGLE GEOGRAPHIC LOCATION (ANY TOWN, USA)

(UPL) User Physical Limitations:
UPL.1: STANDARD DISABILITY DISTRIBUTION FOR USA.

(UML) User Mental Limitations:
UML.1: BELOW STANDARD DISABILITY DISTRIBUTION FOR USA.

(UKB) User Knowledge Base:
General Knowledge
UKB.1: ASSUME AT LEAST A 12TH GRADE EDUCATION

Problem Domain Knowledge
UKB.2: CK: DICTIONARY USAGE AND TERMINOLOGY

Computer (Semantic/Syntactic) Domain Knowledge
UKB.3: CK: BASIC DATABASE FUNCTIONS AND STRUCTURES
UKB.4: CK: DIRECT MANIPULATION/WIMP INTERFACES
UKB.5: PK: NOON-VERB COMMAND INTERACTION STYLES

IV.2 Supported vs. Unsupported Knowledge Requirements

User knowledge requirements are needs users have for task performance. In an ideal world users would know everything about the task and interface, in other words, users would have all the knowledge to meet the user knowledge requirements. In this case, we wouldn’t worry much about providing support. However, users are not omniscient, and more often than not, need assistance while doing tasks. The assistance users need represents unsupported user knowledge requirements.
The TMM task description framework associates user knowledge requirements with mappings. Therefore, analysts can generate a total set of user knowledge requirements by collecting requirements from all of the mappings (and reducing out duplicates).

As mentioned before, users are not omniscient, however, users do bring a certain amount of knowledge to the interaction. Knowledge that users possess is identified within the user class profiles.

As one would expect, the existing interface should also try to support user knowledge for task performance. Presumably this was accomplished within the existing interface design through metaphors, visual cues, prompting, etc. However, the existing design clearly does not support all the knowledge requirements, or we wouldn’t have found usability problems that lead us to do this analysis.

Removing what the user knows and what the interface already provides (and possibly what is provided by training) from the total set of knowledge requirements yields unsupported user knowledge requirements. This knowledge is not supported by the user nor by the interface.

The unsupported knowledge requirements create a barrier to user task performance. How can users execute tasks when they lack knowledge?

To recap, the process for identifying unsupported knowledge requirements is to gather all the knowledge requirements from the TMM task description mappings. Examine each knowledge requirement against the user class profile and the current interface (if one exists), and determine if it is supported or unsupported.

IV.3 Example Analysis

Returning to our dictionary example, all knowledge requirements are classified as either supported (by user class profile or interface) or unsupported. For example, consider the user procedural knowledge requirement associated with mapping 1:

```
(1) PK: A DICTIONARY MUST BE CHOSEN BEFORE THE SEARCH

KEY: (MAPPING NUMBER)  <KNOWLEDGE-CATEGORY>:  <KNOWLEDGE REQUIREMENT>
```
Analysis examine all knowledge requirements against both the user class profile(s) and the current design. To address this example knowledge requirement, the current interface design has a `<Dictionaries In Use>` field which implies dictionaries must be selected to use (i.e., search). The question is: Can it be assumed that users understand this connection? Will the average user realize that the `<Dictionaries In Use>` field must be filled before the searching? Our analyst says yes.

Deduction and inference, as used in this example, is subjective and requires the judgment of a skilled human-computer interaction specialist (whether it is a task analyst, interaction designer, or whatever). This dependency on expertise represents a compromise in the trade-off against using automatic deduction which require much more elaborate and formal knowledge representation techniques—and it is one trade-off that the TMM is willing to take. Also, just opening a debate over user knowledge support helps to foster an user-centered design environment.

Continuing with our example, the following table outlines all the knowledge requirements from our example, as well as, show if its supported or unsupported.
<table>
<thead>
<tr>
<th>Knowledge Requirement</th>
<th>Justification</th>
<th>User Class Profile</th>
<th>Interface</th>
<th>Un-Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) PK: A dictionary must be chosen before the search</td>
<td>Interface has a <code>&lt;DICTIONARIES IN USE&gt;</code> field which implies to the user a dictionary must be selected. Also, if no dictionary is selected, then the <code>&lt;SEARCH COMMAND&gt;</code> field will not accept input.</td>
<td>✔</td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(1) FK: Dictionaries are alphabetically indexed by words</td>
<td>Users understand dictionaries (UCP. UKB. 2)</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(1) CK: Database Functions and Structures</td>
<td>Users understand basic Database Functions (UCP. UKB. 3).</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(2) FK: Search criteria, <code>&lt;WORD&gt;</code>, is POPLSSAY (Also in 2)</td>
<td>Part of the Task.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(3) FK: Databases can be searched</td>
<td>Users understand basic Database Functions (UCP. UKB. 3). Also, the interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying search command exists.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(3) FK: A command, <code>&lt;SEARCH&gt;</code>, exists to locate data in database</td>
<td>Users understand basic Database Functions (UCP. UKB. 3). Also, the interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying search command exists.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(4) PK: The search criteria must be specified for search</td>
<td>Users understand basic Database Functions (UCP. UKB. 3). Also, the interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying input is necessary.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(5) PK: A method exists to search the database</td>
<td>Users understand basic Database Functions (UCP. UKB. 3). Also, the interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying search command exists.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(6) PK: Input of search command name is by keyboard entry</td>
<td>The interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying search command exists.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(6) FK: <code>FIND</code> is the <code>&lt;SEARCH&gt;</code> command</td>
<td>User does not know the <code>FIND</code> command.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(7) PK: Input of search criteria is by keyboard entry</td>
<td>The interface has a prompt <code>&lt;SEARCH COMMAND:&gt;</code> implying search criteria must be entered. Also, the input field is text oriented and users know this (UCP. UKB. 4 - 5).</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(8) PK: <code>&lt;CR&gt;</code> Terminates input and executes command</td>
<td>Command-Line interface style within a WIMPS interface.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(8) PK: Input of <code>&lt;CR&gt;</code> is by keyboard entry</td>
<td>Command-Line interface style within a WIMPS interface.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(9) PK: Search results are presented in <code>&lt;RESULTS&gt;</code> window</td>
<td>System feedback will indicate results.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(9) CK: Format of Search Results</td>
<td>Interface will return results in standard dictionary format, and users understand dictionaries (UCP. UKB. 2)</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(10) PK: When search criteria is not located, the task is repeated</td>
<td>User knows when task fails or is accomplished based on system feedback.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
<tr>
<td>(11) PK: When search criteria is located, the task is completed</td>
<td>User knows when task fails or is accomplished based on system feedback.</td>
<td></td>
<td></td>
<td>✔</td>
</tr>
</tbody>
</table>

Table 3. User Knowledge Requirements Analysis Results

The next table gives a partial list of user knowledge requirements associated with the task `SELECT DICTIONARY DATABASE` (which was not expanded or decomposed in our task description due to space requirements). We include these knowledge requirements to further our example's meaningfulness. These knowledge requirements are self-reporting and obvious.
<table>
<thead>
<tr>
<th>Knowledge Requirement</th>
<th>Justification</th>
<th>User Class Profile</th>
<th>Interface</th>
<th>Un-Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td>FK: There are multiple, selectable, specialized dictionaries</td>
<td>Users understand dictionaries (UCP.UKB.2). Interface has (&lt;SELECT DICTIONARIES&gt;) prompt which implies that multiple dictionaries can be selected.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FK: Able to search on more than one dictionary</td>
<td>Interface has &lt;DICTIONARIES IN USE&gt; field which implies that multiple dictionaries can be used for searching.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK: Abstraction level of dictionary is much higher than that of item</td>
<td>Users understand dictionaries (UCP.UKB.2).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK: Dictionary may not be appropriate for search criteria selection</td>
<td>The search criteria may not give hints for which dictionary to select. E.g., is philosis in a Common or a Medical Dictionary?</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FK: Dictionaries have names, &lt;DICTIONARY&gt;</td>
<td>Users understand dictionaries (UCP.UKB.2). Interface identifies dictionaries by name.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FK: A scrolling list of dictionaries, &lt;DICTIONARY LIST&gt; identifies all dictionaries</td>
<td>Interface has &lt;SELECT DICTIONARIES&gt; field.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK: A dictionary must be correctly identified and selected before use.</td>
<td>Users understand dictionaries (UCP.UKB.2). Users understand noun-verb interaction (UCP.UKB.5). Interface requires entry into &lt;DICTIONARIES IN USE&gt; field before searching is possible.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CK: Membership relationship of search criteria to dictionaries in &lt;DICTIONARY-LIST&gt;</td>
<td>Users cannot be expected to know membership of the dictionaries. Interface does not provide support (apart from searching) to determine membership.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK: Point+Click on &lt;DICTIONARY&gt; selects associated dictionary</td>
<td>Users understand WIMPS interface (UCP.UKB.4).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PK: Point+Click on &lt;USE-DICTIONARY&gt; Button adds selected dictionary to list of dictionaries in use.</td>
<td>Users understand WIMPS interface (UCP.UKB.4).</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4. User Knowledge Requirements Analysis Results for the Task SELECT DICTIONARY DATABASE

It can be seen with both tables that many of the knowledge requirements are supported. This is a good indication about the quality of the design in terms of user knowledge support. However, there are several knowledge requirements that are not supported. These requirements are where analysis want to focus their attention.
V Synthesizing New User Interface Design Requirements

“In spite of appearances, people seldom know what they want until you give them what they ask for.”

(Gause, et al., 1990, p. 53)

At this point in the analysis of our example task (SELECT A DICTIONARY AND LOCATE THE WORD AND MEANING: POPINJAY), we have identified all unsupported knowledge requirements. According to the TMM life cycle, each of these unsupported knowledge requirements must be translated into a new user interface design requirement (UIDR). This section outlines using templates to derive the new UIDRs.

V.1 Templates

By far the easiest way to transform unsupported knowledge requirements into new UIDRs is to use templates. Depending on the category of knowledge, each knowledge requirement is translated into a new UIDR by the following templates:

**FACTUAL KNOWLEDGE UIDR**

\[
\text{UIDR: } \text{COMMUNICATE}<\text{FACT}>\text{ TO USER} \\
\text{(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).}
\]

**CONCEPTUAL KNOWLEDGE UIDR**

\[
\text{UIDR: } \text{PROVIDE UNDERSTANDING OF}<\text{CONCEPT}>\text{ TO USER} \\
\text{(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).}
\]

**PROCEDURAL KNOWLEDGE UIDR**

\[
\text{UIDR: } \text{IDENTIFY POSSIBLE PATHS OF EXECUTION} \text{ TO USER} \\
\text{(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).}
\]

These templates are simplistic, however, their main purpose is to communicate new design requirements to interface designers for re-design purposes. In fact, these templates just represent a standard format that analysts can use to record and communicate unsupported knowledge requirements found during analysis. These are only example templates which can be modified for specific uses.

Recalling that our example analysis produced four unsupported user knowledge requirements, using these templates yields the following UIDRs:

\[
\text{UIDR}_1 \text{COMMUNICATE}<\text{“FIND” IS THE}<\text{}\text{SEARCH}\text{COMMAND}>\text{ TO USER} \\
\text{(AT THE APPROPRIATE TIME IN TASK PERFORMANCE).}
\]
V.2 Alternative Method: The Holistic Approach

Another approach is to examine unsupported knowledge requirements and generate all-encompassing UIDs—i.e., a contracted set of UIDRs that contain all knowledge requirements. This holistic approach is much more dependent on the skill, intuition, and experience of the analyst, and therefore, is not suggested to the first time TMM analyst.

In most cases, this approach yields UIDRs that are abstractions of lower-level UIDRs. With our DICTIONARY example, the holistic approach would produce a single UIDR—which replaces UIDR1 and UIDR2. For example,

\[ \text{UIDR1': Do not mix interaction styles.} \]

*Keep in mind that the templates are only examples of how to formal UIDRs—they do not necessarily need to be followed.*

VI What now?

"Each solution is the source of the next problem."

(Gause, et al., 1990, p. 53)

VI.1 How do UIDRs Help the User Interface Design Process?

UIDRs seem basic and simplistic—how do they help the user interface design process? UIDRs notify interface designers of missing user knowledge support, however, it is up to interface designers to decide how to include these design requirements within re-designs. The creative process of designing user interface solutions still belongs to the interface designers. The TMM does not support the inclusion of the UIDRs into the re-designs.
For closure, however, we have re-designed the example dictionary user interface with the generated UIDRs. The designer addressed each UIDR in the following ways:

- To search, the user enters the `<SEARCH CRITERIA>` into the `WORD/PHRASE` prompt field and selects the `<DO SEARCH>` button, which replaces FIND command word. Since the search is also initiated by the `<DO SEARCH>` button selection, the `<CR>` is no longer required. This re-design resolves UIDR1 and UIDR2.

- UIDR3 and UIDR4 are difficult to resolve. In some cases users will not have any idea as to which dictionary contains a particular word. The two-fold re-design solution is to have a button, `<DICTIONARY SPECIFIC INFORMATION>`, that can help identify types of words associated with each dictionary, and a button `<HOW TO SELECT DICTIONARIES>`, that provides information to the user on dictionary selection. Albeit, these are not total solutions.

- In response to other user knowledge requirements identified within the analyses, the designer also included context specific help by incorporated question buttons. E.g.,

A possible re-design could look like:
Figure 10. Possible User Interface Re-design

Are we finished? No. This new design is henceforth subjected to another round of implementation, evaluation, and re-design again! And so on until the project is canceled or the project manager announces “It’s done!” In either case, odds are, there will still be some usability problems within the design, however sublime.
VII Discussion

"The fish is always the last to see the water."
(Gause, et al., 1990, p. 53)

"Do not feed nuts to a man with no teeth."
(Seashore, Seashore, & Weinberg, 1992, p. 139)

The TMM is not a Holy Grail of the task analysis world. The TMM does not identify and solve usability problems, it only helps to identify and focus designers on usability issues related to user knowledge requirements. Focusing designers on user-centered issues, like user knowledge requirements, is a quality of the TMM that surely benefits user interface design.

This brief introduction has hopefully illustrated the usefulness of the TMM—not by our little example, but through motivating how and why the TMM approaches analysis.
Appendix A: Domain Language

The language or terminology used in the computer semantic and syntactic domains is derived from (Lenorovitz, et al., 1984). Their taxonomies are reproduced here:

<table>
<thead>
<tr>
<th>PERCEIVE</th>
<th>ACQUIRE</th>
<th>DETECT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>SEARCH</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SCAN</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EXTRACT</td>
</tr>
<tr>
<td></td>
<td>IDENTIFY</td>
<td>DISCRIMINATE</td>
</tr>
<tr>
<td>MEDIATE</td>
<td>ANALYZE</td>
<td>CATEGORIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALCULATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ITEMIZE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABULATE</td>
</tr>
<tr>
<td>SYNTHESIZE</td>
<td>Estimate</td>
<td>INTERPOLATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRANSLATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTEGRATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORMULATE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROJECT OR EXTRAPOLATE</td>
</tr>
<tr>
<td>ASSESS</td>
<td>COMPARE</td>
<td>EVALUATE</td>
</tr>
<tr>
<td>DECIDE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COMMUNICATE</td>
<td>TRANSMIT</td>
<td>CALL</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACKNOWLEDGE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESPOND</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUGGEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIRECT</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INFORM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTRUCT</td>
</tr>
<tr>
<td></td>
<td>RECEIVE</td>
<td>REQUEST</td>
</tr>
</tbody>
</table>

User-Internal Taxonomy

User-Input Taxonomy

Taxonomies of user Actions (Lenorovitz, et al., 1984)

The definitions are given in the following tables (still showing the hierarchy):
<table>
<thead>
<tr>
<th>PERCEIVE</th>
<th>ACQUIRE</th>
<th>DETECT</th>
<th>Discover or notice an occurrence (usually unsolicited)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEARCH</td>
<td></td>
<td>PURPOSEFUL EXPLORATION OR LOOKING FOR SPECIFIED ITEM(S).</td>
<td></td>
</tr>
<tr>
<td>SCAN</td>
<td></td>
<td>Glance over quickly, usually looking for overall patterns or anomalous occurrences (not details).</td>
<td></td>
</tr>
<tr>
<td>EXTRACT</td>
<td></td>
<td>Directed, attentive reading, observing, or listening with the purpose of gleaning the meaning or contents thereof.</td>
<td></td>
</tr>
<tr>
<td>CROSS-REFERENCE</td>
<td></td>
<td>Accessing or looking up related information usually by means of an indexing or organized structuring scheme set up for that purpose.</td>
<td></td>
</tr>
<tr>
<td>IDENTIFY</td>
<td></td>
<td>DISCRIMINATE</td>
<td>Roughly classify or differentiate an entity in terms of a gross level grouping or set membership—frequently on the basis of only a limited number of attributes.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECOGNIZE</td>
<td>Specific, positive identification of an entity.</td>
</tr>
<tr>
<td>MEDIATE</td>
<td>ANALYZE</td>
<td>CATEGORIZ</td>
<td>Classify or sort one or more entities into specific sets or groupings, usually on the basis of a well-defined classification scheme.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>CALCULATE</td>
<td>Reckon, mentally compute, or computationally determine.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ITEMIZE</td>
<td>List or specify the various components of a grouping.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TABULATE</td>
<td>Tally or enumerate the frequencies or values of the members of an itemized list or table.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESTIMATE</td>
<td>Mentally gauge, judge, or approximate, often on the basis of incomplete data.</td>
</tr>
<tr>
<td>SYNTHESIZE</td>
<td></td>
<td>INTERPOLATE</td>
<td>Assign an approximate value to an interim point based upon knowledge of values of two or more bracketing reference points.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TRANSLATE</td>
<td>Convert or change from one form or representational system to another according to some consistent mapping scheme.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INTEGRATE</td>
<td>Pull together, and mentally organize a variety of data elements so as to extract the information contained therein.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORMULATE</td>
<td>Generate and put together a set of ideas so as to produce an integrated concept or plan.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PROJECT OR EXTRAPOLATE</td>
<td>Assign an approximate value to a future point based upon the value(s) of preceding point(s).</td>
</tr>
<tr>
<td>ASSESS</td>
<td></td>
<td>COMPARE</td>
<td>Consider two or more entities in parallel so as to note relative similarities and differences.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EVALUATE</td>
<td>Determine the value, amount, or worth of an entity, often on the basis of a standard rating scale or metric.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DECIDE</td>
<td>Arrive at an answer, choice, or conclusion.</td>
</tr>
<tr>
<td>COMMUNICATE</td>
<td>TRANSMIT</td>
<td>CALL</td>
<td>Signal to a specific recipient or set of recipients that a message is forthcoming.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ACKNOWLEDGE</td>
<td>Confirm that a call or message has been received.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RESPOND</td>
<td>Answer or reply in reaction to an input.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SUGGEST</td>
<td>Offer for consideration.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DIRECT</td>
<td>Provide explicitly authoritative instructions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INFORM</td>
<td>Pass on or relay new knowledge or data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>INSTRUCT</td>
<td>Teach, educate, or provide remedial data.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>REQUEST</td>
<td>Solicit, query, or ask for.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RECEIVE</td>
<td>Get, obtain, or acquire an incoming message.</td>
</tr>
</tbody>
</table>

User-Internal Taxonomy
Definitions for Lenorovitz Taxonomies (Lenorovitz, et al., 1984)
<table>
<thead>
<tr>
<th>CREATE</th>
<th>ASSOCIATE</th>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GROUP</td>
<td>Link together or associate for purposes of identification/reference.</td>
<td></td>
</tr>
<tr>
<td>INTRODUCE</td>
<td>INSERT</td>
<td>Make space for and place an entity at a selected location within the bounds of another such that the latter wholly encompasses the former, and the former becomes an integral component of the latter.</td>
<td></td>
</tr>
<tr>
<td>ASSEMBLE</td>
<td>AGGREGATE</td>
<td>Combine two or more components so as to form a new composite entity.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OVERLAY</td>
<td>Superimpose one entity on top of another so as to affect a composite appearance while still retaining the separability of each component layer.</td>
<td></td>
</tr>
<tr>
<td>REPLICATE</td>
<td>COPY</td>
<td>Reproduce one or more duplicated of an entity (no links to master).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INSTANCE</td>
<td>Reproduce an original (master) entity in such a way as to retain a definitional link to the master—i.e., such that any subsequent changes or modifications made to the master will automatically be reflected in each and every instance created therefrom.</td>
<td></td>
</tr>
</tbody>
</table>

| INDICATE   | SELECT (POS/OBJ) | Opt for or choose an entity (e.g., a position or an object) by pointing to it [and possibly other user actions]. |
| REFERENCE  | Opt for or choose an entity by invoking its name. |

| ELIMINATE  | REMOVE    | Cut | Remove a designated portion of an entity and place it in a special purpose buffer (residual components of the original entity usually close in around hole left by cut-out portion). |
|           |          | DELETE | Remove and (irrevocably) destroy a designated portion of an entity. |
|           |          | SUSPEND | Stop a process and temporarily hold in abeyance for future restoration. |
|           |          | TERMINATE | Conclude a process such that it cannot be restarted from the point of interruption, only by complete re-initiation. |
| DISASSOCIATE | RENAME   | Change an entity's title or label, without changing the entity itself. |
|            | UN-GROUP | Eliminate the common bond or reference linkage of a group of entities. |
| DISASSEMBLE | SEGREGATE | Partition and separate an entity into one or more component parts such that the structure and identity of the original is lost. |
|            | FILTER   | Selectively eliminate one or more layers of an overlaid composite. |
|            | SUPPRESS | Conceal or keep back certain aspects or products of a process without affecting the process itself (i.e., affects appearance only). |
|            | SET-ASIDE | Remove entire contents of current (active) work area and store in a readily accessible buffer (for future recall). |

| MANIPULATE | TRANSFORM | Manipulate or change one or more of an entity's attributes (e.g., color, line type, character font, size, orientation) without changing the essential content of the entity itself. |

| ACTIVATE   | EXECUTE | Initiate or activate any of a set of predefined utility or special purpose functions (e.g., sort, merge, calculate, update, extract, search, replace). |

User-Input Taxonomy
Definitions for Lenorovitz Taxonomies (Lenorovitz, et al., 1984)
Appendix B: Analysis Forms

This section provides several blank templates (Forms) for TMM Analysts:

TMM Task Description Form
TMM Identified Knowledge Requirement Form
Translation Analysis Form

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the author(s) copyright notice and title of the publication and its date appear, and notice is given that copying is by permission of the author(s). To copy otherwise, or to republish, requires a fee and/or specific permission.
## TMM Identified Knowledge Element Form

**Task/Design:**

**Designer(s):**

**Date:**

**Reviewed By:**

<table>
<thead>
<tr>
<th>Identified User Knowledge Needs</th>
<th>Notes (e.g., justification of support or non-support)</th>
<th>Supported</th>
<th>Depressed</th>
<th>Supported within Tolerance</th>
<th>Not Supported</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Factual Knowledge Elements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Conceptual Knowledge Elements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Procedural Knowledge Elements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix B: TMM Tutorial
Acknowledgments

I would like to acknowledge the extraordinary efforts of Rex Hartson, and the continuing efforts of Deborah Hix and the Virginia Tech HCI Research Group. I would also like to acknowledge the collected works of Monty Python, with special emphasis on (Monty Python, 1987) CD 2 Track 5 (Architect’s Sketch) which inspires the need for proper task analysis and user-centered design.

References


APPENDIX C: UIDR REPORT FORM

This appendix reproduces a blank UIDR Report Form used in phase 2 of the TMM evaluation experiment.
This form is used by user interface analysts and designers to record new user interface design requirements. *PLEASE ATTACH ALL RELATED ANALYSIS MATERIALS TO THIS FORM (I.E., TMM FORMS, IDENTIFIED GUIDELINES).*

**Interface/Task:**

**Usability Problem:**

**New User Interface Design Requirement(s):**

How did you come up with these new requirements? Check all that apply:
- Using the analysis method (either TMM or Design Guidelines)
- Relying on previous experience with interface design
- Relying on previous task domain knowledge
- By accident (discovered these requirements by other methods), please elaborate:

---

Estimated time to derive user interface design requirement(s); (e.g., 5 min.; 3 hr.): __________

*Please continue on back.*
Characterize (by circling) the usability problem severity (i.e., how much negative effect will this have on the user?):

<table>
<thead>
<tr>
<th>Not Noticeable to User</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>NA Very Noticeable to User</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Bad Impact</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>NA Very Good Impact</td>
</tr>
</tbody>
</table>

Characterize (by circling) your estimation of the UIDR's difficulty (i.e., how difficult is it to include these UIDRs into the interface design?):

<table>
<thead>
<tr>
<th>Difficult to Include</th>
<th>-3</th>
<th>-2</th>
<th>-1</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>+3</th>
<th>NA Easy to Include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Difficult to Re-Design</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>NA Easy to Re-Design</td>
</tr>
<tr>
<td>Difficult to Re-Code</td>
<td>-3</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>NA Easy to Re-Code</td>
</tr>
</tbody>
</table>

Additional comments:
APPENDIX D: ANALYST SURVEYS

This appendix reproduces the analyst surveys used in phase 2 of the TMM evaluation experiment.
USER INTERFACE ANALYST SURVEY

Participant ID Number: __________

Date: _______________________

Researchers
Kevin A. Mayo, Department of Computer Science, 231-6470, 951-4337
H. Rex Hartson, Department of Computer Science

Purpose: The purpose of this questionnaire is to survey individual analysts to assess the method used in deriving new user interface design requirements.

Method: Questions on this survey are of varying form, and the 'closest answer' should be reported. This survey has scaled questions, short answer questions, and an area for further elaboration.

Scaled questions have responses based on a range of answers. The scales represent the range of possible answers, e.g., good to bad, black to white, agree to disagree, conservative to liberal. For each of the questions, both ends of the scale use common terminology. Clarification on range boundaries should be directed to the researcher listed above. E.g., How would you rate: DEGREE REQUIREMENTS:

rigid -3 -2 -1 0 +1 +2 +3 NA flexible

Short answer questions are designed for responses limited to a single paragraph or less. An extended exposition of your viewpoint would be helpful—but not necessary. Please focus your response on the question asked, and refer tangential replies to the area for further elaboration. Clarification on the meaning of the questions should be directed to the researcher listed above.

A section, at the end of the survey, is for further elaboration. This section is for comments and suggestions concerning the study and survey.

Results: The results of this survey will be used by the researchers listed above. Anonymity of the participants is assured, and the researcher agrees that only sanitized results will be released—no names, companies, affiliations, or personally identifying characteristics will be released. All results, when possible, will be reported in whole and not on an individual basis.

Again, thank you for your time and effort.

Complete this survey AFTER all training and analysis

Appendix D: Analyst Surveys
USER INTERFACE ANALYST SURVEY

The following set of questions concerns your overall reaction to the method you used for producing new user interface design requirements.

1. Circle your overall reaction (on the following scales) to the METHOD (TMM or Guidelines) you used to produce new user interface design requirements.

   The analysis METHOD:
   
   difficult to use  -3  -2  -1  0  +1  +2  +3  NA easy to use
   inadequate power -3  -2  -1  0  +1  +2  +3  NA adequate power
   rigid -3  -2  -1  0  +1  +2  +3  NA flexible
   not usable -3  -2  -1  0  +1  +2  +3  NA very usable
   frustrating -3  -2  -1  0  +1  +2  +3  NA satisfying
   dull -3  -2  -1  0  +1  +2  +3  NA stimulating
   terrible -3  -2  -1  0  +1  +2  +3  NA wonderful

   The METHOD PROCESS (i.e., how the METHOD is used):
   
   ambiguous -3  -2  -1  0  +1  +2  +3  NA defined
   too many steps -3  -2  -1  0  +1  +2  +3  NA too few steps
   too much time -3  -2  -1  0  +1  +2  +3  NA too little time
   too complex -3  -2  -1  0  +1  +2  +3  NA too simplistic
   unclear starting point -3  -2  -1  0  +1  +2  +3  NA clear starting pt.
   unclear ending point -3  -2  -1  0  +1  +2  +3  NA clear ending point

2. What is your overall reaction (on the following scales) to the NOTATION and FORMALISMS you used to produce new user interface design requirements? That is, please comment on the notational conventions you used in the METHOD (TMM or Guidelines).

   difficult -3  -2  -1  0  +1  +2  +3  NA easy
   complex -3  -2  -1  0  +1  +2  +3  NA simplistic
   inadequate-
   representational power -3  -2  -1  0  +1  +2  +3  NA representational power
   rigid -3  -2  -1  0  +1  +2  +3  NA flexible
   frustrating -3  -2  -1  0  +1  +2  +3  NA satisfying
   dull -3  -2  -1  0  +1  +2  +3  NA stimulating
   terrible -3  -2  -1  0  +1  +2  +3  NA wonderful

3. What is your overall reaction (on the following scales) to the LEARNABILITY of the PROCESS and NOTATIONS used within the METHOD (TMM or Guidelines) you used to produce new user interface design requirements?

   Learnability of the METHOD's PROCESS:
   difficult to learn -3  -2  -1  0  +1  +2  +3  NA easy to learn
   long time to learn -3  -2  -1  0  +1  +2  +3  NA short time to learn
   difficult concepts -3  -2  -1  0  +1  +2  +3  NA easy concepts

   Learnability of the METHOD's NOTATION:
   difficult to learn -3  -2  -1  0  +1  +2  +3  NA easy to learn
   long time to learn -3  -2  -1  0  +1  +2  +3  NA short time to learn
   difficult notation -3  -2  -1  0  +1  +2  +3  NA easy notation
4. What is your overall reaction (on the following scales) to working in groups using the METHOD (TMM or Guidelines) assigned to you?

Because of WORKING IN GROUPS (vs. individually), analysis is/expected:

- more difficult: -3 -2 -1 0 +1 +2 +3 NA more easy
- more incomplete: -3 -2 -1 0 +1 +2 +3 NA more complete
- increased time: -3 -2 -1 0 +1 +2 +3 NA decreased time
What did you like best about using the analysis METHOD? (Include strengths.)

What did you like least about using the analysis METHOD? (Include weaknesses.)
USER INTERFACE ANALYST SURVEY

ID Number: _____________________ Date: _______________

What does the METHOD/PROCESS/NOTATION not provide?

Did the analyses produce results that you expected or pre-determined before the analyses? Did the analyses produce results that you did not expect? If so, please explain.
The following survey was given to TMM analysts only.
The following set of questions concerns your overall reaction to the TMMs

5. Please circle your overall reactions (on the given scales) to the following:

**Using TMM DOMAINS:**
- difficult to use -3 -2 -1 0 +1 +2 +3 NA easy to use
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Finding TMM DOMAIN ITEMS:**
- difficult to find -3 -2 -1 0 +1 +2 +3 NA easy to find
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Identifying TMM MAPPINGS:**
- difficult to identify -3 -2 -1 0 +1 +2 +3 NA easy to identify
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Identifying TMM MAPPING KNOWLEDGE REQUIREMENTS:**
- difficult to identify -3 -2 -1 0 +1 +2 +3 NA easy to identify
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Classifying TMM KNOWLEDGE REQUIREMENTS** as **Factual, Conceptual, or Procedural:**
- difficult to classify -3 -2 -1 0 +1 +2 +3 NA easy to classify
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Determining if TMM KNOWLEDGE REQUIREMENTS are Supported or Unsupported:**
- difficult to determine -3 -2 -1 0 +1 +2 +3 NA easy to determine
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Synthesizing new USER INTERFACE DESIGN REQUIREMENTS using the TMM:**
- difficult to synthesize -3 -2 -1 0 +1 +2 +3 NA easy to synthesize
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying
**USER INTERFACE ANALYST SURVEY: TMM METHOD**

**ID Number:** ________________________ **Date:** ________________________

**TMM Task Description Domains:**
Are the domain distinctions clear?

Are all necessary levels of abstraction present?

What problems did you have casting tasks and items into specific domains?

**TMM Task Description Mappings:**
Are the mappings obvious and clear? (If not, why not?)

What problems did you have identifying the mappings?

What problems did you have associating user knowledge requirements with each of the mappings?
USER INTERFACE ANALYST SURVEY: TMM METHOD

ID Number: __________________________ Date: _________________

TMM User Knowledge Requirements:
Are there enough knowledge categories? (If not, what else?)

What problems did you have identifying user knowledge and their types?

TMM Unsupported Knowledge Analysis:
What problems did you have determining knowledge support vs unsupported?

What characteristics should be in the user class profile to support this analysis?
TMM Synthesis of Design Requirements:
Does TMM provide enough information to synthesize design requirements? (If not enough information—what else is needed? If too much information—what is not needed?)

TMM Analysis Life-Cycle:
Are there any stages left out of the TMM analysis life-cycle?

Are the stages distinct enough?
TMM Training.
Was the training adequate?

What should be added/removed to/from the training?

What topics should be discussed further?

Other Comments:
The following survey was given to DGL analysts only.
The following set of questions concerns your overall reaction to the Design Guidelines

5. Please report your overall reactions (on the following scales) to the following:

**Using Design Guidelines:**
- difficult to use -3 -2 -1 0 +1 +2 +3 NA easy to use
- inadequate power -3 -2 -1 0 +1 +2 +3 NA adequate power
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Finding applicable Design Guidelines:**
- difficult to find -3 -2 -1 0 +1 +2 +3 NA easy to find
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Selecting out necessary Design Guidelines from applicable Design Guidelines:**
- difficult to select -3 -2 -1 0 +1 +2 +3 NA easy to select
- inadequate support -3 -2 -1 0 +1 +2 +3 NA adequate support
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying

**Synthesizing new USER INTERFACE DESIGN REQUIREMENTS using Design Guidelines**
(tailoring the Design Guidelines into UIDs):
- difficult to synthesize -3 -2 -1 0 +1 +2 +3 NA easy to synthesize
- inadequate capability -3 -2 -1 0 +1 +2 +3 NA adequate capability
- rigid -3 -2 -1 0 +1 +2 +3 NA flexible
- dull -3 -2 -1 0 +1 +2 +3 NA stimulating
- frustrating -3 -2 -1 0 +1 +2 +3 NA satisfying
Design Guidelines and Task Descriptions:
Task description/decomposition is essential to task analysis. In this light and your experience within this experiment, how do design guidelines help task description/decomposition?

How do design guidelines hinder task description/decomposition?

Design Guidelines Synthesis of Design Requirements:
Do Design Guidelines provide enough information to synthesize user interface design requirements? (If not enough information—what else is needed? If too much information—what is not needed?)
Design Guidelines Analysis Life-Cycle:
Are there any stages left out of the Design Guidelines analysis life-cycle?

Are the stages distinct enough?

Design Guidelines Training.
Was the training adequate?

What should be added/removed to/from the training?

What topics should be discussed further?
Other Comments:
APPENDIX E:  UIDR QUALITY REPORT FORM

This appendix reproduces the UIDR Quality Report Form used in phase 3 of the TMM evaluation experiment.
DIRECTIONS FOR THE UIDR QUALITY FORM

INTRODUCTION
The purpose of the UIDR QUALITY FORM is to record your comparison results of two sets of User Interface Design Requirements (UIDRs). You will make several head-to-head comparisons of the sets against criteria defined below. Please record your responses on the following scale by checking a single box:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is equivalent to Set 1</th>
<th>Set 2 is slightly better than Set 1</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
<td>□</td>
</tr>
</tbody>
</table>

Each box indicates the criterion's level of support in relation to both UIDR sets. The scale ranges from Set 1 being much better to Set 2 to Set 2 being much better to Set 1. The comparison criteria used in this evaluation are:

**USABILITY PROBLEM RELATED CRITERIA**
- Addresses Usability Problem
- Cohesiveness
- Specificity

**USER RELATED CRITERIA**
- User Centeredness
- Addresses Users' Needs
- Addresses Users' Task Structure

**ARTIFACT RELATED CRITERION**
- Avoid Introduction of New Usability Problems
DIRECTIONS FOR THE UIDR QUALITY FORM

ADRESSES USABILITY PROBLEM

UIDRs address usability problems identified during formative evaluation. Therefore, the primary criterion for evaluating UIDRs is based on the question: In comparison, how well does each UIDR set identify requirements for possible solution(s) to the particular usability problems?

Does the UIDR set address the usability problem?

E.g.: USABILITY PROBLEM: Users don't understand meaning of United Database.

UIDR (Addresses usability problem):
Identify meaning of United Database to user, or change wording (of United Database) to more directly reflect the intended meaning.
This UIDR specifically address the users' lack of understand of the term United Database.

UIDR (Does not address stated usability problem):
Represent database elements using ICONs.
Obviously this UIDR does not directly address the usability problem.

In this comparison, you should examine the usability problem, user task, and interface prototype to determine comparatively how well each UIDR set addresses the stated usability problems.
COHESIVENESS

Cohesiveness is defined, for this purpose, to characterize UIDR sets that focus on one (or a select related few) concepts/ideas/interface features. That is, all UIDRs in a set should be about the same thing. This criterion is concerned with cohesiveness within a single UIDR set—NOT between UIDR sets.

E.g.:
UIDR SET 1 (cohesive within the UIDR set)
UIDR: Communicate meaning/use of CD (Certificate of Deposit) to user
UIDR: Identify possible CD Deposit Periods to user
UIDR: Inform the user of Penalty for Early Withdrawal
Each of the UIDRs within Set 1 focuses on the meaning and use of CDs.

UIDR SET 2 (less cohesive within UIDR set)
UIDR: Display all deposit periods for CDs
UIDR: Display phone numbers of Loan Officers
UIDR: Allow user access to personal accounts
Set 2 diverges and includes information about loan officers and personal accounts

Set 1 is more cohesive than Set 2.
Directions for the UIDR Quality Form

Specificity

Specificity is defined to characterize the level of detail and precision of the UIDR sets—ranging from ambiguous (or vague) to fully and precisely specified. In other words, this criterion focuses on the wording and completeness of description of the UIDRs.

E.g.: UIDR (not well specified)
Create a means for protecting a small group of human beings from the hostile elements of their environment.
The solution to this could be: Igloo, Bavarian Castle, or Space Station—all are very different solutions to this ambiguous UIDR.

UIDR (more specific)
Create a shelter for human beings that live in a temperate weather zone using only bricks and mortar.
This UIDR is more specific because it includes the climate and available building materials.

The following example illustrates UIDRs that are given in terms of design solutions. Although it is not the intent of UIDRs to provide design solutions, when interpreted in the proper way, these UIDRs are still acceptable as requirements. Designers must view these UIDRs as suggestions of possible design solutions, without constraining the solution space.

E.g.: UIDR (not well specified)
Make error messages explicit.

UIDR (more specific)
Use RED color coding coupled with explicit error messages to notify users of error conditions and situations.
This UIDR suggests color coding—but the specific color (red) could be ignored.

Specificity in UIDRs is affected by many factors:
- word ambiguity or alternative word meanings (e.g., small vs. an exact measurement)
- interface feature ambiguity (e.g., selectable item vs. a button labeled 'OK')

DIRECTIONS FOR THE UIDR QUALITY FORM

USER-CENTEREDNESS
The well known concept and practices of user centeredness\(^2\) means that design should focus on the user. The sets of UIDRs should, therefore, reflect user-centeredness—NOT system-centeredness.

\textit{Is the UIDR set formulated using user-centered terms and concepts?}

\textbf{E.g.}: \textit{USABILITY PROBLEM}: User unable to proceed (i.e., didn’t understand or didn’t know what to do) when Interactive RoloDex System displayed error message: “System Database Full. Stack Overflow—Error 0091”.

\textbf{UIDR (User-Centered)}:
Inform the user that the RoloDex is full. Provide and identify several alternative resolution paths to the user.
\textit{This UIDR focuses on the users.}

\textbf{UIDR (System-Centered)}:
When system memory is full, send error message: “System Database Full. Stack Overflow—Error 0091”. \textit{This UIDR focuses on system functionality.}

In this example, the first UIDR focuses on the users’ needs/task, whereas the second UIDR focuses on system functionality.


Directions for the UIDR Quality Form

Address Users’ Needs

More specific than the previous user-centered criterion, this criterion compares how well each set of UIDRs addresses users’ needs associated with task performance. The emphasis on users’ mental and physical needs is what makes this criterion different from the criterion for addressing the usability problem.

Does the UIDR set address users’ physical needs?

E.g.: UIDR (addresses both left and right handed people):

Tracking system is controlled by one hand and can be set for either right- or left-hand orientation.

UIDR (physical needs not explicitly addressed):

Tracking system is manually controlled.

Does the UIDR set address users’ mental needs?

E.g.: UIDR (reduces users’ mental need to know transaction code—so that users don’t need to remember it):

Always display transaction code (and corresponding English description) in all windows of the user task.

UIDR (inadequate support for users’ mental need to know the transaction code):

Display transaction code in first window of user task.

Also, consider the users’ needs in the task and computer domains. In other words, Do the UIDRs provide specific guidance to designers to make the users’ chore of working in these domains easier?

E.g.: The previous UIDRs on displaying transaction codes serves as an example of task domain support—in this case the users are constantly reminded which transaction is occurring by the displayed code and English counterpart.

Wording is very important when comparing the UIDRs with this criterion.

E.g.: UIDR: PROVIDE INFORMATION ABOUT <PRINT-FIELD “OK” BUTTON>

This UIDR implies supporting the users’ information (knowledge) needs associated with <PRINT-FIELD “OK” BUTTON>, such as: what does this button do? how it is used? etc.

UIDR: PROVIDE BUTTON “OK” WITHIN PRINT-FILE DIALOG BOX

This UIDR provides only functional/artifact needs not necessarily associated with users’ mental or physical needs—no user support for how to use the button or what it is the button for.
DIRECTIONS FOR THE UIDR QUALITY FORM

ADDRESS USERS’ TASK STRUCTURE

Task structure is addressed in UIDRs by support for the conceptualization of how tasks fit together to accomplish user work goal(s). This criterion is related to the overall task structure rather than simple tasks in isolation.

In this comparison, you’ll examine UIDR sets to determine the extent to which the UIDRs incorporate an understanding of, and familiarity with the users’ task structure.

Does the UIDR set reference the users’ task structure?

E.g.: UIDR (with user task structure):
Inform users that files larger than a certain size must be segmented (via a provided segmentation task) before being sent using an e-mail system.

UIDR (without user task structure):
Provide a menu selection for file segmenting capability.

The first UIDR specifies when the segmenting sub-task is necessary or useful as well as how to accomplish it. That is, it gives information that a user must segment a file in order to successfully perform the higher level task of sending a file by email.

The second UIDR only gives information that users can segment a file—without relating this sub-task to the larger task structure.

Again, wording is critical when comparing sets of UIDRs with this criterion. UIDR sets that are artifact-oriented (i.e., concerned with icons, menus, buttons) are lacking in this criterion; while UIDR sets that are basically task-oriented (i.e., concerned with task structure and user actions) are strong in this criterion—see above example.
AVOID INTRODUCTION OF NEW USABILITY PROBLEMS

The function of UIDRs is to provide designers requirements that address usability problems (identified in formative evaluation). In some cases, UIDRs can spur a designer in a direction that may solve the initial usability problem, but introduces additional usability problems.

Does the UIDR set potentially introduce inherent (to the UIDR set) new usability problems?

E.g.: USABILITY PROBLEM: Users were unable to differentiate among system functions.

UIDR (could possibly lead to usability problems):
For each invoked system function, provide a new window.
This UIDR introduces usability problems—i.e., crowded screen and excessive window manipulation problems.

A better alternative is:

UIDR (provides alternatives to avoid new usability problems):
Provide a clean and distinct transition between system functions. If multiple windows are used, then hide or effectively tile them.

This criterion judges the possible inclusion of usability problems based on the UIDR sets. In your comparisons, you will need to look into your crystal ball and divine the future impact of the UIDR sets.
As an example of comparing UIDR sets, consider the following usability problem statement for a MEDICAL DATABASE SYSTEM. Two sets of UIDRs are given and evaluated using a subset of the comparison criteria.

Users: Physicians (GPs)
Task: Dispensing Prescriptions using current patient DB
Problem: Physicians voiced a need for automated support for accessing patient data. Physicians need this to be able to cross-reference all prescriptions patients are currently taking—(currently not supported).

| Set 1: UIDR1: System should support functionality that allows acting physicians to access patient database |
| Set 2: UIDR1: Provide access by acting physician to patient database |
| UIDR2: Provide automatic cross-referencing and notification of patient prescription side-effects and drug interactions once new prescriptions have been entered |

USABILITY PROBLEM RELATED COMPARISON CRITERION: Addresses Usability Problem:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

UIDR Set 2 clearly specifies a task that users will perform that Set 1 does not specify—the need for this task is imbedded within the problem description. Hence, Set 2 UIDRs are much better than Set 1 UIDRs for this criterion.

USER RELATED COMPARISON CRITERION: Addresses Users' Needs

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

While both sets addresses the users' need to access patient data, Set 2 UIDRs specify added functionality for cross-referencing prescriptions to identify drug-interactions—this functionality helps to reduce user knowledge requirements (mental load). Set 1 does not support this added functionality and thus fails to address some user needs; therefore, Set 2 is judged better than Set 1. In other words, Set 2 is better because it addresses Why the users need access to the patient data—the real user need is prescription cross-referencing not just database access.
**EXAMPLE FOR THE UIDR QUALITY FORM**

**ARTIFACT RELATED COMPARISON CRITERION:** Avoids Introduction of Usability Problems

<table>
<thead>
<tr>
<th></th>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Set 2 is equivalent to Set 1

Set 2 is slightly better than Set 1

Set 2 is better than Set 1

Set 2 is much better than Set 1

Neither of these two sets of UIDRs inherently introduce usability problems—their level of description is too high.
EXAMPLE FOR THE **UIDR QUALITY FORM**

Please examine the two sets of UIDRs to determine which set **ADDRESSES** the USABILITY PROBLEM better:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is equivalent to Set 1</th>
<th>Set 2 is slightly better than Set 1</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please elaborate:

Please examine the two sets of UIDRs to determine which set is more COHESIVE:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is equivalent to Set 1</th>
<th>Set 2 is slightly better than Set 1</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please elaborate:
Please examine the two sets of UIDRs to determine which set is more SPECIFIC:

Set 1 is much better than Set 2  Set 1 is better than Set 2  Set 1 is slightly better than Set 2  Set 1 is equivalent to Set 2  Set 2 is equivalent to Set 1  Set 2 is slightly better than Set 1  Set 2 is better than Set 1  Set 2 is much better than Set 1

Please elaborate:

Please examine the two sets of UIDRs to determine which set is more USER-CENTERED:

Set 1 is much better than Set 2  Set 1 is better than Set 2  Set 1 is slightly better than Set 2  Set 1 is equivalent to Set 2  Set 2 is equivalent to Set 1  Set 2 is slightly better than Set 1  Set 2 is better than Set 1  Set 2 is much better than Set 1

Please elaborate:
**EXAMPLE FOR THE Uldr Quality Form**

Please examine the two sets of UIDs to determine which set **addresses users' needs** better:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please elaborate:

Please examine the two sets of UIDs to determine which set **addresses users' task structure** better:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Please elaborate:
Please examine the two sets of UIDRs to determine which set AVOIDS INTRODUCTION OF USABILITY PROBLEMS better:

<table>
<thead>
<tr>
<th>Set 1 is much better than Set 2</th>
<th>Set 1 is better than Set 2</th>
<th>Set 1 is slightly better than Set 2</th>
<th>Set 1 is equivalent to Set 2</th>
<th>Set 2 is equivalent to Set 1</th>
<th>Set 2 is slightly better than Set 1</th>
<th>Set 2 is better than Set 1</th>
<th>Set 2 is much better than Set 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

Please elaborate:

Are there any other UIDRs (you can think of) that you think are better?

Please use this space to elaborate on any other differences you found between the two UIDR sets.
APPENDIX F: PHASE TWO DATA

This appendix contains all the raw data collected in phase two of the TMM evaluation experiment.
### TABLE A.F.1. RAW DATA FOR TMM ANALYST SURVEYS

<table>
<thead>
<tr>
<th>TMM</th>
<th>G01</th>
<th>S02</th>
<th>+1</th>
<th>+2</th>
<th>+2</th>
<th>+2</th>
<th>+1</th>
<th>+2</th>
<th>+2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G01</th>
<th>S10</th>
<th>-3</th>
<th>-2</th>
<th>+3</th>
<th>0</th>
<th>0</th>
<th>+1</th>
<th>+1</th>
<th>-2</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-2</td>
<td>-3</td>
<td>-3</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+1</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>+1</td>
<td>-3</td>
<td>0</td>
<td>+2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G02</th>
<th>S13</th>
<th>-1</th>
<th>+2</th>
<th>+1</th>
<th>0</th>
<th>-1</th>
<th>0</th>
<th>+1</th>
<th>-2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>0</td>
<td>-1</td>
<td>+2</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>+2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G02</th>
<th>S16</th>
<th>+2</th>
<th>+2</th>
<th>0</th>
<th>+1</th>
<th>+1</th>
<th>+1</th>
<th>+1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>-1</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G03</th>
<th>S15</th>
<th>+1</th>
<th>+3</th>
<th>-1</th>
<th>+1</th>
<th>-1</th>
<th>+2</th>
<th>+1</th>
<th>-2</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>-3</td>
<td>-1</td>
<td>+1</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>+2</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>+1</td>
<td>+2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G03</th>
<th>S01</th>
<th>+1</th>
<th>+2</th>
<th>+2</th>
<th>+1</th>
<th>+1</th>
<th>0</th>
<th>0</th>
<th>-1</th>
<th>-2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>+2</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G04</th>
<th>S03</th>
<th>-1</th>
<th>+2</th>
<th>-2</th>
<th>+2</th>
<th>0</th>
<th>+1</th>
<th>+2</th>
<th>-2</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>+2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G04</th>
<th>S08</th>
<th>+2</th>
<th>+2</th>
<th>+1</th>
<th>+1</th>
<th>+1</th>
<th>+2</th>
<th>+2</th>
<th>+2</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>-1</td>
<td>+2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G05</th>
<th>S12</th>
<th>-2</th>
<th>0</th>
<th>-1</th>
<th>-2</th>
<th>-3</th>
<th>+1</th>
<th>-1</th>
<th>-2</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>-3</td>
<td>+2</td>
<td>+2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TMM</th>
<th>G05</th>
<th>S05</th>
<th>-1</th>
<th>+2</th>
<th>+1</th>
<th>-2</th>
<th>-2</th>
<th>0</th>
<th>-3</th>
<th>+1</th>
<th>+1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-2</td>
<td>-1</td>
<td>+1</td>
<td>-2</td>
<td>-2</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGL</td>
<td>G06</td>
<td>S17</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGL</td>
<td>G06</td>
<td>S19</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>+2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGL</td>
<td>G07</td>
<td>S30</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>-1</td>
<td>0</td>
<td>+2</td>
<td>+1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>DGL</td>
<td>G07</td>
<td>S26</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td>DGL</td>
<td>G08</td>
<td>S18</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td>DGL</td>
<td>G08</td>
<td>S22</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>-1</td>
<td>-1</td>
<td>-2</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGL</td>
<td>G09</td>
<td>S28</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>DGL</td>
<td>G09</td>
<td>S29</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DGL</td>
<td>G10</td>
<td>S24</td>
<td>0</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>+1</td>
<td>+3</td>
<td>-2</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td>DGL</td>
<td>G10</td>
<td>S25</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>+3</td>
<td>+3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendix F: Phase Two Data
### TABLE A.F.3. METHOD COMPARISONS (TMM AND DGL)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>METHOD $F$ and $Pr &gt; F$</th>
<th>TEAM(METHOD) $F$ and $Pr &gt; F$</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M1</td>
<td>difficult to use</td>
<td>easy to use</td>
<td>6.75</td>
<td>0.0266</td>
<td>TMM: -0.100</td>
</tr>
<tr>
<td></td>
<td>inadequate power</td>
<td>adequate power</td>
<td>0.04</td>
<td>0.8455</td>
<td>TMM: +1.300</td>
</tr>
<tr>
<td>M3</td>
<td>rigid</td>
<td>flexible</td>
<td>1.53</td>
<td>0.2446</td>
<td>TMM: +0.600</td>
</tr>
<tr>
<td>M4</td>
<td>not usable</td>
<td>very usable</td>
<td>7.12</td>
<td>0.0236</td>
<td>TMM: +0.700</td>
</tr>
<tr>
<td>M5</td>
<td>frustrating</td>
<td>satisfying</td>
<td>5.26</td>
<td>0.0447</td>
<td>TMM: -0.200</td>
</tr>
<tr>
<td>M6</td>
<td>dull</td>
<td>stimulating</td>
<td>0.00</td>
<td>1.0000</td>
<td>TMM: +0.700</td>
</tr>
<tr>
<td>M7</td>
<td>terrible</td>
<td>wonderful</td>
<td>2.00</td>
<td>0.1877</td>
<td>TMM: +0.900</td>
</tr>
<tr>
<td>Mean_M</td>
<td>(average of M1 to M7)</td>
<td></td>
<td>3.78</td>
<td>0.0806</td>
<td>TMM: +0.557</td>
</tr>
</tbody>
</table>

### TABLE A.F.4. PROCESS COMPARISONS (TMM AND DGL)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>METHOD $F$ and $Pr &gt; F$</th>
<th>TEAM(METHOD) $F$ and $Pr &gt; F$</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P1</td>
<td>ambiguous</td>
<td>defined</td>
<td>5.25</td>
<td>0.0448</td>
<td>TMM: -0.900</td>
</tr>
<tr>
<td>P2</td>
<td>too many steps</td>
<td>too few steps</td>
<td>4.17</td>
<td>0.0685</td>
<td>TMM: -1.100</td>
</tr>
<tr>
<td>P3</td>
<td>too much time</td>
<td>too little time</td>
<td>0.02</td>
<td>0.8845</td>
<td>TMM: +0.100</td>
</tr>
<tr>
<td>P4</td>
<td>too complex</td>
<td>too simplistic</td>
<td>11.84</td>
<td>0.0063</td>
<td>TMM: -0.900</td>
</tr>
<tr>
<td>P5</td>
<td>unclear starting point</td>
<td>clear starting point</td>
<td>0.22</td>
<td>0.6495</td>
<td>TMM: +1.000</td>
</tr>
<tr>
<td>P6</td>
<td>unclear ending point</td>
<td>clear ending point</td>
<td>0.05</td>
<td>0.8232</td>
<td>TMM: +0.800</td>
</tr>
<tr>
<td>Mean_P</td>
<td>(average of P1 to P6)</td>
<td></td>
<td>3.01</td>
<td>0.1134</td>
<td>TMM: -0.166</td>
</tr>
<tr>
<td>Variable</td>
<td>Left Anchor (-3)</td>
<td>Right Anchor (+3)</td>
<td>METHOD ( F \text{ and } Pr &gt; F )</td>
<td>TEAM(METHOD) ( F \text{ and } Pr &gt; F )</td>
<td>TMM &amp; DGL Grand Means</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>difficult to use</td>
<td>2.17</td>
<td>0.1750</td>
<td>0.48</td>
<td>0.8403</td>
</tr>
<tr>
<td></td>
<td>easy to use</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>inadequate power</td>
<td>8.39</td>
<td>0.0177</td>
<td>0.46</td>
<td>0.8530</td>
</tr>
<tr>
<td></td>
<td>adequate power</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N3</td>
<td>rigid</td>
<td>0.38</td>
<td>0.5518</td>
<td>0.68</td>
<td>0.6994</td>
</tr>
<tr>
<td></td>
<td>flexible</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N4</td>
<td>not usable</td>
<td>0.65</td>
<td>0.4410</td>
<td>1.49</td>
<td>0.2820</td>
</tr>
<tr>
<td></td>
<td>very usable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N5</td>
<td>frustrating</td>
<td>6.23</td>
<td>0.0341</td>
<td>1.73</td>
<td>0.2145</td>
</tr>
<tr>
<td></td>
<td>satisfying</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N6</td>
<td>dull</td>
<td>1.09</td>
<td>0.3232</td>
<td>2.26</td>
<td>0.1227</td>
</tr>
<tr>
<td></td>
<td>stimulating</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N7</td>
<td>terrible</td>
<td>0.64</td>
<td>0.4454</td>
<td>1.64</td>
<td>0.2387</td>
</tr>
<tr>
<td></td>
<td>wonderful</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean N</td>
<td></td>
<td>5.73</td>
<td>0.0403</td>
<td>1.14</td>
<td>0.4187</td>
</tr>
<tr>
<td></td>
<td>(average of N1 to N7)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**FIGURE A.F.1. LEARNABILITY COMPARISONS (TMM AND DGL)**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>METHOD $F$ and $Pr &gt; F$</th>
<th>TEAM(METHOD) $F$ and $Pr &gt; F$</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>difficult to learn easy to learn</td>
<td>12.00 0.0061</td>
<td>0.54 0.8017</td>
<td></td>
<td>TMM: -0.400</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.000</td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>long time to learn short time to learn</td>
<td>45.76 0.0001</td>
<td>1.48 0.2769</td>
<td></td>
<td>TMM: -1.000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
<td></td>
</tr>
<tr>
<td>L3</td>
<td>difficult concepts easy concepts</td>
<td>36.00 0.0001</td>
<td>2.87 0.0604</td>
<td></td>
<td>TMM: -0.400</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.000</td>
<td></td>
</tr>
<tr>
<td>Mean_L1</td>
<td>(average of L1 to L3)</td>
<td>34.48 0.0002</td>
<td>1.27 0.3543</td>
<td></td>
<td>TMM: -0.600</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.033</td>
<td></td>
</tr>
</tbody>
</table>

**Learnability of the METHOD’s PROCESS:**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Left Anchor (-3)</th>
<th>Right Anchor (+3)</th>
<th>METHOD $F$ and $Pr &gt; F$</th>
<th>TEAM(METHOD) $F$ and $Pr &gt; F$</th>
<th>TMM &amp; DGL Grand Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4</td>
<td>too complex too simplistic</td>
<td>15.03 0.0037</td>
<td>1.63 0.2417</td>
<td></td>
<td>TMM: +0.222</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
<td></td>
</tr>
<tr>
<td>L5</td>
<td>unclear starting point clear starting point</td>
<td>10.20 0.0109</td>
<td>0.17 0.9894</td>
<td></td>
<td>TMM: -0.667</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.100</td>
<td></td>
</tr>
<tr>
<td>L6</td>
<td>unclear ending point clear ending point</td>
<td>9.58 0.0128</td>
<td>0.88 0.5673</td>
<td></td>
<td>TMM: +0.333</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.200</td>
<td></td>
</tr>
<tr>
<td>Mean_L2</td>
<td>(average of L3 to L6)</td>
<td>12.17 0.0068</td>
<td>0.57 0.7822</td>
<td></td>
<td>TMM: -0.037</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DGL: +2.133</td>
<td></td>
</tr>
</tbody>
</table>
FIGURE A.F.2. DATA FOR TMM METHOD

| Variable | Mean   | Std Error | T      | Prob>|T| |
|----------|--------|-----------|--------|------|---------|
| M1       | -0.1000000 | 0.5467073 | -0.1829132 | 0.8589 |
| M2       | 1.3000000  | 0.4725816 | 2.7508479  | 0.0224* |
| M3       | 0.6000000  | 0.4988877 | 1.2026756  | 0.2598 |
| M4       | 0.7000000  | 0.3666667 | 1.9090909  | 0.0886 |
| M5       | -0.2000000 | 0.4898979 | -0.4082483 | 0.6926 |
| M6       | 0.7000000  | 0.3666667 | 1.9090909  | 0.0886 |
| M7       | 0.9000000  | 0.3144660 | 2.8619943  | 0.0187* |
| MEAN_M   | 0.5571429  | 0.2722527 | 2.0463429  | 0.0710 |
FIGURE A.F.3. DATA FOR TMM PROCESS
TMM Notation & Formalisms

Size indicates Frequency

| Variable | Mean   | Std Error | T      | Prob>|T| |
|----------|--------|-----------|--------|-----|---|
| N1       | 0.4444444 | 0.6478835 | 0.6859943 | 0.5121 |
| N2       | 0.1111111 | 0.4843221 | 0.2294157 | 0.8243 |
| N3       | 1.3333333 | 0.3726780 | 3.5777088 | 0.0072*|
| N4       | 1.0000000 | 0.4409586 | 2.2677868 | 0.0531 |
| N5       | -0.1111111 | 0.4230985 | -0.2626129 | 0.7995 |
| N6       | 0.3333333 | 0.4409586 | 0.7559289 | 0.4714 |
| N7       | 0.5555556 | 0.2939724 | 1.8898224 | 0.0955 |
| MEAN_N   | 0.5238095 | 0.2896554 | 1.8083889 | 0.1082 |

FIGURE A.F.4. DATA FOR TMM NOTATION & FORMALISMS
FIGURE A.F.5. DATA FOR TMM LEARNABILITY
(DECOMPOSED INTO PROCESS- AND NOTATION-LEARNABILITY)
Using TMM Domains

Size indicates Frequency

| Variable | Mean   | Std Error | T     | Prob>|T| |
|----------|--------|-----------|-------|------|-----|
| D1       | -1.0500000 | 0.3685557 | -2.8489585 | 0.0191* |
| D2       | -0.4000000 | 0.5811865 | -0.6882472 | 0.5086 |
| D3       | 0.2000000  | 0.5120764 | 0.3905667 | 0.7052 |
| D4       | 0       | 0.3333333 | 0     | 0.0000 |
| D5       | -0.7000000 | 0.5174725 | -1.3527289 | 0.2091 |
| MEAN_D   | -0.3900000 | 0.3566667 | -1.0934579 | 0.3026 |

FIGURE A.F.6. DATA FOR USING TMM DOMAINS
Finding TMM Domain Items

Size indicates Frequency

| Variable | Mean    | Std Error | T       | Prob>|T| |
|----------|---------|-----------|---------|------|------|
| DI1      | -0.6000000 | 0.4988877 | -1.2026756 | 0.2598 |
| DI2      | -0.5000000 | 0.4772607 | -1.0476454 | 0.3221 |
| DI3      | 0.2000000  | 0.4666667 | 0.4285714  | 0.6783 |
| DI4      | 0.2000000  | 0.2905933 | 0.6882472  | 0.5086 |
| DI5      | -0.4000000 | 0.3399346 | -1.1766968 | 0.2695 |
| MEAN_Di  | -0.2200000 | 0.2412007 | -0.9121035 | 0.3855 |

FIGURE A.F.7. DATA FOR FINDING TMM DOMAIN ITEMS
Identifying TMM Mappings

Size indicates Frequency

| Variable | Mean | Std Error | T   | Prob>|T| |
|----------|------|-----------|-----|-----|
| MA1      | 0.0  | 0.3944053 | 0   | 1.0000 |
| MA2      | 0.0  | 0.3651484 | 0   | 1.0000 |
| MA3      | 0.5000000 | 0.3073181 | 1.6269784 | 0.1382 |
| MA4      | 0.1000000 | 0.4068852 | 0.2457696 | 0.8114 |
| MA5      | -0.1000000 | 0.3785939 | -0.2641353 | 0.7976 |
| MEAN_MA  | 0.1000000 | 0.2654137 | 0.3767703 | 0.7151 |

FIGURE A.F.8. DATA FOR IDENTIFYING TMM MAPPINGS
Identifying TMM Mapping Knowledge Requirements

![Graph showing the relationship between different factors and their frequency.]

Size indicates frequency

| Variable | Mean   | Std Error | T      | Prob>|T| |
|----------|--------|-----------|--------|-----------------|
| KR1      | 0.60000000 | 0.4268749 | 1.4055639 | 0.1934 |
| KR2      | 0.30000000 | 0.3958114 | 0.7579367 | 0.4679 |
| KR3      | 0.40000000 | 0.2666667 | 1.5000000 | 0.1679 |
| KR4      | 0.40000000 | 0.3399346 | 1.1766968 | 0.2695 |
| KR5      | 0.30000000 | 0.4955356 | 0.6054055 | 0.5599 |
| MEAN_KR  | 0.40000000 | 0.2936362 | 1.3622298 | 0.2062 |

FIGURE A.F.9. DATA FOR IDENTIFYING TMM MAPPINGS KNOWLEDGE REQUIREMENTS
Classifying TMM
Knowledge Requirements
(as Factual, Conceptual, or Procedural)

Size indicates Frequency

| Variable | Mean     | Std Error | T        | Prob>|T|
|----------|----------|-----------|----------|------|
| CK1      | -1.000000| 0.5962848 | -1.6770510| 0.1278|
| CK2      | -0.300000| 0.5174725 | -0.5797410| 0.5763|
| CK3      | -0.700000| 0.4484541 | -1.5609177| 0.1530|
| CK4      | 0        | 0.3333333 | 0        | 1.0000|
| CK5      | -0.800000| 0.5120764 | -1.5622669| 0.1527|
| MEAN_CK  | -0.560000| 0.3721410 | -1.5048062| 0.1666|

FIGURE A.F.10. DATA FOR CLASSIFYING TMM KNOWLEDGE REQUIREMENTS
Determining if Knowledge Requirements are Supported or Unsupported

| Variable | Mean  | Std. Error | T     | Prob>|T| |
|----------|-------|------------|-------|-----|-----|
| S1       | 1.500000 | 0.4013865  | 3.7370466 | 0.0046* |
| S2       | 1.000000 | 0.3651484  | 2.7386128 | 0.0229* |
| S3       | 0.756000 | 0.4166667  | 1.8000000 | 0.1054 |
| S4       | 0.900000 | 0.3785939  | 2.3772174 | 0.0414* |
| S5       | 1.100000 | 0.3785939  | 2.9054880 | 0.0174* |
| MEAN_S   | 1.050000 | 0.2187083  | 4.8009145 | 0.0010* |

FIGURE A.F.11. DATA FOR DETERMINING IF KNOWLEDGE IS SUPPORTED OR UNSUPPORTED (TMM)
Figure A.F.12. Data for synthesizing user interface design requirements using TMM
### Variable Summary for DGL Method

| Variable | Mean   | Std Error | T     | Prob>|T| |
|----------|--------|-----------|-------|------|-------------------|
| M1       | 1.7000000 | 0.4229526 | 4.0193631 | 0.0030* |
| M2       | 1.4000000 | 0.3399346 | 4.1184388 | 0.0026* |
| M3       | 1.5000000 | 0.4013865 | 3.7370466 | 0.0046* |
| M4       | 1.8000000 | 0.2000000 | 9.0000000 | 0.0001* |
| M5       | 0.9000000 | 0.3144660 | 2.8619943 | 0.0187* |
| M6       | 0.7000000 | 0.3666667 | 1.9090909 | 0.0886 |
| M7       | 0.5000000 | 0.3726780 | 1.3416408 | 0.2126 |
| **MEAN_M** | **1.2142857** | **0.2898510** | **4.1893445** | **0.0023** * |

**FIGURE A.F.13. DATA FOR DGL METHOD**
DGL Process

Size indicates Frequency

| Variable | Mean    | Std Error | T       | Prob>|T| |
|----------|---------|-----------|---------|-----|-----|
| P1       | 0.8000000 | 0.4163332 | 1.9215378 | 0.0868 |
| P2       | -0.1000000 | 0.1000000 | -1.0000000 | 0.3434 |
| P3       | 0       | 0.1490712 | 0       | 1.0000 |
| P4       | 0.6000000 | 0.1632993 | 3.6742346 | 0.0051* |
| P5       | 1.3000000 | 0.4229526 | 3.0736306 | 0.0133* |
| P6       | 0.6000000 | 0.4760952 | 1.2602521 | 0.2393 |
| MEAN_P   | 0.5333333 | 0.1606315 | 3.3202294 | 0.0089* |

FIGURE A.F.14. DATA FOR DGL PROCESS

Appendix F: Phase Two Data
DGL Notation & Formalisms

Size indicates Frequency

| Variable | Mean  | Std Error | T     | Prob>|T| |
|----------|-------|-----------|-------|------|
| N1       | 1.7000000 | 0.3958114 | 4.2949748 | 0.0020* |
| N2       | 1.8000000 | 0.2000000 | 9.0000000 | 0.0001* |
| N3       | 1.7000000 | 0.3958114 | 4.2949748 | 0.0020* |
| N4       | 1.4000000 | 0.3399346 | 4.1184388 | 0.0026* |
| N5       | 0.9000000 | 0.2333333 | 3.8571429 | 0.0039* |
| N6       | 0.8000000 | 0.3590110 | 2.2283441 | 0.0528 |
| N7       | 0.8000000 | 0.2000000 | 4.0000000 | 0.0031* |
| MEAN_N   | 1.3000000 | 0.1825121 | 7.1228164 | 0.0001* |

FIGURE A.F.15. DATA FOR DGL NOTATION & FORMALISMS
DGL Learnability

Process
- Difficult to Learn
- Long Time to Learn
- Difficult Concepts

Notation
- Difficult to Learn
- Long Time to Learn
- Difficult Notation

Average Response (Notation)
Average Response (Process)

Size indicates Frequency

| Variable    | Mean   | Std Error | T      | Prob>|T| |
|-------------|--------|-----------|--------|-------|
| L1          | 2.000000 | 0.2981424 | 6.7082039 | 0.0001* |
| L2          | 2.100000 | 0.2768875 | 7.5843087 | 0.0001* |
| L3          | 2.000000 | 0.2581989 | 7.7459667 | 0.0001* |
| L4          | 2.100000 | 0.2768875 | 7.5843087 | 0.0001* |
| L5          | 2.100000 | 0.3785939 | 5.5468407 | 0.0004* |
| L6          | 2.200000 | 0.3687301 | 5.6594533 | 0.0003* |
| MEAN_L_Not | 2.033333 | 0.2650413 | 7.6717591 | 0.0001* |
| MEAN_L_Proc| 2.133333 | 0.3413843 | 6.2490677 | 0.0001* |

FIGURE A.F.16. DATA FOR DGL LEARNABILITY
(DECOMPOSED INTO PROCESS- AND NOTATION-LEARNABILITY)
Using Design Guidelines

![Graph showing relationships between various design attributes and their frequency.]

- Size indicates Frequency

| Variable | Mean    | Std Error | T       | Prob>|T| |
|----------|---------|-----------|---------|------|
| U1       | 1.7000000 | 0.3349959 | 5.0746897 | 0.0007* |
| U2       | 0.9000000 | 0.2768875 | 3.2504180 | 0.0100* |
| U3       | 0.8000000 | 0.3590110 | 2.2283441 | 0.0528 |
| U4       | 0.6000000 | 0.3711843 | 1.6164477 | 0.1405 |
| U5       | 0.7000000 | 0.3666667 | 1.9090909 | 0.0886 |
| MEAN_U   | 0.9400000 | 0.2700617 | 3.4806858 | 0.0069* |

FIGURE A.F.17. DATA FOR USING DESIGN GUIDELINES
Finding Applicable DGL

Size indicates Frequency

| Variable | Mean   | Std Error | T      | Prob>|T| |
|----------|--------|-----------|--------|------|-----|
| F1       | 0.4000000 | 0.4268749 | 0.9370426 | 0.3732 |
| F2       | 0.9000000 | 0.4582576 | 1.9639610 | 0.0811 |
| F3       | 0.6000000 | 0.4000000 | 1.5000000 | 0.1679 |
| F4       | 0.2000000 | 0.3887301 | 0.5144958 | 0.6193 |
| F5       | 0.7000000 | 0.4229526 | 1.6550319 | 0.1323 |
| MEAN_F   | 0.5600000 | 0.3862066 | 1.4500000 | 0.1810 |

FIGURE A.F.18. DATA FOR FINDING APPLICABLE DGL
Selecting Necessary DGLs from Applicable DGLs

Size indicates Frequency

| Variable | Mean   | Std Error | T       | Prob>|T| |
|----------|--------|-----------|---------|------|----|
| S1       | 1.000000 | 0.3944053 | 2.5354628 | 0.0319* |
| S2       | 0.800000 | 0.3265986 | 2.4494897 | 0.0368* |
| S3       | 0.900000 | 0.2768875 | 3.2504180 | 0.0100* |
| S4       | 0.500000 | 0.3073181 | 1.6269784 | 0.1382  |
| S5       | 0.600000 | 0.2666667 | 2.2500000 | 0.0510  |
| MEAN_S   | 0.760000 | 0.2595723 | 2.9278933 | 0.0169* |

FIGURE A.F.19. DATA FOR SELECTING NECESSARY DGLS FROM APPLICABLE DGLS
Synthesizing new UIDs using DGLs
(i.e., Tailoring)

Size indicates Frequency

| Variable | Mean   | Std Error | T     | Prob>|T| |
|----------|--------|-----------|-------|------|
| SU1      | 2.100000 | 0.2768875 | 7.5843087 | 0.0001* |
| SU2      | 1.700000 | 0.3000000 | 5.6666667 | 0.0003* |
| SU3      | 1.300000 | 0.3349959 | 3.8806450 | 0.0037* |
| SU4      | 1.000000 | 0.4472136 | 2.2360680 | 0.0522 |
| SU5      | 1.000000 | 0.4216370 | 2.3717082 | 0.0418* |
| MEAN_SU  | 1.420000 | 0.2723560 | 5.2137648 | 0.0006* |

FIGURE A.F.20. DATA FOR SYNTHESIZING NEW UIDRS USING DGLS (I.E., TAILORING)
General TMM Feedback

Q: "What did you like best about using the analysis METHOD?"

S1 "It breaks down the task adequately"

S2 "Once learned, it's clear to follow and use. The [design] specification can be derived easily from the use of TMM."

S3 "The listing of knowledge required to perform [or map] a task from domain to domain makes identifying unsupported knowledge clear."

S5 "Systematic"

S8 "Seems to be a good concrete way of identifying interface shortcomings. [We] found some problems we didn't see at the beginning. [TMM] made formulating the problem easier."

S12 "The idea of whether something is supported or not supported by user class/interface is helpful to keep in mind."

S13 "TMM's structure gave me confidence that I was addressing all aspects of the problem. The method allowed 'general' requirements to be formed in places where our team could not agree on a specific solution. We were able to apply the basic method partially and derive requirements."

S15 "Make clear distinction between different realms (domains) of user activity. Emphasizes where required knowledge comes from and reveals when the application [interface] is deficient."

S16 "The end-result centered approach. Almost all of the actions of the method are directed towards creating the design requirements."

FIGURE A.F.21. TMM ANALYSTS' RESPONSES ON 'WHAT DID YOU LIKE MOST?'
<table>
<thead>
<tr>
<th>Q: “What did you like least about using the analysis METHOD?”</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>S1</strong> “Sometimes the distinction between computer domains and knowledge domains is not very clear. There is a ‘steep’ learning curve for the method.”</td>
<td></td>
</tr>
<tr>
<td><strong>S3</strong> “Listing all sub-tasks and knowledge is not usually necessary. Problems (i.e., unsupported knowledge) are usually clear early in the process.”</td>
<td></td>
</tr>
<tr>
<td><strong>S8</strong> “... the shades of distinction between the domains were hard to crystallize in my mind.”</td>
<td></td>
</tr>
<tr>
<td><strong>S12</strong> “Most things do not fit specifically into the categories. One can see how to fix stuff immediately, and the formalisms are in the way.”</td>
<td></td>
</tr>
<tr>
<td><strong>S13</strong> “... the method would be very tedious if strict compliance and complete analyses were required. I don’t feel I had a grasp of the use of feedback mappings ...”</td>
<td></td>
</tr>
<tr>
<td><strong>S15</strong> “Too elaborate for simple user problems. Some problems cannot be clearly translated into different domains.”</td>
<td></td>
</tr>
<tr>
<td><strong>S16</strong> “The amount of time that is required to achieve an in-depth analysis. Despite the end-result centering, the detail that is required in the specification is at too low of a level (in fairness, this has been true of every design analysis method I have used.)”</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE A.F.22. TMM ANALYSTS’ RESPONSES ON ‘WHAT DID YOU LIKE LEAST?’**
General TMM Feedback

Q: "What does the Method/Process/Notation not provide?"

S1  "The method does not provide a way for analysts to point out obvious solutions to some of the problems"

S3  "Guidelines for required depth of task breakdown."

S8  "Not easy to express concurrent actions."

S13 "There was not a clear way to state that a feature of an interface was poorly done unless it could be shown that there was some form of unprovided knowledge."

S15 "A means of reporting redundant features. A means of recommending a complete abandonment of bad designs. E.g., I don’t want to say: ‘Highlight elements for longer period of time’ ... I wanted to say ‘This is a moronic way of performing this function. Try again.’"

S16 "A mechanism for expressing interface changes. While not perhaps a goal of the method, the temptation to suggest possible interface changes is too great and should be included."

FIGURE A.F.23. TMM ANALYSTS’ RESPONSES—GENERAL TMM FEEDBACK

TMM Task Description Domains

Q: (1) Are the domain distinctions clear?
Q: (2) Are all necessary levels of abstraction present?
Q: (3) What problems did you have casting tasks and items into specific domains?

S1  (1) "Many times the domains seemed to overlap."

S2  (3) "Computer semantic and syntactic [domain items] get mixed up at times."

S3  (1) "Not always, but usually."

S5  (3) "Distinction primarily, as well as what to include/exclude (i.e., what’s too obvious—if anything to include)"

S13 (1) "UAN and computer syntactic were clear, but computer semantic and problem [domains] gave me troubles in finding the exact level of detail." (2) "Yes, but possibly the problem and computer semantic [domains] could be combined." (3) "... Some tasks seemed to begin in the computer semantic domain ..."

S15 (1) "No—syntactic domain & user action are blurred." (3) "Syntactic and articulation domains can get far too detailed."

FIGURE A.F.24. TMM ANALYSTS’ RESPONSES ON TMM DOMAINS
TMM Task Description Mappings

Q: (1) Are the mappings obvious and clear?
Q: (2) What problems did you have identifying the mappings?
Q: (3) What problems did you have associating user knowledge requirements with each of the mappings?

S2  (1) “Most of the time. There exists 1-1, 1-many, and many-1 mappings. It takes time to identify.”

S5  (2) “Sometimes the gulf of evaluation was difficult to map. (Where to stop or loop back through, etc.)”

S13 (1) “Yes for the syntactic & UAN domains because I understood the domains (and was familiar with them from the HCI class). The mappings from problem to semantic, and semantic to syntactic were a bit more difficult.” (3) “Separating a concept into separate facts. Deciding whether a procedure was actually a WIMPS concept.”

FIGURE A.F.25. TMM ANALYSTS’ RESPONSES ON TASK DESCRIPTION MAPPINGS

TMM User Knowledge Requirements

Q: (1) Are there enough knowledge requirement categories?
Q: (2) What problems did you have identifying user knowledge and their types?

S5  (1) “Yes” (2) “Sometimes tough to distinguish between well known concepts & factual knowledge, etc.”

S8  (2) “hard to know if something is a concept or a fact. E.g.: ‘Buttons should be pressed to activate.’ This is a fact in general, but also a concept about buttons.”

S15 (1) “Yes, too many. This categorization was not useful.” (3) “Knowledge is not so easily categorized. [I] Recommend just Procedural & Factual/Conceptual.”

FIGURE A.F.26. TMM ANALYSTS’ RESPONSES ON KNOWLEDGE REQUIREMENTS
TMM Unsupported Knowledge Analysis

Q: (1) What problems did you have determining knowledge support vs. unsupported?
Q: (2) What characteristics should be in the user class profile to support this analysis?

S1  (1) “...support vs. unsupported was generally easy to determine.”

S5  (1) “Very little.”  (2) “Seems pretty complete.”

S10 (1) “Being [un]supported did not appear to be a prerequisite for being a problem. I expected that it would. If the unsupported mapping appeared between the problem and [computer] semantic domain it seemed to be the most important/serious problem. I.e., conceptual/metaphor problems are more difficult to overcome than physical and semantic problems.”

S16 (1) “None, TMM allowed for very clear identification.”

FIGURE A.F.27. TMM ANALYSTS’ RESPONSES ON KNOWLEDGE ANALYSIS

TMM Synthesis of Design Requirements

Q: (1) Does TMM provide enough information to synthesize design requirements?

S1  “TMM could provide guidelines to the designers for incorporating the requirements.”

S2  “It generally does provide enough information.”

S8  “I think enough was provided. It was hard to express [requirement]s without telling designer what to do, however.”

S13 “… We often were tempted to include actual ‘design’ into these requirements.”

S15 “Yes—knowledge gaps are clearly revealed.”

S16 “The only information that TMM lacks is a mechanism for grouping of abstractions. This would allow an analyst to work at a higher level in their evaluation.”

FIGURE A.F.28. TMM ANALYSTS’ RESPONSES ON SYNTHESIS OF UIDRS
TMM Life Cycle

Q: (1) Are there any stages left out of the TMM analysis life cycle?
Q: (2) Are the stages distinct enough?

S5  (1) “I don’t think so. It seems to map pretty well inherently with other models.” (2) “I think so.”

S8  (1) “Seems okay.” (2) “Yes. Definitely.”

S10 (1) “Yes. Maybe a ‘road-block’ that defines the end-system (delivery platform) so you can iterate in/out design requirements to either fit into the delivery platform or expel them from the process” (2) “Yes, I like the inside/outside paths for information flow since systems analysis results are all part of all modules.”

FIGURE A.F.29. TMM ANALYSTS’ RESPONSES ON TMM LIFE CYCLE

TMM Training

Q: (1) Was the training adequate?
Q: (2) What should be added/removed to/from the training?
Q: (3) What topics should be discussed further?

S5  (2) “Examples & Non-examples—especially with determining domains and knowledge types.”

S13 (2) “[Add more] examples of procedural decomposition.”

S15 (1) “Yes.” (2) “Less pompous jargon, better use of the English language (recommended for HCI in general!) Many concepts superfluous.”

S16 (1) “No, more is needed. Although TMM is not overly complex a few more samples is needed.”

FIGURE A.F.30. TMM ANALYSTS’ RESPONSES ON TRAINING
General DGL Feedback

Q: "What did you like best about using the analysis METHOD?"

S17 "Simple. Logical Ordering. Bruit\textsc{s}AM made guidelines more usable."

S18 "The flexibility to tailor guidelines to the specific situation of use. And the full representation power of the English Language."

S19 "It was easy to formalize some of the weaknesses in the user interface based on the guidelines."

S22 "[Using DGL] becomes easier with practice. The list of guidelines is well defined making it easy to understand them and decide whether they are applicable or not."

S24 "Easy to understand and use."

S25 "Using Bruit\textsc{s}AM was very easy accompanied with the guideline title pages. both are complements to each other. Just using one without the other would have been a very frustrating process. The power of the search and ‘see references’ were more than enough to reduce time and find the adequate guideline."

S26 "Ease of learning. Ease of use."

S28 "Good and complete grouping of analysis guidelines. Easy to generate requirements. Easy use of notations and definitions. Simple process to follow."

S29 "[We] were able to quickly identify problems and solutions, and support decisions with previous work. Could be very useful working with management."

FIGURE A.F.31. DGL ANALYSTS’ RESPONSES ON ‘WHAT DID YOU LIKE MOST?'}
General DGL Feedback

Q: “What did you like least about using the analysis METHOD?”

S17 “Not enough cross referencing. Not up to date on current interface features, so [guidelines] required some adaptation.”

S18 “The Smith & Mosier guidelines are an inadequate base for generating UINDRs for WIMP interfaces. (I don’t think this is necessarily a problem with guidelines in general, i.e., a set of good guidelines for HyperCard style applications would probably have worked very well.)”

S19 “It was difficult sometimes to pinpoint the exact guideline/s by analysis as many are overlapping—yet, have subtle differences.”

S22 “Too many guidelines to have to sift through. [It] would have been easier if we could have just formulated a requirement without have to look at the guidelines (though maybe they would not have been good enough—I don’t know.)”

S24 “Inaccuracy of finding ‘right’ guidelines to fit the problem. Not sure of completeness of results.”

S25 “Using BruitSAM, the gather function really didn’t work as expected. I anticipated that only the guidelines that I chose would appear. Instead several more along with duplications appeared. Also, there were situations in which I couldn’t find the guideline that I was looking for. For example, in one interface there was a temporal situation in which the display wasn’t highlighted long enough. I couldn’t quite find a specific guideline that addressed this problem.”

S26 “Easy to overlook guidelines which may exactly match requirement. Search for guidelines not always exhaustive/complete.”

S28 “Sometimes its’ hard to find the related requirements, such as information groups about tasks, usability, etc.”

S29 “Restriction to use of S&M (or other) guidelines limited problems that could be addressed (or how they could be addressed) and potentially limited recommendations for solutions.”

S30 “[It] becomes a rote task. Identifying specific problems/DG’s becomes repetitive. Seemed to use some DG’s many times.”

FIGURE A.F.32. DGL ANALYSTS’ RESPONSES ON ‘WHAT DID YOU LIKE LEAST?’
General DGL Feedback

Q: "What does the Method/Process/Notation not provide?"

S17 "Often too general. [DGL] does not cover all new interface features."

S18 "In this case, it did not provide an appropriate set of guidelines for the type of interfaces being analyzed."

S19 "It does not provide a consistent feedback on the amount of effort needed to redesign and re-implement as that is dependent on the application domain knowledge of the analysis method user."

S24 "A suitable stopping point."

S25 "It provides for very general guideline abuses. However, when it came to having a specific problem the method/process/notation left me open to various other interpretations of other guidelines to cover my specific problem area."

S26 "[The] process does not provide complete/exhaustive search for guidelines."

S29 "Some flexibility in the selection of guidelines. Perhaps this was mostly because of the use of S&M and its age."

S30 "Allowing general extensions to the DG’s, as opposed to tailoring to a specific user interface."

FIGURE A.F.33. DGL ANALYSTS’ RESPONSES—GENERAL DGL FEEDBACK
DGL and Task Descriptions

Q: (1) How do design guidelines help task description/decomposition?
Q: (2) How do design guidelines hinder task description/decomposition?

S17  (1) "Guidelines provide only a general task breakdown." (2) "Some tasks cross guideline boundaries"

S18  (1) "I'm a bit confused by this question. My experience using design guidelines here and elsewhere was directed at the constructional domain (the interface design) not the behavioral domain (task analysis)?" (2) "(See above) I'd say the guidelines might help task analysis a little and they certainly would not hinder it."

S19  (1) "The design guidelines help in task decomposition by making sure that the tasks are consistent and usable. This is turn helps in the coding phase also." (2) "The more one gets carried away with design—it could effect the implementers competency. This may or may not be an issue."

S22  (1) "Design guidelines as given by S&M do not help task description/decomposition. Using the design guidelines to evaluate an interface and to come up with UIDRs helps us come out with a list of violations, the fixing of each of these violations can be considered as a very general task. Detailed decomposition of that task needs to be done using other methods. To conclude, DGs do not help task decomposition very much." (2) "There are too many DGs."

S24  (1) "They help task description by defining interface laws that each task must abide to." (2) "Too many design guidelines hinder a task description down to one possible description which may not be the 'best' way. There is no way to systematically rank which guidelines are more important/less important."

S25  (1) "They help in the following ways: no surprises to the user; add for conformity to what the user is used to and probably anticipating; and they serve as a guide to stay on a certain path that leads to clarity and reduces confusion." (2) "If someone wanted to be very creative with an interface the rigid guidelines would probably not support their end product."

S26  (1) "It helped: meet user expectation; keep user action consistent with functions/tasks; provide a method for task completion." (2) "Guidelines have to be filtered—those affecting user perception/expectation and ease of use. Rarely, but sometimes, conflicting guidelines."

S29  (1) "Didn't feel the connection here was very direct. A little confused about this question because we didn't do task analysis."

S30  (1) "Knowing guidelines makes identification problems easier. Often predicted cause of problems during system exploration." (2) "Many DG's seem too specific for the problems encountered. A single broader DG would suffice for many."

FIGURE A.F.34. DGL ANALYSTS' RESPONSES ON DGL AND TASK DESCRIPTION
DGL Synthesis of Design Requirements

Q: (1) Does DGL provide enough information to synthesize design requirements?

S17 "Guidelines are not as specific—often [they] must be modified quite a bit to derive the UIDR."

S18 "I think that they are certainly useful in the process. Background knowledge in the application domain and in the domain of design are certainly required prerequisites to deriving requirements. (In other words, I think that the background knowledge is more important than the guidelines)."

S19 "Yes, the design guidelines provide enough information if not too much."

S22 "Yes, they do. Because they are very precise."

S24 "Too much information is the case. A system of ranking Design Guidelines is needed."

S25 "No, because a lot of interpretation is left to the developer. The guidelines are descriptive but not detailed. They really wouldn’t answer the question of whether something is correct or lacking more clarity without some usability testing. To try and compensate by having very detailed information would be just as bad, because that would stifle creativity."

S26 "[DGLs] provide most of the information required. Additional requirements [could be] more specific context for guidelines. Since guidelines do not have priority levels burden of deciding relevant/critical ones is on the interface designer."

S29 "No they do not at all. Task analysis is also required."

S30 "In general, yes. There are some specific cases where information is needed, with respect to relatively new terms (e.g., buttons). Many DG’s seemed to have a very limited range of application—one wonders if they are necessary. Too specific maybe."

FIGURE A.F.35. DGI, ANALYSTS’ RESPONSES ON DGL SYNTHESIS OF UIDRS
DGL Life Cycle

Q: (1) Are there any stages left out of the DGL analysis life cycle?
Q: (2) Are the stages distinct enough?

S17 (1) “Verification of completeness.” (2) “Yes.”

S18 (2) “Yes, I suppose. Stages in life cycles like this are useful for pedagogical purposes, but in skilled actual design, of course, the boundaries are blurred.”

S19 (1) “No.” (2) “Yes.”

S24 (1) “There should be a better defined stopping point. Some criteria should be developed to stop the process when all the design guidelines have been found.” (2) “Yes.”

S25 (1) “They only stage left out would be re-testing because even when those changes are implemented whose to say that they weren’t other factors contributing to the same problem?” (2) “They are quite distinct. I was able to follow each stage without any difficulty.”

S26 (1) “When alternative guidelines can solve the problem, rating them by difficulty (individually) [should be supported].”

S29 “[It] wasn’t entirely clear when search for guidelines was complete.”

S30 (1) “Don’t think so.” (2) “Yes, as much as can be expected.”

FIGURE A.F.36. DGL ANALYSTS’ RESPONSES ON DGL LIFE CYCLE
DGL Training

Q: (1) Was the training adequate?
Q: (2) What should be added/removed to/from the training?
Q: (3) What topics should be discussed further?

S17 (1) “Yes.”

S18 (1) “More emphasis on the synthesis of UIDR’s and corresponding less general introduction to guidelines (especially the Motif & Apple guidelines we did not use) would have been more effective.”

S19 (1) “Yes.” (2) “An actual example on line computer system.”

S24 (1) “Yes.” (2) “Nothing.”

S25 (1) “More than adequate, just the familiarization with the additional guides was enough including S&M.” (2) “It was quite adequate.” (3) “Going over the overall outcome. For each scenario what were the total guidelines that were applied to each scenario.”

S29 (1) “Yes.” (2) “A more clear delineation of steps [is needed].”

S30 (1) “Yes, it would have been better if the DG search system was better. I had a general idea of types of DG’s; but most DG’s were not known. ⇒ I found many DG’s applicable to earlier tasks/systems that I couldn’t think of/find on the fly.”

FIGURE A.F.37. DGL ANALYSTS’ RESPONSES ON TRAINING
The following figures are comparisons of parametric and non-parametric analyses of the data. First there is a summary of three different analyses across the Likert scale data comparing TMM and DGL. The columns represent:

TABLE A.F.6. COMPARISONS OF TMM AND DGL USING PARAMETRIC AND NONPARAMETRIC ANALYSES

- Column 1 = Data Variable, <var>
- Column 2 = Wilcoxon Rank Sum (2 sample test)
- Column 3 = ANOVA1 (Model <var> = METHOD)
- Column 4 = ANOVA2 (Model <var> = METHOD GROUP(METHOD))

All data shown are p values

<table>
<thead>
<tr>
<th>&lt;var&gt;</th>
<th>W-RS</th>
<th>ANOVA1 Method</th>
<th>ANOVA2 Method</th>
<th>Group (Method)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>0.0183*</td>
<td>0.0179*</td>
<td>0.0266*</td>
<td>0.4960</td>
</tr>
<tr>
<td>M2</td>
<td>0.6076</td>
<td>0.8655</td>
<td>0.8455</td>
<td>0.1893</td>
</tr>
<tr>
<td>M3</td>
<td>0.1878</td>
<td>0.1769</td>
<td>0.2446</td>
<td>0.8375</td>
</tr>
<tr>
<td>M4</td>
<td>0.0152*</td>
<td>0.0169*</td>
<td>0.0236*</td>
<td>0.4569</td>
</tr>
<tr>
<td>M5</td>
<td>0.1015</td>
<td>0.0750*</td>
<td>0.0447*</td>
<td>0.1402</td>
</tr>
<tr>
<td>M6</td>
<td>0.9999</td>
<td>1.00</td>
<td>1.00</td>
<td>0.6397</td>
</tr>
<tr>
<td>M7</td>
<td>0.4532</td>
<td>0.4228</td>
<td>0.1877</td>
<td>0.0077</td>
</tr>
<tr>
<td>P1</td>
<td>0.0379*</td>
<td>0.0295*</td>
<td>0.0448*</td>
<td>0.5740</td>
</tr>
<tr>
<td>P2</td>
<td>0.0598</td>
<td>0.0373*</td>
<td>0.0685</td>
<td>0.7565</td>
</tr>
<tr>
<td>P3</td>
<td>0.7401</td>
<td>0.8664</td>
<td>0.8845</td>
<td>0.8538</td>
</tr>
<tr>
<td>P4</td>
<td>0.0015*</td>
<td>0.0019*</td>
<td>0.0063*</td>
<td>0.6421</td>
</tr>
<tr>
<td>P5</td>
<td>0.9999</td>
<td>0.6980</td>
<td>0.6495</td>
<td>0.1637</td>
</tr>
<tr>
<td>P6</td>
<td>NA</td>
<td>0.8180</td>
<td>0.8232</td>
<td>0.5740</td>
</tr>
<tr>
<td>N1</td>
<td>0.1795</td>
<td>0.1088</td>
<td>0.1750</td>
<td>0.8403</td>
</tr>
<tr>
<td>N2</td>
<td>0.0079*</td>
<td>0.0038*</td>
<td>0.0177*</td>
<td>0.8530</td>
</tr>
<tr>
<td>N3</td>
<td>0.4412</td>
<td>0.5117</td>
<td>0.5518</td>
<td>0.6994</td>
</tr>
<tr>
<td>N4</td>
<td>0.7029</td>
<td>0.4773</td>
<td>0.4410</td>
<td>0.2820</td>
</tr>
<tr>
<td>N5</td>
<td>0.0682</td>
<td>0.0467*</td>
<td>0.0341*</td>
<td>0.2145</td>
</tr>
<tr>
<td>N6</td>
<td>0.3107</td>
<td>0.4193</td>
<td>0.3232</td>
<td>0.1227</td>
</tr>
<tr>
<td>N7</td>
<td>0.5587</td>
<td>0.4934</td>
<td>0.4484</td>
<td>0.2387</td>
</tr>
<tr>
<td>L1</td>
<td>0.0032*</td>
<td>0.0011*</td>
<td>0.0061*</td>
<td>0.8017</td>
</tr>
<tr>
<td>L2</td>
<td>0.0004*</td>
<td>0.0001*</td>
<td>0.0001*</td>
<td>0.2769</td>
</tr>
<tr>
<td>L3</td>
<td>0.0014*</td>
<td>0.0003*</td>
<td>0.0001*</td>
<td>0.0604</td>
</tr>
<tr>
<td>L4</td>
<td>0.0074*</td>
<td>0.0034*</td>
<td>0.0037*</td>
<td>0.2417</td>
</tr>
<tr>
<td>L5</td>
<td>0.0053*</td>
<td>0.0008*</td>
<td>0.0109*</td>
<td>0.9894</td>
</tr>
<tr>
<td>L6</td>
<td>0.0105*</td>
<td>0.0054*</td>
<td>0.0128*</td>
<td>0.5673</td>
</tr>
</tbody>
</table>

As expected, the ANOVA models are more sensitive.
Next, 10 one-sample Likert scales were randomly selected for comparing the t-test results against the Wilcoxon Signed Rank test (this analysis was performed by hand, so p values only exist for t-tests results).

TABLE A.F.7. 10 RANDOMLY SELECTED LIKERT SCALES COMPARING PARAMETRIC AND NONPARAMETRIC ANALYSES

<table>
<thead>
<tr>
<th>&lt;var&gt;</th>
<th>Wilcoxon Signed Rank</th>
<th>t-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>Reject H0, Accept H1</td>
<td>0.0191</td>
</tr>
<tr>
<td>D2</td>
<td>Fail to reject H0</td>
<td>0.5086</td>
</tr>
<tr>
<td>D3</td>
<td>Fail to reject H0</td>
<td>0.7052</td>
</tr>
<tr>
<td>D4</td>
<td>Fail to reject H0</td>
<td>1.0000</td>
</tr>
<tr>
<td>D5</td>
<td>Fail to reject H0</td>
<td>0.2091</td>
</tr>
<tr>
<td>S1</td>
<td>Reject H0, Accept H1</td>
<td>0.0046</td>
</tr>
<tr>
<td>S2</td>
<td>Fail to reject H0</td>
<td>0.0229</td>
</tr>
<tr>
<td>S3</td>
<td>Fail to reject H0</td>
<td>0.1054</td>
</tr>
<tr>
<td>S4</td>
<td>Fail to reject H0</td>
<td>0.0414</td>
</tr>
<tr>
<td>S5</td>
<td>Reject H0, Accept H1</td>
<td>0.0174</td>
</tr>
</tbody>
</table>
APPENDIX G: UIDRs

Reproduced within this appendix are all 32 grouped UIDR sets generated in phase two and evaluated in phase three. Each UIDR set is identified by the interface design, usability problem and the associated synthesis method.

Set 1 = UIDRs synthesized using TMM

Set 2 = UIDRs synthesized using DGL
UIDR SET COMPARISON SHEET: COMPARISON NUMBER 1

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>IceDragon's IMTS 4.0 (Music Tracking System)</td>
<td>Sort contents of Media Database by Artist</td>
<td>User unable to perform task within usability performance specification ... difficulty selecting buttons ... USER ERROR: user selected Sort ICON (not the button Artist)</td>
</tr>
</tbody>
</table>

---

Set 1

1. Allow for a larger space to make sort selection
2. Allow access to any sort function with one click (e.g., use [SORT] pop-down menu for choice—plus designers can add more choices to sort without redesigning interface
3. Don't use icons unless they are hot
4. *Reverse:* Use icons that are accessible and functional
5. Clearly communicate that the sort by (album, artist, label) are buttons to the user
6. Communicate through feedback that a sort is complete
7. Identify mechanism for checking sort accuracy
8. Provide entire database or multiple entry viewing mechanisms
9. Clarify button operation
10. Separate buttons to reflect meanings
11. Communicate difference between icons & buttons
12. Make use of icons consistent (sometimes they're used as part of a button other times as just a label)

Set 2

1. Make menu items distinct from display items
2. Clearly indicate control buttons
3. Use the button symbol consistently. It is not clear that the author/album/... areas are buttons and can be selected
4. Make the label and picture distinct from a button
5. The control elements should be displayed more distinctly and consistently (they should be same size and shape)
6. The control elements should be grouped logically
7. Some icons when depressed have actions associated with them while other don't—be consistent
8. Highlighted icons send a false signal to the user when there is no action behind them
9. There should be some form of indication when a task is completed
**UIDR Set Comparison Sheet: Comparison Number 2**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>IceDragon's IMTS 4.0 (Music Tracking System)</td>
<td>Enter Don McLean's Single American Pie into system</td>
<td>User unable to perform task within usability performance specification ... confusion in terminology ... (multiple use of term Single)</td>
</tr>
</tbody>
</table>

### Set 1

1. Reorganize the media selection area
2. Possibly use more segmenting (pop-up menus?)
3. Communicate that “New Media” represent the creation of a new entry
4. Communicate that Media type buttons are modifiable list entries to the user
5. Communicate that the number of media type buttons are also modifiable lists
6. Need to clarify meanings of attributes
7. Group attributes logically
8. Communicate hierarchy of media buttons and their formats
9. Make explicit that media/format buttons are set on entry (i.e., they are not search criteria)
10. Make explicit the Five implicit functional groupings on the right side of interface

### Set 2

1. Further describe “single” in context (i.e., cassette single, 7” single)
2. Repress items that are not relevant to current selection
3. Provide hierarchic menus for sequential selection
4. Organize and label hierarchical menus (this would group single under LP | Cassette | CD)
5. Display the record type options in logical groups
6. The wording of “Single” and “12” Single” are not distinct from each other
7. Data should be grouped logically by function w/appropriate hierarchy
8. Task requirements should be explicit to the user (the classification is not clear for Single—may be cassette or CD)
9. Grouping the different selections of music format (e.g., cassette, LP, CD) should be separate and distinct with some logical order
# UIDR SET COMPARISON SHEET: COMPARISON NUMBER 3

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>IceDragon’s IMTS 4.0 (Music Tracking System)</td>
<td>Enter Rod Stewart’s Storyteller Triple CD Box Set into system</td>
<td>User unable to perform task within usability performance specification ... difficulty adding new media ... USER ERROR: user entered objects before (New Media) action</td>
</tr>
</tbody>
</table>

## Set 1

1. Suggestion: Once a card is filled by traversing to a different card, all fields are locked. If a user tries to edit any field, a dialog box should warn the user of the potential of overwriting or replacing information of a piece; the box contains buttons to “New Card”, “Edit”, and “Cancel”. An “Edit” button on the card showing a locked icon, when pressed, it will be unlocked.
2. Provide an understanding to the user that any entry information may be changed at any time, or do not allow
3. Identify new creation mechanism to the user
4. Identify blank DB entries to user, do not allow multiple new record creations that are empty
5. Groups of attributes should be made more clear
6. Make clear that changes made in text are permanent
7. Communicate procedure of create entry before edit
8. Provide knowledge that new entries should only be make in blank cards
9. Make it apparent whether you are editing an existing card or a new one
10. Make it apparent that there are other cards existing and that others can be added

## Set 2

1. Do not allow user to enter data unless in new/modify mode
2. Display user modes for data entry/modification
3. Insure that the interface will not allow user to accidentally overwrite existing records
4. Allow users to undo accidental changes to a record
5. Provide an option that allows users to change data
6. Data already entered should be protected from change
7. There should be explicit action from the user to change the data so s/he knows when the data is being changed (e.g., there should probably be a button to turn editing on/off for a particular entry)
8. The “New” button should be changed to “Add”
9. There should be some indicator of the path that the user should take when adding new media
10. There should be no surprises when trying to Add new media even if the user types in the first and then tries to add
UIDR Set Comparison Sheet: Comparison Number 4

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>IceDragon’s IMTS 4.0</td>
<td>Exploratory usage</td>
<td>User had difficulty with Button objects: <em>Hide</em> and <em>Show Buttons</em> and palette buttons for <em>Compact</em>, <em>Sort</em> (Album, Artist, Label), <em>About</em>, <em>Media</em> (New, Delete), and <em>Search</em> ... inconsistent button usage</td>
</tr>
</tbody>
</table>

Set 1

1. Suggestion: Change "Hide" & "Show" palette buttons into a toggle button w/ button name changing dynamically
2. Suggestion: Fix inconsistency on buttons of the palette. Pop-up a dialog w/ choices when clicking on an icon that does nothing now or use a pop-up menu for these, or give more button icons
3. Clearly communicate to the user which interface elements are buttons, labels, or lists
4. Communicate to the user the Palette's location, state, and operations
5. Implementation of controls is inconsistent. Explain their operation/make more consistent. Add Help
6. Show/Hide palette unnecessary/misleading. Explain
7. Identify what a Palette is
8. Communicate relationship among Show/Hide and Palette
9. Provide knowledge that sub-tasks are individual buttons and their icons are not buttons
10. Have the palette available to user at all times
11. Make use of palette icons consistent (either make all buttons or none buttons)
12. Make feedback from palette functions consistent (there is completely different feedback for each one)

Set 2

1. Make buttons distinctive
2. Provide more informative labels for buttons and palette
3. Use familiar wording
4. Give user feedback on null search
5. Only offer available options (gray out/hide available options)
6. Group logically related options
7. Display a continuous indication of current sort field
8. The "About" button should say "About Software" (gets confusing because we thought it meant about album)
9. The "About" button does not logically belong in the palette
10. Buttons should all look the same, be the same size, and be distinctive in position and/or format
11. All forms for control should be in consistent format
12. There needs to be some form of help available
**UIDR Set Comparison Sheet: Comparison Number 5**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic Table of Elements</td>
<td>Identify (list) all Noble Gasses</td>
<td>User unable to perform task within usability performance specification ... User missed highlight (too brief) of Noble Gasses after following correct path through General Info Button</td>
</tr>
</tbody>
</table>

### Set 1

1. Rename “General” to something more descriptive
2. Allow access to these types of sorts [in General field] in conjunction with other “sorts”
3. Do not “time-out” Noble Gas highlight
4. De-highlight only when selecting other “sort” and then select icons of new “sort”
5. Communicate functionality of “General Info” button to user
6. Communicate how to maintain highlighting (e.g., holding down mouse button is unclear)
7. Make clear how to reach groups of elements
8. Provide mechanisms for viewing element groups over extended [time] periods
9. Communicate how to keep gasses highlighted for as long as user wants
10. Give user control over <Noble gas cells highlight> function (e.g., click to de-highlight), or
11. Have interface perform <Noble gas cells highlight> function for a longer time duration

### Set 2

1. Leave selection of element types highlighted until user clicks on the icon again
2. Mention in Help that selection remains highlighted while button is depressed
3. Let user specify when a task is completed, i.e., allow user to control when the list of noble gases de-highlighted
4. The user should be able to pace and control the transaction sequencing by explicit action (rather than requiring the user to keep pace with computer processing)
5. Update the data at a rate appropriate to human perceptual capabilities for that kind of data change (i.e., the data is displayed so short that the user cannot visualize or make use of it)
6. Avoid temporal feedback from a user’s action (a user might miss some feedback if it is displayed only temporally)
**UIDR SET COMPARISON SHEET: COMPARISON NUMBER 6**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic Table of Elements</td>
<td>Identify (list) all</td>
<td>User unable to perform task within usability performance specification ... USER ERROR: User Identified Elements #58-71 as Lanthanides (thus missing Element #57)</td>
</tr>
<tr>
<td></td>
<td>Lanthanides</td>
<td></td>
</tr>
</tbody>
</table>

**Set 1**

1. Suggestion: Add a button in “General” section to include “Lanthanides”
2. Be consistent (why some group uses symbol “∅”, “∗” while others appear after clicking on a button?)
3. Communicate that Lanthanum is a Lanthanide to user at appropriate time (e.g., when user attempts to select Lanthanide interface label).
4. Provide clear indication of Lanthanide group; current notation on the display for row of Lanthanides is misleading.
5. Make information about Lanthanides more easily available (as with the Noble Gas button)
6. “Lanthanide ∅” and “Actinides ∗” labels shouldn’t look like row labels for bottom 2 rows
7. Make “General” button highlight format also include Lanthanides and Actinides, i.e., make identification of subsets of elements consistent

**Set 2**

1. Treat Lanthanides and Actinides as selections which highlight all element members (as works with Noble Gases)
2. Add some supplementary text to emphasize that La is a member of the Lanthanides group
3. In order to direct the users attention to La as part of the Lanthanides group, use a more distinguishing highlighting feature
4. Labels should have consistent meanings and format—always use same label to indicate the same kind of entry (Lanthanides point to elements #58-71), but use “∅” to find #57
5. Related labels should be grouped together so that users can find them easily
6. Labels should be consistently used (e.g., elements #58-71 should all have “∅” on them.)
7. Group elements in logical order when displaying graphical data, or group together by a distinct visual trait
### UIDR Set Comparison Sheet: Comparison Number 7

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic Table of Elements</td>
<td>Change Boiling Point &amp; Melting Point (of any element) to Kelvin</td>
<td>User unable to perform task within usability performance specification ... difficulty getting help on changing units ... difficulty selecting buttons</td>
</tr>
</tbody>
</table>

---

#### Set 1

1. Make the pop-down menu’s visible by use of:
   1) making the °C a button
   2) adding a pop-down menu icon [ °C P ]
2. Allow user to set preference for temp. type for one or all elements (i.e., change °C to K for all elements)
3. Communication unit conversion mechanism to user
4. Communicate help mechanism more clearly to the user
5. The help for changing units requires the user to hold the mouse button down—this is not the case with help on other topics—make "Help" consistent
6. Make explicit the data item buttons
7. Make explicit that press and hold also selects a button (in help or data screen for certain items), or don’t require press and hold for selection
8. Make [?] icon mean same thing on all screens
9. Give users identification of what information [?] icon leads to
10. Make information on “Instructions” card consistent: some buttons are just automatically described and not usable; other are only described when used

#### Set 2

1. Provide a list of available user commands/options
2. Provide task oriented help on the element display screen
3. Denote selectable/changeable options on element display screen
4. Clearly indicate control buttons
5. Indicate imbedded menus
6. The button to get help for changing units should not act differently from other buttons on that screen
7. The labels should be distinguished from buttons—in particular the boiling point button
8. Control entries should be distinct from text. (In help on changing units, the help should be text not menu selections; in using buttons to change boiling point, the buttons should not be text.)
9. The control information should be distinctively displayed to the user in position and/or format (e.g., should be an explicit button or control to change units)
10. Don’t surprise the user by hiding functionality—always display text that can be clicked on by bold facing or underlining it
11. Follow standard feedback procedures
### UIDR SET COMPARISON SHEET: COMPARISON NUMBER 8

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Periodic Table of Elements</td>
<td>List all elements that exist as a GAS at 26°C</td>
<td>User could not perform task within the usability performance specification of 3 Minutes</td>
</tr>
</tbody>
</table>

### Set 1

1. Need to be able to sort on Boiling Pt. and be able to specify a temperature to see the 1st elements that meets the criteria
2. Add temperatures and sort capability to the scrolling list (show only the ones that meets the searching criteria)
3. Communicate sort mechanism location to the user at the table display
4. Identify the sorted navigation method
5. Give better explanation of sorting and viewing sorted list
6. Make sorting concepts understandable to teens
7. Make existence of sort function known and location of sort button clear
8. Make result of sort explicit
9. Make linear scan understandable, clear.
10. Shorten sequence of steps needed to perform task
11. Provide shortcut linear search function instead of just “Next” / “Previous” (one at a time)
12. Link something to “Solid” / “Liquid” / “Gas” buttons or make them not look like buttons

### Set 2

1. Provide a list of options always available
2. Provide help for all tasks including sort
3. Use natural language query formation
4. Provide an easier way to answer given query or provide adequate guidance
5. Add a facility to find elements by attribute values such as boiling point
6. Dialogue/Interface should match the task
7. Query/Search should exist and be more flexible and provide aids to the user
8. Menu options (sorting, next, prev) should be grouped together and should be labeled more clearly
9. Menu options should be distinct from data display (“Sort” and “Gas”/“Solid”/“Liquid” look the same on the screen)
10. Data access should be accomplished by minimal user actions and result-set data should be grouped together in an easy-to-access list of elements
<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titus Andronicus</td>
<td>Using the Glossary to find gramercy</td>
<td>The user could not use the glossary to find definition of a word (say, gramercy) within the usability performance specification of 30 Seconds (user could not even get to glossary)</td>
</tr>
</tbody>
</table>

**Set 1**

1. Need clear wording in “Help” text to know finding a glossary item. The complex steps of actually going to a glossary and getting the right item is non-trivial.
2. Regular Find (rectangle box surrounding the item)—cannot find glossary while highlighted Find does. Needs a more consistent way.
3. Provide direct route to glossary, e.g., add glossary icon to screen cards and add glossary icon to 1st card
4. Communicate to the user (more clearly) that there is a glossary
5. Provide knowledge that only special words (i.e., underlined) are in glossary
6. Communicate that *underlined* results of searches can be found in the glossary
7. Identify the glossary search sub-function of the find operation to the user
8. Provide information to user on how to locate word in glossary (the 4 steps)
9. Double-clicking selects words
10. Give user knowledge of what return key does
11. Fill in instructions for users on help screen (don’t use passive voice, e.g., “is looked up”)

**Set 2**

1. Separate glossary from “Find” function
2. Label Icons or use intuitive icons
3. Minimize user actions—add a glossary button
4. Controlling logic for going to the glossary from a word should require fewer steps (such as a double click)
5. Buttons to search and go to the glossary should be distinct
6. Provide a standard help button that is available on every card
7. User should be able to ignore blanks in command entry.
8. In searching for a word, upper and lower case should be equivalent.
9. The basic commands w / appropriate command guidance (for glossary) should be available to the user (in this case, the user cannot directly find the glossary)
10. Display all functionality of the system in menu options
11. Describe all functionality in on-line help system—provide clearer help for glossary
<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titus Andronicus</td>
<td>Answer the question: What news did ( \text{&amp;} \text{milius} ) bring in his first appearance?</td>
<td>User could not locate ( \text{&amp;} \text{milius} ) within the usability performance specification of 1 Minute (e.g., using the Find operation)</td>
</tr>
</tbody>
</table>

**Set 1**

1. Re-label "Find" icon—i.e., use "Find Text" not eyeball—i.e., make Eyeball = Find explicit
2. Rewrite "Find" dialog box text to be more descriptive (i.e., finding next occurrence not just find)
3. Allow user to understand search occurs forward not necessarily from the beginning (depends on where you are in stack)
4. Allow Find from 1st introductory screen
5. Communicate to the user how to enter from the keyboard any "special" characters that occur in the script—i.e., make ability to type in special characters explicit
6. Communicate to the user in the documentation that a partial string search can be performed
7. Provide an understanding of the operation of the search direction
8. Identify the locations of the help information for the search
9. Character ligatures like "\E" should be supported—i.e., "\E = AE".
10. Searches should allow partial words—i.e., have the search function follow standard protocol, finding any string (e.g., "milius") not just whole words
11. Searches should be case-insensitive
12. Find command is not explicit
13. Make option to: (i) highlight and then find, or (2) enter word into find buffer
14. Make user aware that "\E" and "AE" are different (or else allow searches to go either way)

**Set 2**

1. Do not require users to enter characters which are not normal keys—it is hard to find special character to search on
2. Allow wildcards within searches
3. Allow special characters in string searches
4. Allow entry of special characters or accept regular, e.g., \AE or AE
5. Unless otherwise specified treat "\AE" as "\E"
6. Allow user to search for partial or incomplete words—user could not use other parts of the word to search on
7. Make search available on all the cards
8. The searching of text should be upper/lower case insensitive
9. Use familiar wording for the user (\E is strange to the user because it looks like AE, but it means A)
10. The search of text should use consistent text format as its display (the display is \E, but the search text is A)
11. Need to have a special table that could be used as a cross-reference of special characters
# UIDR Set Comparison Sheet: Comparison Number 11

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titus Andronicus</td>
<td>Free exploration of Play text</td>
<td>User indicated that the function of the buttons associated with traversing through the play text was not clear</td>
</tr>
</tbody>
</table>

## Set 1

1. Need "Help" button in Scene cards
2. Probably could label the traversing buttons
3. Can traverse within a scene; can traverse among scenes and to the front of the play; but cannot move from act to act or to the beginning of an act.
4. Identify in the help screen that the [ ] button returns to the table of contents (unsupported user knowledge)
5. Communicate the action of the scene navigation buttons to the user (un-documented)
6. Communicate the action of the intra-scene navigation buttons more clearly
7. Explain "Field": distinguish between "Field" arrow buttons and scrollbar, or eliminate arrows if no distinction exists
8. Either: Provide knowledge that left/right buttons change scene, or Provide knowledge that left/right buttons change card and that one scene is on one card
9. Provide more direct knowledge of location within play
10. Make user aware when they have scrolled to some boundary (end of page, etc.)
11. Add description or icons to scene-change buttons
12. Make "go to beginning" function more clear than shown by arrow to line

## Set 2

1. Denote that the rightmost arrows scroll by scenes/acts
2. Include the scene/act scrolling icons in the help
3. Label buttons to indicate function
4. The arrow symbols used for paging through text and for moving through cards gave that symbol two different meanings
5. Group the buttons for paging with the scroll bar
6. Groups of icons should be labeled clearly
7. Different icons that do significantly different things should have distinct labels
8. Clearly label navigational options
### UIDR Set Comparison Sheet: Comparison Number 12

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Titus Andronicus</td>
<td>Using the Find operation (via the eyeball ICON)</td>
<td>User experienced difficulty with functional ambiguity associated with the Find Button's operation</td>
</tr>
</tbody>
</table>

#### Set 1

1. Eliminate glossary feature from Find
2. Add a mouse over pop-up glossary definition in window to words that are underlined
3. Need clear wording in “Help” text to know finding a glossary item. The complex steps of actually going to a glossary and getting the right item is non-trivial.
4. Regular Find (rectangle box surrounding the item)—cannot find glossary while highlighted Find does. Needs a more consistent way.
5. Add glossary icon to screen cards. Add glossary icon to 1st card
6. Add glossary to [scene] list on 1st screen for direct access!
7. Support going to glossary page from last page (Act) of Play
8. Identify the different functions of the eyeball and “Again” buttons more clearly
9. Identify all the possible search text entry methods in the help screen more accurately
10. Provide information for searching occurrences of glossary words
11. Provide more explicit knowledge of 2 separate functions of eyeball (find text/find in glossary)
12. Communicate knowledge of scope of search
13. Better icon for Find button
14. Show what the most recently found item was—so “Again” will have meaning and context

#### Set 2

1. Separate icons, “Find” and “Again” should not be stacked-- inconsistent
2. Label icons and/or use intuitive icons
3. Indicate current function of multi-function buttons/icons
4. Buttons to search and go to the glossary should be distinct
5. Icons for commands should be in a standard location and format (same size, similar shape). Display formats (command icons in this case) should have a consistent structure evident to the user, so that any particular command icon is always presented in same place and same way
6. Stay consistent in feedback of related functions
### UIDR SET COMPARISON SHEET: COMPARISON NUMBER 13

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celtic History Museum</td>
<td>Use of transporter E.g., task: Follow a hyperlink and get back (i.e., return)</td>
<td>User got lost in museum, e.g., within Hall of Religion: Druids card, the user followed the Ogam hyperlink and lost way/context/place within museum</td>
</tr>
</tbody>
</table>

**Set 1**

1. Add a button for returning to the place of initiating the hyperlink
2. Add a truthful map or status field showing exactly where (which artifact, which hall) the user is
3. Provide direct Hyperlink history return facility
4. Identify possible return paths to user after following hyperlinks
5. Provide mechanisms for returning to last exhibit; inform user of name & location of last exhibit
6. Provide knowledge of previous location or method of return
7. Make user aware of where hyperlinks will take them either before or after it is taken
8. Provide a direct, automatic means to get back to original place (e.g., hypertext, bookmark, backward trace)

**Set 2**

1. Maintain a list of all user activities for user review
2. Allow users to reverse control of their actions
3. Record history of user actions/path
4. When using hypertext links, require users to take only one simple key action to return to the last screen
5. Each data display should provide needed context
6. Display some indication of context to the user
7. Display the results of previous entries affecting present actions and context information
8. A back up option needs to be installed to let the user have the ability to retrace their steps
## UIDR Set Comparison Sheet: Comparison Number 14

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celtic History Museum</td>
<td>Trace bibliographic reference and get back (i.e., return)</td>
<td>User could not trace a bibliographic reference and return—e.g., within <em>Marriage in Ancient Ireland</em> card, the user could not trace the reference [56] and reliability get back to the marriage card</td>
</tr>
</tbody>
</table>

### Set 1

1. Provide direct route to Bibliography—e.g., add hyperlink feature to bibliographic reference
2. Add return feature (button) in bibliography: Add a button for returning to the place of initiating the hyperlink
3. Communicate user's position (artifact & Hall location) from the overview map
4. Identify return path or last exhibit from Bibliography & overview map
5. Provide a personal bibliography list for the user
6. Provide information about getting to Bibliography
7. Provide knowledge of original location or method of return
8. Make it possible to go directly from a reference to its bibliographic listing & return directly to where you were (e.g., hypertext, bookmark, backward trace)

### Set 2

1. Maintain a list of all user activities to be browsed by the user
2. Minimize steps for tasks—provide "Bibliography" button on all cards
3. Always provide full guidance information
4. Ensure that moving from a citation & reference can be done in a fewer number of actions
5. Do not expect the user to remember the citation # and the place in the museum where the citation occurred
6. When using hypertext links require users to take only one simple key action to return to the last screen
7. Designing sequence control logic should consider task requirements and associated user characteristics (i.e., the tracing of references is not supported)
8. Provide reversible actions
9. In the bibliographic section some highlighted function should be available to indicate what number reference was selected
10. When returning to the map the area associated with reference should be highlighted for guidance purposes
UIDR SET COMPARISON SHEET: COMPARISON NUMBER 15

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celtic History Museum</td>
<td>Window and card scrolling</td>
<td>User confounded text-field scrolling with card scrolling—i.e., cards</td>
</tr>
<tr>
<td></td>
<td>Task: Exploratory learning</td>
<td>with more information on a topic allow/feature scrolling, while</td>
</tr>
<tr>
<td></td>
<td></td>
<td>text-fields within the cards also scrolled—thus confusing users</td>
</tr>
</tbody>
</table>

Set 1

1. Eliminate page up/down metaphor— inconsistent with walking through the museum. Replace:
   - & ->>: Exhibit, Artifact
   - walk-to-next, walk-fast-to-next
   - to go to the next page
2. To go to the next page or next artifact be consistent— e.g., walk from a1-a2-a3-a4-b1-c1-d1-d2-e1
3. Provide understanding of text field identification & manipulation— i.e., text fields of different cards should be easily distinguishable
4. Identify card scrolling to user at appropriate time
5. Provide clear and unambiguous method for viewing the various pages of an exhibit
6. Make function of Down-arrow explicit
7. Make user aware of difference between:
   - changing topics, and
   - getting more information on the same topic
8. Don’t say “next page of this artifact” in Introductory to Museum, referring to the arrow that changes the whole topic (rather than scrolling for more information on this one)

Set 2

1. Do not allow scrolling within the card, or
2. Scroll all cards that have additional information
3. Label icons/buttons with their function
4. Group similar menu options
5. Label grouped options
6. Have consistent control within windows
7. Redesign the down arrow icon to look like moving to another picture/depth (i.e., process it represents), test the resulting symbol set with a representative group of users to see if the meaning is understood
8. Design interface so the different controls are distinct in position or format and are labeled clearly (e.g., the text scrolling button is next to the card scrolling and looks the same)
9. There should be a distinguishing icon difference between navigating between going left-right & up-down
10. When there is only a single path to take this should be indicated to the user
11. Some suggested help should be installed to alleviate taking ambiguous directions
### UIDR SET COMPARISON SHEET: COMPARISON NUMBER 16

<table>
<thead>
<tr>
<th>Interface</th>
<th>Task</th>
<th>Usability Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Celtic History Museum</td>
<td>Entering and exiting the hallways via the Hallway view</td>
<td>User confused with arrows in Hallway view</td>
</tr>
</tbody>
</table>

#### Set 1

1. Introductory information (as contained on the next pages after hallway) is not readily accessible. Icons (graphic) are more powerful than ⇒ right arrow—hence user may click on 1st picture icon and miss the introduction
2. Make “Introductory Arrow” on hallway represent more obviously that it’s a clickable button (i.e., [ INTRO ⇒ ]) or make an introductory icon that can be used in every hallway to represent the introductory information
3. Identify and explain that the “Enter Arrow” on Hall map enters the hall
4. Identify and explain that the “Exit Arrow” on Hall map exits the hall
5. Explain black arrows in hallway views. Perhaps eliminate.
6. Provide knowledge that lobby/introductory cards are at beginning of hall
7. Provide knowledge that lobby introductions can be quite large
8. Show an Introduction icon right after right arrow
9. Show a museum overview map icon right after left arrow
10. Since you can’t have a hallway map ‘till you’ve seen the introduction, either provide a hallway map immediately or skip the introductory after right arrow

#### Set 2

1. Label the two arrow keys consistently and informatively to designate the function it performs; make labels sufficiently different form one another to prevent user confusion
2. Buttons (arrows) should be labeled clearly and correctly (text mentions right arrow when it seems left should be used)
3. The arrow key indicated move to the next frame, but in fact moves to a completely different category than the one being displayed
4. Indicate w/display or words where the arrow keys will send you
APPENDIX H:  PHASE THREE DATA

This appendix outlines the survey results from phase three of the TMM evaluation experiment.
TABLE A.H.1. RAW DATA FROM EVALUATOR 1

<table>
<thead>
<tr>
<th></th>
<th>UP</th>
<th>COH</th>
<th>SPEC</th>
<th>UC</th>
<th>UN</th>
<th>UT</th>
<th>AUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTER1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UP 01</td>
<td>+1</td>
<td>0</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
</tr>
<tr>
<td>UP 02</td>
<td>-2</td>
<td>-2</td>
<td>-1</td>
<td>+2</td>
<td>+1</td>
<td>0</td>
<td>-2</td>
</tr>
<tr>
<td>UP 03</td>
<td>+1</td>
<td>-2</td>
<td>+1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>+1</td>
</tr>
<tr>
<td>UP 04</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UP 05</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>UP 06</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>UP 07</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>-2</td>
<td>+1</td>
</tr>
<tr>
<td>UP 08</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+3</td>
<td>+1</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td>UP 09</td>
<td>+3</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
</tr>
<tr>
<td>UP 10</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td>UP 11</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
<td>0</td>
</tr>
<tr>
<td>UP 12</td>
<td>+3</td>
<td>+2</td>
<td>+3</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td></td>
</tr>
<tr>
<td>UP 13</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td>UP 14</td>
<td>-2</td>
<td>+2</td>
<td>+1</td>
<td>-2</td>
<td>-2</td>
<td>-3</td>
<td>-1</td>
</tr>
<tr>
<td>UP 15</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td>UP 16</td>
<td>+3</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
</tr>
</tbody>
</table>

TABLE A.H.2. RAW DATA FROM EVALUATOR 2

<table>
<thead>
<tr>
<th></th>
<th>UP</th>
<th>COH</th>
<th>SPEC</th>
<th>UC</th>
<th>UN</th>
<th>UT</th>
<th>AUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INTER1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UP 01</td>
<td>0</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>-1</td>
</tr>
<tr>
<td>UP 02</td>
<td>-2</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 03</td>
<td>0</td>
<td>-2</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>UP 04</td>
<td>+2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>UP 05</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 06</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 07</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 08</td>
<td>0</td>
<td>+1</td>
<td>+3</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
</tr>
<tr>
<td>UP 09</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>+1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UP 10</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UP 11</td>
<td>+3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>+3</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td>UP 12</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>UP 13</td>
<td>0</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>UP 15</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+3</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td>UP 16</td>
<td>+3</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+3</td>
<td>+3</td>
<td>0</td>
</tr>
</tbody>
</table>
### Table A.H.3. Raw Data from Evaluator 3

<table>
<thead>
<tr>
<th></th>
<th>INTER1</th>
<th>UP</th>
<th>COH</th>
<th>SPEC</th>
<th>UC</th>
<th>UN</th>
<th>UT</th>
<th>AUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL3</td>
<td>INTER1</td>
<td>UP_01</td>
<td>+2</td>
<td>-2</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER1</td>
<td>UP_02</td>
<td>-3</td>
<td>-1</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER1</td>
<td>UP_03</td>
<td>+1</td>
<td>-2</td>
<td>+1</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER1</td>
<td>UP_04</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>0</td>
<td>+2</td>
<td>+3</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER2</td>
<td>UP_05</td>
<td>-3</td>
<td>-2</td>
<td>+3</td>
<td>-2</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER2</td>
<td>UP_06</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER2</td>
<td>UP_07</td>
<td>-1</td>
<td>+1</td>
<td>+2</td>
<td>+2</td>
<td>-1</td>
<td>-3</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER2</td>
<td>UP_08</td>
<td>+2</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
<td>+2</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER3</td>
<td>UP_09</td>
<td>+2</td>
<td>+2</td>
<td>-1</td>
<td>+3</td>
<td>0</td>
<td>+3</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER3</td>
<td>UP_10</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER3</td>
<td>UP_11</td>
<td>-1</td>
<td>-2</td>
<td>0</td>
<td>+2</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER3</td>
<td>UP_12</td>
<td>+1</td>
<td>-2</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
<td>+3</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER4</td>
<td>UP_13</td>
<td>0</td>
<td>0</td>
<td>+3</td>
<td>0</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER4</td>
<td>UP_14</td>
<td>-1</td>
<td>-2</td>
<td>+3</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER4</td>
<td>UP_15</td>
<td>0</td>
<td>-2</td>
<td>+2</td>
<td>+3</td>
<td>+1</td>
<td>0</td>
</tr>
<tr>
<td>EVAL3</td>
<td>INTER4</td>
<td>UP_16</td>
<td>-2</td>
<td>-3</td>
<td>+1</td>
<td>+1</td>
<td>+2</td>
<td>-1</td>
</tr>
</tbody>
</table>

### Table A.H.4. Raw Data from Evaluator 4

<table>
<thead>
<tr>
<th></th>
<th>INTER1</th>
<th>UP</th>
<th>COH</th>
<th>SPEC</th>
<th>UC</th>
<th>UN</th>
<th>UT</th>
<th>AUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL4</td>
<td>INTER1</td>
<td>UP_01</td>
<td>-2</td>
<td>+2</td>
<td>+3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER1</td>
<td>UP_02</td>
<td>+2</td>
<td>0</td>
<td>-2</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER1</td>
<td>UP_03</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>+2</td>
<td>-1</td>
<td>+2</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER1</td>
<td>UP_04</td>
<td>+3</td>
<td>+3</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER2</td>
<td>UP_05</td>
<td>-3</td>
<td>-3</td>
<td>0</td>
<td>-2</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER2</td>
<td>UP_06</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>+2</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER2</td>
<td>UP_07</td>
<td>+2</td>
<td>0</td>
<td>+1</td>
<td>0</td>
<td>+2</td>
<td>-1</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER2</td>
<td>UP_08</td>
<td>-1</td>
<td>+2</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER3</td>
<td>UP_09</td>
<td>-1</td>
<td>+1</td>
<td>+2</td>
<td>+3</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER3</td>
<td>UP_10</td>
<td>-2</td>
<td>0</td>
<td>0</td>
<td>-3</td>
<td>-2</td>
<td>-2</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER3</td>
<td>UP_11</td>
<td>+2</td>
<td>0</td>
<td>+3</td>
<td>+2</td>
<td>+2</td>
<td>+1</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER3</td>
<td>UP_12</td>
<td>-2</td>
<td>-3</td>
<td>+2</td>
<td>+1</td>
<td>-2</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER3</td>
<td>UP_13</td>
<td>+2</td>
<td>+3</td>
<td>0</td>
<td>+2</td>
<td>0</td>
<td>+3</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER4</td>
<td>UP_14</td>
<td>+2</td>
<td>+3</td>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>+1</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER4</td>
<td>UP_15</td>
<td>+1</td>
<td>+3</td>
<td>+2</td>
<td>0</td>
<td>+3</td>
<td>0</td>
</tr>
<tr>
<td>EVAL4</td>
<td>INTER4</td>
<td>UP_16</td>
<td>+3</td>
<td>-2</td>
<td>-1</td>
<td>+2</td>
<td>+2</td>
<td>-3</td>
</tr>
</tbody>
</table>
Reproduced statistical analysis for evaluator responses.

General Linear Models Procedure
Class Level Information

<table>
<thead>
<tr>
<th>Class</th>
<th>Levels</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVAL</td>
<td>4</td>
<td>EVAL1 EVAL2 EVAL3 EVAL4</td>
</tr>
<tr>
<td>INTER</td>
<td>4</td>
<td>INTER1 INTER2 INTER3 INTER4</td>
</tr>
<tr>
<td>UPROB</td>
<td>16</td>
<td>UP_01 UP_02 UP_03 UP_04 UP_05 UP_06 UP_07 UP_08 UP_09 UP_10 UP_11 UP_12 UP_13 UP_14 UP_15 UP_16</td>
</tr>
</tbody>
</table>

Number of observations in data set = 64

General Linear Models Procedure
Dependent Variable: S1_UP

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>105.92187500</td>
<td>2.22</td>
<td>0.0145</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>113.07812500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>219.00000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square C.V. S1_UP Mean
0.427769 -274.1961 -0.57812500

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>21.19062500</td>
<td>8.51</td>
<td>0.0055</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>4.04687500</td>
<td>0.54</td>
<td>0.6595</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>71.31250000</td>
<td>2.36</td>
<td>0.0186</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>9.17187500</td>
<td>1.22</td>
<td>0.3146</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>21.39062500</td>
<td>8.51</td>
<td>0.0055</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>4.04687500</td>
<td>0.54</td>
<td>0.6595</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>71.31250000</td>
<td>2.36</td>
<td>0.0186</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>9.17187500</td>
<td>1.22</td>
<td>0.3146</td>
</tr>
</tbody>
</table>

FIGURE A.H.1. STATISTICAL ANALYSIS FOR EVALUATOR RESPONSES FOR VARIABLE R_UP
### Figure A.H.2: Statistical Analysis for Evaluator Responses for Variable S2_COH

```
<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>131.37500000</td>
<td>3.68</td>
<td>0.0002</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>84.62500000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>216.00000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square: 0.600825, C.V.: -548.5334, S2_COH Mean: -0.25000000
```

### Figure A.H.3: Statistical Analysis for Evaluator Responses for Variable S3_SPEC

```
<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>4.00000000</td>
<td>2.13</td>
<td>0.1517</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>6.12500000</td>
<td>1.09</td>
<td>0.3648</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>108.3750000</td>
<td>4.80</td>
<td>0.0001</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>12.8750000</td>
<td>2.28</td>
<td>0.0920</td>
</tr>
</tbody>
</table>

Source         | DF | Type III SS | F Value | Pr > F |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>4.00000000</td>
<td>2.13</td>
<td>0.1517</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>6.12500000</td>
<td>1.09</td>
<td>0.3648</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>108.3750000</td>
<td>4.80</td>
<td>0.0001</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>12.8750000</td>
<td>2.28</td>
<td>0.0920</td>
</tr>
</tbody>
</table>

R-Square: 0.604938, C.V.: -104.9926, S3_SPEC Mean: -1.06250000
```
### General Linear Models Procedure

**Dependent Variable: S4_UC**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>82.18750000</td>
<td>2.79</td>
<td>0.0024</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>69.81250000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>152.00000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$R^2 = 0.460647 \quad \text{C.V.} = -209.7764 \quad \text{S4_UC Mean} = -0.59375000$$

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>22.56250000</td>
<td>14.54</td>
<td>0.0004</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>3.31250000</td>
<td>0.71</td>
<td>0.5501</td>
</tr>
<tr>
<td>UPORB(INTER)</td>
<td>12</td>
<td>44.12500000</td>
<td>2.37</td>
<td>0.0183</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>12.18750000</td>
<td>2.62</td>
<td>0.0624</td>
</tr>
</tbody>
</table>

**FIGURE A.H.4. STATISTICAL ANALYSIS FOR EVALUATOR RESPONSES FOR VARIABLE S4_UC**

### General Linear Models Procedure

**Dependent Variable: S5_UN**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>104.92187500</td>
<td>3.65</td>
<td>0.0002</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>68.07812500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>173.00000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$$R^2 = 0.536045 \quad \text{C.V.} = -191.9967 \quad \text{S5_UN Mean} = -0.64062500$$

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>26.26562500</td>
<td>17.36</td>
<td>0.0001</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>8.17187500</td>
<td>1.80</td>
<td>0.1607</td>
</tr>
<tr>
<td>UPORB(INTER)</td>
<td>12</td>
<td>65.81250000</td>
<td>3.63</td>
<td>0.0008</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>4.67187500</td>
<td>1.03</td>
<td>0.3886</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>26.26562500</td>
<td>17.36</td>
<td>0.0001</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>8.17187500</td>
<td>1.80</td>
<td>0.1607</td>
</tr>
<tr>
<td>UPORB(INTER)</td>
<td>12</td>
<td>65.81250000</td>
<td>3.63</td>
<td>0.0008</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>4.67187500</td>
<td>1.03</td>
<td>0.3886</td>
</tr>
</tbody>
</table>

**FIGURE A.H.5. STATISTICAL ANALYSIS FOR EVALUATOR RESPONSES FOR VARIABLE S5_UN**
### General Linear Models Procedure

**Dependent Variable: S6_UT**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>93.687500000</td>
<td>3.35</td>
<td>0.0004</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>66.312500000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>160.000000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square C.V. S6_UT Mean
0.532805 -228.5033 -0.53125000

**Source**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>.1</td>
<td>18.06250000</td>
<td>12.26</td>
<td>0.0011</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>11.56250000</td>
<td>2.62</td>
<td>0.0626</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>57.87500000</td>
<td>3.27</td>
<td>0.0019</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>6.18750000</td>
<td>1.40</td>
<td>0.2552</td>
</tr>
</tbody>
</table>

**Source**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>18.06250000</td>
<td>12.26</td>
<td>0.0011</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>11.56250000</td>
<td>2.62</td>
<td>0.0626</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>57.87500000</td>
<td>3.27</td>
<td>0.0019</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>6.18750000</td>
<td>1.40</td>
<td>0.2552</td>
</tr>
</tbody>
</table>

**FIGURE A.H.6. STATISTICAL ANALYSIS FOR EVALUATOR RESPONSES FOR VARIABLE S6_UT**

### General Linear Models Procedure

**Dependent Variable: S7_AUP**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>49.54687500</td>
<td>1.34</td>
<td>0.2060</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>87.45312500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>137.000000000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square C.V. S7_AUP Mean
0.358069 -1274.569 -0.10937500

**Source**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type I SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>0.76562500</td>
<td>0.39</td>
<td>0.5334</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>13.54687500</td>
<td>2.32</td>
<td>0.0877</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>28.93750000</td>
<td>1.24</td>
<td>0.2866</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>6.29687500</td>
<td>1.08</td>
<td>0.3672</td>
</tr>
</tbody>
</table>

**Source**

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Type III SS</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERCEPT</td>
<td>1</td>
<td>0.76562500</td>
<td>0.39</td>
<td>0.5334</td>
</tr>
<tr>
<td>INTER</td>
<td>3</td>
<td>13.54687500</td>
<td>2.32</td>
<td>0.0877</td>
</tr>
<tr>
<td>UPROB(INTER)</td>
<td>12</td>
<td>28.93750000</td>
<td>1.24</td>
<td>0.2866</td>
</tr>
<tr>
<td>EVAL</td>
<td>3</td>
<td>6.29687500</td>
<td>1.08</td>
<td>0.3672</td>
</tr>
</tbody>
</table>

**FIGURE A.H.7. STATISTICAL ANALYSIS FOR EVALUATOR RESPONSES FOR VARIABLE S7_AUP**
### Figure A.H.8. Statistical Analysis for Evaluator Responses for Variable R_UP

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>95.75520833</td>
<td>5.47</td>
<td>0.0001</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>41.46701389</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>137.22222222</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square: 0.629109  
C.V.: -152.3214  
R_UP Mean: -0.63020833

### Figure A.H.9. Statistical Analysis for Evaluator Responses for Variable R_USER

<table>
<thead>
<tr>
<th>Source</th>
<th>DF</th>
<th>Sum of Squares</th>
<th>F Value</th>
<th>Pr &gt; F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>19</td>
<td>72.71354167</td>
<td>4.45</td>
<td>0.0001</td>
</tr>
<tr>
<td>Error</td>
<td>45</td>
<td>38.73090278</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncorrected Total</td>
<td>64</td>
<td>111.44444444</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R-Square: 0.566167  
C.V.: -157.6323  
R_USER Mean: -0.58854167

---

Appendix H: Phase Three Data — 326 —
APPENDIX I: TMM & DGL TRAINING MATERIALS

This appendix contains reproductions of the overhead transparencies used in the TMM and DGL training in phase two.
This package contains a sample set of **TMM Lecture Materials**.

Questions or comments should be addressed to

Kevin A. Mayo  
Department of Computer Science—0106  
Virginia Tech (VPI&SU)  
Blacksburg, Virginia 24061

or  
mayo-chi@xerox.com  
mayo@cs.vt.edu

**TMM Introduction**

- **TMM** = Task Mapping Model
- **User-centered** task analysis technique
- Uses **situational evaluation** to synthesize new user interface design requirements

**Process:**
- Tasks are described in TMM framework
- Descriptions are analyzed for user knowledge needs in task performance
- Knowledge needs are analyzed to identify unsupported needs
- New user interface design requirements are synthesized from unsupported needs

- Used by user interface developers (in interface design and evaluation)

---

Appendix I: TMM & DGL Training Materials
TMM Functionality

- **TMM does/is:**
  - Support the user interface development process by synthesizing new user interface design requirements
  - A task analysis technique that provides:
    - a task description framework
    - methods for analyzing task descriptions
    - methods for synthesizing new user interface design requirements

- **TMM does not/is not:**
  - Restrict design solution space (i.e., force specific design solutions on designers)
  - A model of the cognitive processes of users, only the needs associated with performing tasks

Outline

*Brief Overview of TMM*

Task Description Framework

User Mappings & Knowledge Requirements

Task Ordering Issues

TMM Life Cycle

Example
Theory of Action

Describing Tasks in TMM

- Observation: users perceive /work tasks at various levels of abstraction—domains
  I.e., users maintain various languages by which a task is conceptualized

- TMM task description domains: Problem, Computer Semantic, Computer Syntactic, and Articulation

- Similar domains are found in other models
Domain Framework

- Task Domains (Items are domain specific objects, operations, and sub-tasks.)
  - Problem Domain
    - Items within scope of problem, e.g., quarterly totals, documents
  - Computer Semantic Domain
    - Abstract computer items representative of problem domain items, e.g., file, disk
  - Computer Syntactic Domain
    - Grammatical components for articulation, e.g., Select <Disk-Drive-ID>
  - Articulation Domain
    - Specification of user action (in UAN). Also, specification of feedback to drive evaluation paths.

UAN

- User Action Notation
- Used in articulation domain of TMM framework
- Behavioral modeling of user actions, interface feedback, and interface state
- Model structure
  - User Action
  - Interface Feedback
  - Interface State
- Notation visually representative of actions

References


Domain Item Terminology

- Based on taxonomy of user actions
- Used in all but Articulation Domain

User Task Mappings

- Domains represent levels of task abstraction, but what connects the items across the levels?
- Mappings ...
  - relate domain items across domain boundaries
  - represent the necessary user reconceptualization
    - I.e., the user must perform the mapping during task performance
  - identify necessary user knowledge
  - are directional ...
    - Execution paths
    - Evaluation paths

---

Knowledge Requirements

- Mappings identify relationships among the domain items, but how does the user perform the mapping? How does the user know the relationship?
- User knowledge requirements ...
  - mappings have knowledge requirements
    i.e., knowledge required to perform the mapping
e.g., knowing what to do next, knowing particular system facts, understanding system concepts, etc.
  - types of knowledge in task performance
    - factual knowledge
    - conceptual knowledge
    - procedural knowledge
  - e.g., Letters and memos are stored in system files

Factual Knowledge

Definition: **Factual knowledge** is comprised of single or collections of declarative facts users need for mapping items from one domain to another.

*E.g.*

- problem-computer semantic mapping that requires the factual knowledge: family tree information is stored in a database.
- computer semantic-computer syntactic mapping that requires the factual knowledge: column deletion command is 'col-del'.
- computer syntactic-articulation mapping that requires the factual knowledge: keyboard is QWERTY style.

```
Problem Domain                          Computer Semantic Domain
  Send EMail to John Doe               Edit <Messages>
  Composition Message                  with <Editor>
  Mail Message                         Message has been created
  Mail <Message> to <J. Doe Mail Address>

Computer Semantic Domain              Computer Domain
  <Messages>                           Use the proper editor, <Editor>, to create the file <Message>

Key:
1: FK: Messages are contained within files
   FK: Files are changed through editing
2: FK: Message is prepared
3: FK: All EMail must have an address
   FK: John Doe has an EMail address
   FK: EMail is sent through mail system
```
**Conceptual Knowledge**

Definition: **Conceptual knowledge** is the understanding of relationships between collections of factual knowledge, ideas, and other conceptual knowledge.

*E.g.*

- problem-computer semantic mapping that requires conceptual knowledge about *birds* (including: general characteristics, color, shape, wings, etc.)
- computer semantic-computer syntactic mapping that requires conceptual knowledge about *files* (including: general characteristics, commands, contents, etc.)
- computer syntactic-articulation mapping that requires conceptual knowledge about *voice activated systems* (including: general characteristics, commands, initialization, etc.)

![Conceptual Knowledge Diagram](image)

**Procedural Knowledge**

Definition: **Procedural knowledge** represents possible courses of action, user goals, task ordering and structure, needed during task performance.

*E.g.*

- problem-computer semantic mapping that requires procedural knowledge: Task: Locate database associated with quarterly report.
- computer semantic-computer syntactic mapping that requires procedural knowledge: Task: Perform visual search for print command.
- computer syntactic-articulation mapping that requires procedural knowledge: Task: Move pointer/cursor to right-hand side of box outline (screen position: 655, 231).

![Procedural Knowledge Diagram](image)
Outline

Brief Overview of TMM
Task Description Framework
User Mappings & Knowledge Requirements

Task Ordering Issues

TMM Life Cycle
Example

Task Ordering Issues

- Users can (and do!) perform tasks in a number of different ways by time, order, and alternative method selection
- For completeness and accuracy, all these differences should be captured in task descriptions

- Tasks relationships are characterized as:
  - mandatory or conditional task performance
  - repeated task performance
  - task performance related by order (i.e., order independent, sequential, concurrent, and interleaved)

E.g., consider the overall task: Check Email
- Get Email must precede Read Email
  (how can a message be read before its procured?)
- Read Email must precede Answer Email, but Answer Email is conditionally performed
- ... etc ...

TMM Training Materials—17

TMM Training Materials—18
Task Ordering Notation

- Tasks grouped are delineated with left bracket ( [ )
- Each group has a timing relationship
- Each group has a repeating relationship

Sequential Tasks

- Often, the most common to user tasks
- User performs task A, B, and C in order

E.g.,

```
[ Task: Get Car to 55 mph.
  Task: Set Cruise Control Speed
```

In this example, the user must first get the vehicle to a specified speed and then set the cruise control.
Conditional Tasks

- Sometimes a task is only performed based on a *user-evaluatable* condition

E.g.,

- **Task:** Examine dataset for errors
  - **Task:** Evaluate Dataset
    - IF an error exists in the dataset
    - **Task:** Correct Dataset

In this example the **Correct Dataset** task is performed *only if an error exists*

Revised Tasks

- Task groupings (or single tasks) may be repeated
- "*" = 0 or more task group performances
- "+" = 1 or more task group performance

E.g.,

- **DO+**
  - **Task:** Locate City Name
  - **Task:** Change Font Type

- **DO+**
  - **Task:** Locate City Name
  - **Task:** Change Font Type

These are equivalent
Repeated Tasks (cont)

- The number of repetitions can be specified

E.g.,

```
DO 3
  DO 10
  Task: Insert Diskette
  Task: Format Diskette
  Task: Name Diskette
  Task: Name Box of Diskettes
```

This example shows task repetition, as well as, task nesting

Conditional Repeated Tasks

- Task repetition can also be based on a user-evaluable condition
- Two types of task repetition (looping)
  - test-before looping
  - test-after looping
    *Based on when the user evaluates the condition*

E.g., test-before looping

```
DO WHILE more <File_Icon> to Update
  Task: Locate next file, <File_Icon>
  Task: Update <File_Icon>
```

E.g., test-after looping

```
DO
  Task: Locate next file, <File_Icon>
  Task: Update <File_Icon>
  UNTIL no more <File_Icon> to Update
```
Order Independent Tasks

- User tasks are not always sequential, in fact, there are cases where the user must perform several tasks, but the order of performance is unimportant—order independent tasks.
- Notation is ‘&’

E.g.,

\[
\begin{align*}
\text{Task:} & \quad \text{Set VCR Weekday to Wednesday} \\
\& & \quad \text{Set VCR Hour to 7 pm} \\
\text{Task:} & \quad \text{Set VCR Minute to 21}
\end{align*}
\]

Alternative Methods

- Some user tasks can be performed in several different ways—alternative methods.
- Notation is either ‘|’ or ‘OR’

E.g.,

Notation Alternative 1: 

\[
\begin{align*}
\text{Task: Select File} \\
\text{Task: Key A D} \\
\text{Task: Select File} \\
\text{Task: Select 'Duplicate' item from 'File' menu}
\end{align*}
\]

Notation Alternative 2: 

\[
\begin{align*}
\text{Task: Select File} \\
\text{Task: Key A D} \\
\text{Task: Select 'Duplicate' item from 'File' menu}
\end{align*}
\]

These examples, both equivalent, outline the duplicate file task in Macintosh™ OS.
Concurrent Tasks

- Tasks that are performed simultaneously are concurrent
- Notation is ‘||’

E.g.,

\[
\begin{align*}
|| & \text{ Task: Depress the Clutch} \\
& \text{ Task: Release the Accelerator}
\end{align*}
\]

This is a common example to those of us with standard transmissions. These two tasks are performed at the same time. Another common example is:

\[
\begin{align*}
|| & \text{ Task: Move mouse to Position X, Y} \\
& \text{ Task: Visually track mouse on screen}
\end{align*}
\]

Interleaved Tasks

- Tasks that are suspended and resumed during task performance are interleaved
- Notation is ‘↔’

E.g.,

\[
\begin{align*}
↔ & \text{ Task: Write} \\
& \text{ Task: Look up a word in Thesaurus}
\end{align*}
\]
Task Nesting

- All of the task repetition and timing controls can be nested.

E.g.,

```
Task: Turn Cruise Control on
  Task: Scan Dash for Speed
  ┌ Task: Release Accelerator Pedal
  │ Task: Press Accelerator Pedal Down
  │ Task: Set Cruise Control Speed
```

Outline

- Brief Overview of TMM
- Task Description Framework
- User Mappings & Knowledge Requirements
- Task Ordering Issues
- TMM Life Cycle
- Example
**User Classes in TMM**

- Users are individuals—not homogenous groups
- Users can, however, possess common characteristics
  - User classes
  - User class profiles
- Characteristics can be based on:
  - Users' background and experience
  - Users' knowledge about system and task
  - Users' physical capabilities

- User class profiles are used by TMM for knowledge analysis
- User class profiles add a dimension to analysis
  - i.e., based on user class profiles, should there be:
    - single user interface design
    - multiple user interface design
    - composite user interface design

---

**Deriving User Class Profiles**

- TMM does not directly support the derivation of user class profiles
- TMM modeling does, however, require certain information within user class profiles
  - User knowledge of task structure
  - User knowledge of system and interface
  - User physical capabilities

- How is this information gathered?
  - Experimental manipulations
  - Interviewing
  - Introspective report
  - Repertory grids

---

TMM Life Cycle

- TMM life cycle
  - Select and describe a task
  - Analyze description for knowledge and determine support
  - Recast unsupported knowledge in new user interface design requirements

Task Descriptions

- Tasks are described using the TMM framework

An approach (albeit not the only one):
  - Select a task for analysis
    - Formative evaluation of interface designs can help identify candidates for analysis. I.e., choose tasks with usability problems.
  - Identify and define domain items
    - Problem Domain items
    - Computer Semantic Domain items
    - Computer Syntactic Domain items
    - Articulation sequence, if possible
    - Iterate
  - Identify mappings among domain items
    - Create arcs between mapped items
    - Identify necessary user knowledge for mapping
    - Iterate
  - Don't forget to generate a task description for both the execution and evaluation paths
Knowledge Analysis

- User knowledge is analyzed using TMM task descriptions, user class profiles, and current system interface designs

An approach (albeit not the only one):
- Generate a task description
- Derive the user class profile(s)
- For each of the knowledge requirements identified in the task description (i.e., the factual, conceptual, and procedural knowledge):
  - Answer: Does the user class profile(s) indicate the user's awareness of the knowledge?
    - YES: Supported knowledge requirement
    - Else, answer: Does the user interface design provide the information to the user?
      - YES: Supported knowledge requirement
      - NO: Un-supported knowledge requirement

- This identifies all the un-supported user knowledge requirements

Interface Design Support

- At this point the analysis has identified un-supported user knowledge requirements i.e., a mapping knowledge requirement that the user does not know, and the system does not help the user know
- This knowledge is translated into new user interface design requirements

An approach (albeit not the only one):
- For each un-supported knowledge requirement
  - Identify the mappings and associated domain items
  - Identify the related user interface design characteristics
  - Identify the related user class profile characteristics
  - With regards to the above data, define the new user interface design requirement (see templates)
UIDR Templates

- **UIDR** = User Interface Design Requirement

Possible templates for UIDRs

**Factual Knowledge**
- UIDR: Communicate *<un-supported knowledge>* to user (at the appropriate time in task performance)
- UIDR: Communicate *<Network is Down>* to user (at appropriate time in task performance)

**Conceptual Knowledge**
- UIDR: Provide understanding of *<concept>* to user (at the appropriate time in task performance)
- UIDR: Provide understanding of *<Nuclear Core Reaction>* to user (at appropriate time in task performance)

**Procedural Knowledge**
- UIDR: Identify *<possible paths of executions>* to user (at the appropriate time in task performance)
- UIDR: Identify *<next sub-task is enter PIN>* to user (at the appropriate time in task performance)

Other Uses of TMM

- Metrics over task description
  - *An area of future research*
  - Goodness-of-fit
    - Ratio of the number of supported user knowledge requirements to the total number of user knowledge requirements
  - Task stacking
    - Number of process (tasks and sub-tasks) users must suspend and resume during task performance
  - Both need empirical validation

- Documentation and training
  - *An area of future research*
  - TMM task descriptions can be used in documentation
  - TMM analysis identifies user knowledge requirements—this can focus user training
  - Both need empirical verification
• The following slides contain an short example of using the TMM for situational analysis

Example TMM Analysis

• Example comes from


• Other sources of information include


Interface v.1

- **Software Requirement**
  Design an interactive Dictionary system that will automate finding words and their meanings across multiple, independent dictionaries.

- **Consider ...**

---

Task Description

- **Task:** Use the Dictionary System to find the word 'POPINJAY'
- **Decomposition into domains ...**

---

- **Mappings on Execution & Evaluation Paths ...**
Knowledge Requirements

- Mappings are user reconceptualizations that have knowledge requirements

<table>
<thead>
<tr>
<th>Mapping</th>
<th>Knowledge Type</th>
<th>Knowledge Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Procedural</td>
<td>A dictionary must be chosen before the search</td>
</tr>
<tr>
<td></td>
<td>Procedural</td>
<td>Database is alphabetically indexed by words</td>
</tr>
<tr>
<td></td>
<td>Procedural</td>
<td>Database Functions and Structures</td>
</tr>
<tr>
<td>2</td>
<td>Procedural</td>
<td>Search criteria is specified below the search</td>
</tr>
<tr>
<td></td>
<td>Procedural</td>
<td>A command, &lt;CR&gt; is entered to begin data in database</td>
</tr>
<tr>
<td>3</td>
<td>Procedural</td>
<td>The search criteria must be specified below the search</td>
</tr>
<tr>
<td>4</td>
<td>Procedural</td>
<td>Explicit action is necessary to invoke search command</td>
</tr>
<tr>
<td>5</td>
<td>Procedural</td>
<td>Input of &lt;*&gt; is specified by keyboard entry</td>
</tr>
<tr>
<td>6</td>
<td>Procedural</td>
<td>Input of &lt;*&gt; is specified by keyboard entry</td>
</tr>
<tr>
<td>7</td>
<td>Procedural</td>
<td>Input of search criteria is by keyboard entry</td>
</tr>
<tr>
<td>8</td>
<td>Procedural</td>
<td>Search criteria, &lt;CR&gt; is entered in the command.</td>
</tr>
<tr>
<td>9</td>
<td>Procedural</td>
<td>Search results are presented in a separate window.</td>
</tr>
<tr>
<td></td>
<td>Procedural</td>
<td>Format of search results</td>
</tr>
<tr>
<td>10</td>
<td>Procedural</td>
<td>When desired results are found, the task is repeated</td>
</tr>
<tr>
<td>11</td>
<td>Procedural</td>
<td>When desired results are found, the task is completed</td>
</tr>
</tbody>
</table>

User Class Profile

(UCP) User Class Profile

(UID) User Demographics:
- UD.1: Age: 15-45 yr.
- UD.2: Equal Sex Distribution
- UD.3: Single geographic location (Any Town, USA)

(UPL) User Physical Limitations:
- UPL.1: Standard disability distribution for USA.

(UML) User Mental Limitations:
- UML.1: Below standard disability distribution for USA.

(UKB) User Knowledge Base:
  General Knowledge
    UKB.1: Assume at least a 12th grade education.
  Problem Domain Knowledge
    UBK.2: CK: Dictionary Usage and Terminology
    Computer (Semantic/Syntactic) Domain Knowledge
    - CK.3: Basic Database functions and structures
    - CK.4: DK: Direct Manipulation/WIMPS Interfaces
    - CK.5: PK: Noun-Verb Command Interaction Styles
### Knowledge Analysis

- Each knowledge requirement is compared against: *user class profile & current user interface design*
- Knowledge requirements are classified as either supported or unsupported

<table>
<thead>
<tr>
<th>Knowledge Requirement</th>
<th>Justification</th>
<th>Supp</th>
<th>Supp</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 FK - A dictionary must be chosen before the search</td>
<td>Interface has a &lt;select&gt; element in the form field which allows user to select. Also, if no dictionary is selected, the form field is not active.</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Dictionaries are alphabetically ordered by name</td>
<td>User understands that dictionaries are ordered alphabetically by name</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Database Functions and Searches are consistent</td>
<td>User understands how Database Functions and Searches are consistent</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Search criteria is required in query</td>
<td>Part of the Task</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Databases can be searched</td>
<td>User understands how Database Functions and Searches are consistent</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - A command to execute search is provided</td>
<td>User understands how to execute search</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - The search criteria must be specified for search</td>
<td>User understands how to specify search criteria</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Options on the database to invoke search command</td>
<td>User understands how to invoke search command</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Input of search command</td>
<td>The command has a prompt to specify search command</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Input of the command</td>
<td>User does not know the correct command</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Input of search criteria is by keyboard entry</td>
<td>The interface has a prompt to specify search criteria</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Input of search criteria is by keyboard entry</td>
<td>Command Line interface style that has a WYSIWYG interface</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Search results are presented after processing</td>
<td>System feedback with task feedback</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - Format of Search Results</td>
<td>Interface well suited to the intended application and task</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - When search criteria is not met, the task is not performed</td>
<td>User knows when task fails or is accomplished based on system feedback</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>11 FK - When search criteria is met, the task is completed</td>
<td>User knows when task fails or is accomplished based on system feedback</td>
<td>✔</td>
<td>✔</td>
</tr>
</tbody>
</table>

---

### UIDRs

**UIDR1** Communicate "find" is the <search> command> to User (at the appropriate time in task performance).

**UIDR2** Communicate "<CR> Terminates input and executes command> to User (at the appropriate time in task performance).

**UIDR3** Provide understanding of <dictionary may not be appropriate for Search criteria selection> to User (at the appropriate time in task performance).

**UIDR4** Provide understanding of <Membership relationship of search criteria to dictionaries in Dictionary-List> to User (at the appropriate time in task performance).

*Keep in mind that the templates are only examples of how to format UIDRs—they do not necessarily need to be followed.*
Interface v.2

- Problems still exist!
  - Not all UIR s can/will be included within a new user interface design
  - Solutions are often the sources for new problems!

TMM Training Materials—47
This package contains a sample set of

**Design Guideline**

**Lecture Materials.**

Questions or comments should be addressed to

Kevin A. Mayo  
Department of Computer Science—0106  
Virginia Tech (VPI&SU)  
Blacksburg, Virginia 24061

or

mayo.chi@xerox.com  
mayo@cs.vt.edu

---

### Design Guidelines (DG)

- Design guidelines are statements based on observations and empirical evidence that direct designers towards more usable user interface designs.

- E.g., Reduce user memory load  
  Directs designers towards designing interfaces that do not rely on users' memory—making the interface (design) easier for users to use.

- Not all design guidelines are applicable at any one time

- These are guidelines, not rules  
  *(With the exception of compliance.)*
DG's Place

Stage 1: Initial Design
- Design Objectives
- Task/Function Analysis
- Focus on Users
- Design Guidelines
- Structured Walk-through

Stage 2: Formative Evaluation
- Rapid Prototyping
- User Derived Interfaces
- User Acceptance Testing

Stage 3: Summative Evaluation
- Operational Software Interface
- Benchmarking
- Formal Experimentation

Initial Design
- Specify initial design constraints
- Focus designers toward usable designs

Formative Evaluation
- Verify usability problems

Summative Evaluation
- Compliancy

DG Research

- Types of DG
  - Generic DG independent of platform, interaction style, interaction devices
    - Smith and Mosier
    - Various Sources

- Specific DG
  - X Based DG
    - OSF/Motif™
    - OpenLook®
    - Apple®
    - Microsoft® Windows™

- This presentation includes the Smith and Mosier generic design guidelines, the Apple® guidelines, and the OSF/Motif™ guidelines.
DG Research References

Sample List:


Smith and Mosier (S&M)

- 944 general guidelines used for tailoring
- Six functional areas of interaction:

<table>
<thead>
<tr>
<th>Functional Area</th>
<th>#</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Data Entry</td>
<td>199</td>
</tr>
<tr>
<td>2 Data Display</td>
<td>298</td>
</tr>
<tr>
<td>3 Sequence Control</td>
<td>184</td>
</tr>
<tr>
<td>4 User Guidance</td>
<td>110</td>
</tr>
<tr>
<td>5 Data Transmission</td>
<td>83</td>
</tr>
<tr>
<td>6 Data Protection</td>
<td>70</td>
</tr>
</tbody>
</table>

- Each guideline
  - numbered sequentially w/i section
  - single sentence
  - examples are provided
  - cross references are provided

The following discussion contains information and quotes from:

### S&M Data Entry 1.

<table>
<thead>
<tr>
<th>Designation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>selection, entry, and display position of an item</td>
</tr>
<tr>
<td>Directional</td>
<td>entry of directional data (e.g., heading, bearing)</td>
</tr>
<tr>
<td>Text</td>
<td>initial entry and editing of text</td>
</tr>
<tr>
<td>Data Forms</td>
<td>entry into defined fields</td>
</tr>
<tr>
<td>Graphics</td>
<td>entry into defined fields</td>
</tr>
<tr>
<td>Plotting</td>
<td>entry of data that shows spatial, temporal, or other relationships among data</td>
</tr>
<tr>
<td>Drawing</td>
<td>checking correctness of entry</td>
</tr>
<tr>
<td>Processing</td>
<td>checking correctness of entry</td>
</tr>
</tbody>
</table>

### S&M Design Objectives

- Consistency of data entry transactions
- Minimal memory load on user
display
- Minimal entry actions by users
display
- Flexibility for user control of data entry
1.0•24 Prompting Data Entry
Provide prompting for the required formats and acceptable values for data entries.

Example

(Good) Vehicle type: __
   c = Car
   t = Truck
   b = Bus

(Bad) Vehicle type: __

Exception: Prompting may not be needed by skilled users and indeed may hinder rather than help their performance in situations where display output is slow; for such users prompting might be provided as an optional aid.

Comment: Prompting is particularly needed for coded data entries. Menu selection may be appropriate for that purpose, because menu selection does not require the user to remember codes but merely to choose among displayed alternatives. Other methods of prompting include labeling data fields, such as

   Vehicle type (c/t/b): __

and/or providing optional guidance displays.

See also: 1.4•5, 4.4•7, and Section 3.1.3

0 "Data display refers to computer output of data to a user, and assimilation of information from such outputs."

1 Text: output of textual data

2 Data forms: related information displayed in labeled fields

3 Tables: data is displayed in row-column format to show order/relationships

4 Graphics (.1 Scaling; .2 Scatterplots; .3 Curves and line graphs; .4 Bar graphs; .5 Pie charts; .6 Pictures and diagrams; .7 Flowcharts; .8 Maps and situation displays): Show spatial, temporal, or other relations

5 Format: organization of different types of data in a display

6 Coding: distinctive means for highlighting different categories of data

7 Display control (.1 Selection; .2 Framing; .3 Update; .4 Suppression; .5 Window overlays): procedures by which users can specify data

8 Design change
2.6.21 Limited Use of Size Coding
Consider size coding, i.e., varying the size of displayed alphanumerics and other symbols, only for applications where displays are not crowded.

Comment: Perhaps as many as five symbols might be used for data categorization, but two or three will probably prove the practical limit.

2.6.22 Adequate Differences in Size
For size coding, a larger symbol should be at least 1.5 times the height of the next smaller symbol.

Comment: An increase in symbol height must usually be accompanied by a proportional increase in width to preserve a constant aspect ratio and so facilitate symbol recognition.

S&M Sequence Control 3.
.
.0 "Sequence control refers to use actions a computer logic that initiate, interrupt, or terminate transactions."
.
.1 Dialogue types (.1 Question and answer; .2 Form filling; .3 Menu selection; .4 Function keys; .5 Command language; .6 Query language; .7 Natural language; .8 Graphic interaction): sequence control must match the needs of users and tasks.
.
.2 Transaction selection: the control actions
.
.3 Interrupt: allow user to change ongoing transactions
.
.4 Context definition: help users know where they are in the task
.
.5 Error management: help users avoid and correct errors in task performance
.
.6 Alarms: let users control computer generated alarms and signals
.
.7 Design change
### S&M Sequence Control

User should *feel in control* of the computer and its actions. Dialogue type selection is based on user, task, and user-training level.

<table>
<thead>
<tr>
<th>Dialogue Type</th>
<th>Required User Training</th>
<th>Tolerable Speed of Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q&amp;A</td>
<td>Little/None</td>
<td>Moderate</td>
</tr>
<tr>
<td>Form Filling</td>
<td>Moderate/Little</td>
<td>Slow</td>
</tr>
<tr>
<td>Menu Selection</td>
<td>Little/None</td>
<td>Very Fast</td>
</tr>
<tr>
<td>Function Keys</td>
<td>Moderate/Little</td>
<td>Very Fast</td>
</tr>
<tr>
<td>CLI</td>
<td>High</td>
<td>Moderate/Slow</td>
</tr>
<tr>
<td>Query Lang.</td>
<td>High/Moderate</td>
<td>Moderate</td>
</tr>
<tr>
<td>Natural Lang.</td>
<td>Moderate/Little</td>
<td>Fast</td>
</tr>
<tr>
<td>Graphic Interaction</td>
<td>High</td>
<td>Very Fast</td>
</tr>
</tbody>
</table>

Estimates in this table may vary among system designs.

### S&M Sequence Control

**Question and answer**

"Question-and-answer dialogues, where the computer poses questions for a user to answer, are suited to novice users."

**Form filling**

"Form filling permits a user to enter a series of related date items or control options as a single transaction."

**Menu selection**

"Menu selection permits a user to specify control entries by pointing at displayed options or keying associated codes."

**Function keys**

"Function keys permit control entries by direct selection of labeled keys, rather than from displayed menus."

**Command language**

"Command language permits a user to specify desired control actions by composing messages to a computer."

**Query language**

"Query language is a special form of command language that can be used to request information from a computer."

**Natural language**

"Natural language recognition might permit a novice user to compose commands without any special training."

**Graphic interaction**

"Graphic interaction permits a user to select displayed control elements by pointing and other direct manipulation."
S&M Sequence Control e.g.

3.2.13 Stacked Control Entries

Allow users to key a sequence of commands or option codes as a single stacked control entry.

Comment: In particular, allow users to enter stacked entries from any menu so that an experienced user can make any specific control entry without having to view subsequent menus.

Comment: Control entry stacking may be helpful when a user is being prompted to enter a series of parameter values, and knows what several succeeding prompts will request and what values to enter.

Comment: Control entry stacking will permit a transition from simple step-by-step control entry by novice users, as in menu selection and question-and-answer dialogues, to the entry of extended command-language statements by experienced users; entry stacking is especially helpful in time-shared systems where computer response to any user entry may be slow.

See also: 3.1.3\*36, 3.1.5\*13, 3.5\*4, 3.5\*5


.0 "User guidance refers to error messages, alarms, prompts, and labels, as well as to more formal instructional material."

.1 Status information: on current processing should be available—automatically

.2 Routine feedback: should be provided by the computer about transactions

.3 Error feedback: should be provided by the computer for errors or unexpected events

.4 Job aids: should provide uses with guidance on tasks and interface features

.5 User records: assesses the performance and improvement of user interface designs

.6 Design change
4.3.6 Neutral Wording for Error Messages

Adopt neutral wording for error messages; do not imply blame to the user, or personalize the computer, or attempt to make a message humorous.

Example

(Good) Entry must be a number.
(Bad) Illegal entry.
(Bad) I need some digits.
(Bad) Don't be dumber, user a number.

Comment: Error messages should reflect a consistent view that the computer is a tool, with certain limitations that a user must take into account in order to make the tool work properly. If error messages reflect an attitude that the computer (or its programmer) imposes rules, or establishes legality, the user may feel resentful. If error messages reflect personalization of the computer, as if it were a friendly colleague, a naive user may be misled to expect human abilities the machine does not actually possess. If error messages are worded humorously, any joke will surely wear thin with repetition, and come to seem an intrusion on a user's concern with efficient task performance.

Comment: The same considerations apply for the wording of computer-generated prompts and other instructional material.

S&M Data Transmission 5.

1. "Data transmission refers to computer-mediated communication among system users, and also with other systems."

2. Preparing messages: involves specification of contents, format, and header information.

3. Addressing messages: may require user action and computer aids to specify the destinations for data

4. Initiating transmission: usually under user control, with computer aids for that process

5. Controlling transmission: often handled automatically, but users may need information about that process

6. Receiving messages: may require computer aids for queueing, reviewing, filing, or otherwise disposing of data

6. Design change
5.3.7 Assignment of Priority

When messages will have different degrees of urgency, i.e., different implications for action by their recipients, allow the sender of a message to designate its relative priority, or else have the computer assign priority automatically.

Comment: The computer might impose limits on the priority that any particular user can assign to messages. In a military system, for example, only certain users might be authorized to send messages at the highest priority levels.

See also: 5.5.6

5.3.8 Automatic Queuing for Transmission

Provide automatic queuing of outgoing messages, in order to reduce the need for user involvement in the routine processing of data transmission.

Example: The computer might queue outgoing messages when communication channels to some addresses are temporarily unavailable, and then initiate transmission automatically when a link can be established.

Comment: Specific requirements will vary with the application, but some queuing should be provided.

See also: 5.0.4

5.4 Data Protection

0.0 "Data protection concerns security from unauthorized use, and potential loss from equipment failure and user errors."

1. User identification: procedures should be as simple as possible, consistent with adequate protection

2. Data access: constraints established to exclude unauthorized users should not hinder legitimate use of data

3. Data entry/change: constraints may be needed to prevent unauthorized data change as well as data loss from user errors

4. Data transmission: procedures should ensure data protection when sending and receiving messages

5. Design Change
S&M Data Protection e.g.

6.0*5 Protection from Interrupts

When a proposed user action will interrupt a current transaction sequence, provide automatic means to prevent data loss; if potential data loss cannot be prevented, warn the user and do not interrupt without user confirmation.

Example: If a user should interrupt a series of changes to a data file, then the computer might automatically save both the original and the changed versions of that file for subsequent user review and disposition.

Comment: Some interrupt actions such as BACKUP, CANCEL, or REVIEW, will by their definition cause only limited data change, and so need no special protection. However, if an interrupt action may cause extensive data change (e.g., RESTART, LOG-OFF), then require the user to confirm that action before processing.

See also: 3.3*6

DG Tailoring Process

- Locate all relevant design guidelines
  - System/Interface Artifacts
  - User tasks/environment

- Select design guidelines to use
  - Conflicting
  - Budget

- Translate design guideline
  - User Interface Design Requirement (UIDR)

E.g.,

- 2.0*17 Common Abbreviations: When abbreviations are used, choose those abbreviations that are commonly recognized, and do not abbreviate words that produce uncommon or ambiguous abbreviations.
- UIDR: Use (°C) to represent Degrees Celsius
Apple® Design Guidelines

- This section of materials contains information and quotes taken directly from:

  *The reader is directed to this source for further explanation.*

- Discussion and examples are taken from:
  - General design principles
  - Principles of graphic communication
  - Programming strategy
  - Screen elements
  - Color and sound

Apple™ Fundamental Principles

- 10 Fundamental Principles

  *Metaphors from the real world*
  - Use concrete metaphors and make them plain, so that users have a set of expectations to apply to computer environments.
  - Whenever appropriate, use audio and visual effects that support the metaphor.

  *Direct Manipulation*
  - Users want to feel that they are in charge of the computer's activities.
Fundamental Principles

See-and-point (instead of remember-and-type)
- Users select actions from alternatives presented on the screen.
- The general form of user actions is noun-then-verb, or "Hey, you—do this."
- Users rely on recognition, not recall; they shouldn't have to remember anything the computer already knows.
- Most programmers have no trouble working with a command-line interface that requires memorization and Boolean logic. The average user is not a programmer.

Consistency
- Effective applications are both consistent within themselves and consistent with one another.

WYSIWYG (what you see is what you get)
- There should be no secrets from the user, no abstract commands that only promise future results.
- There should be no significant difference between what the user sees on the screen and what eventually gets printed.

User Control
- The user, not the computer, initiates and controls all actions.

Feedback and dialog
- Keep the user informed.
- Provide immediate feedback
- User activities should be simple at any moment, though they may be complex taken together.
Fundamental Principles

Forgiveness
- Users make mistakes; forgive them.
- The user's actions are generally reversible—let users know about any that aren't.

Perceived stability
- Users feel comfortable in a computer environment that remains understandable and familiar rather than changing randomly.

Aesthetic integrity
- Visually confusing or unattractive displays detract from the effectiveness of human-computer interactions.
- Different "things" look different on the screen.
- Users should be able to control the superficial appearance of their computer workplaces—to display their own style and individually.
- Messes are acceptable only if the user makes them—applications aren't allowed this freedom.
Apple® Graphic Communication

Principles of graphic communication

- Good design must communicate, not just dazzle. It must inform, not just impress.
- The services of a skilled graphic designer are worth the expense.

Visual consistency
construct a believable environment for users—i.e., consistency between real world and interaction metaphor

Simplicity
Simple design is good design—i.e., don't clutter screen with too many windows, complex icons, a google-plex buttons, etc.

Interface artifacts comprise the user's interaction language—keep it simple!

Clarity

Apple® Programming Strategies

Modelessness

Long-term modes, e.g., insert within a word processor

Short-term modes, i.e., user must do something to remain within mode, e.g., holding down mouse button to scroll

Alert modes, i.e., user must fix some condition in order to proceed

- With few exceptions, a given action on the user's part should always have the same result, irrespective of past activities.
- Modes are acceptable if:
  - emulate a familiar real-life situation that is itself modal, e.g., choosing different tools in a graphics application
  - they change attributes, not behavior
  - they block other system functions to emphasize the modality
**Programming Strategies**

*Event Loop*
- Applications are prepared for the user to do anything at any time.

*Reversible actions*
- Always provide a way out.
- Users should be able to cancel actions easily.
- Users should have a range of deliberate choices to confirm that they do want to do something particularly drastic, complex, or time-consuming.

*The screen*
- The screen is the stage for human-computer interactions.

*Plain language*
- Communicate with the user in concise and simple terms

**Designs for the Disabled**

*Vision disabilities*
- Problems with output display—design software with a "zoom" feature that increases the size of characters.
- Don't let people's ability to use your software depend on their ability to distinguish one color from another.
- Be sure that all information conveyed by color coding is presented in some other way (e.g., text, positioning)

*Hearing disabilities*
- Aside from the obvious exceptions of music or voice-synthesis applications, software should never rely solely on sound to provide important information.
- Supplement all audible messages with visual cues, or allow the user to choose visible instead of audible messages.
### Screen Elements

- The look of the screen provides a basic visual context for consistent use across applications.
  - Desktop
  - Windows
    - Controls (*buttons, check boxes, etc.*)
    - Dialog Boxes
    - Alerts
  - Menus
    - Hierarchical
    - Pop-up
    - Pallets
    - Tear-off
  - Color
  - Sound

### Interaction

- Direct manipulation of graphical user interface objects
  - Pointing device (*e.g.*, *mouse, trackball*)
    - Pointing action
    - Selecting action
    - Appearance of pointing device can indicate a mode (*e.g.*, *arrow, I-beam, crosshairs, wristwatch*)
  - Keyboard
Color

- Color is hue + saturation + brightness
  - Hue is the color: red, blue, green, etc.
  - Saturation is the purity of a color—rich, intense colors are highly saturated; dull or diluted colors are not very saturated. E.g., pink is a low-saturation red; navy blue is a high-saturation blue.
  - Brightness is how light or dark a color is—how much white is in it.

- Standard uses include:
  - discrimination among different areas
  - identify functionally related objects
  - identify crucial features

- Color coding (meaning of color)
  - Red: Stop, error, failure

Color Design Principles

- Design in black and white
  - Color should be supplementary
  - Color should not be the only characteristics that distinguishes two objects

- Problems stem from
  - Monitors (variability of quality, and some users will not have color)
  - Printing (very costly and not very accurate across media)
  - Colorblindness (8% males, and 0.5% females in U.S. and Europe)
    - Most common: distinguishing red and green from gray
    - Also, in another form: yellow, blue, and gray are indistinguishable
  - Lighting (environment effects, e.g., dim lighting makes colors more difficult to distinguish)
Color Design Principles

- Limit color use
  - In the standard interface part of applications (menus, window frames, etc.) color should be used minimally or not at all
  - User attention should be on content of the application and not the interface features of the application
- Color coding should be allowed or provided to make information clearer. Providing the user with a small initial selection of distinct colors—four to seven at most—with the capability of changing those available, or adding more, is the best solution.

Graphics applications are an exception.

Color Contrast Notes

- Colors look best against a background of neutral grey.
- Reading and legibility studies in the print (paper) world show that colored text is harder to read than black text on a white background.
- The most illegible color is light blue, which should be avoided for text, thin lines, and small shapes.
- Cases where adjacent objects whose colors differ only in the amount of blue should be avoided.
- Small areas of colors are hard to discriminate—to be able to tell what color an object is, that object must be large enough to see without effort.
- Apple guidelines also provide specific recommendations on color use.
\textbf{TM Sound}

- When is sound used?
  - If it can be \textit{integrated throughout the application}, e.g., games
  - If it can be used to \textit{alert} the user

- General guidelines:
  - \textit{Restraint}: If you overuse sound it will probably be annoying.
  - \textit{Redundancy}: Never let sound be the only indication that something has happened.
  - \textit{Unobtrusiveness}: Do not use loud or harsh sounds, and avoid jingles of more than a few notes.
  - \textit{Significant differences}: All sounds should be distinguishable from one another.
  - \textit{User control}: Let the user control volume and on/off.

\textbf{OSF/Motif™}

- \textit{OSF} = Open Software Foundation
- Two primary principles:
  - \textit{Know the user}.
  - \textit{Empower the user}.
- OSF outlines seven other \textit{principles}
- OSF outlines \textit{models} users employ for consistency

- This section of materials contains information and quotes taken directly from:

- The reader is directed to this source for further explanation.
**Motif™ Design Principles**

*Adopt the user's perspective*
- Get user involved with design
- Understand the users' tasks

*Give the user control*
- Keep interfaces flexible
- Use progressive disclosure
  - i.e., present available functions in logical order

*Use real-world metaphors*
- Allow direct manipulation
- Provide rapid response
- Provide output as input (*equal-opportunity interfaces*)

---

**Motif™ Design Principles**

*Keep interfaces natural*
- Make navigation easy
- Provide natural shades and colors

*Keep interfaces consistent*
- *Intra-application* consistency
- *Inter-application* consistency

*Communicate application actions to the user*
- Give the user feedback
- Anticipate errors
- Use explicit destruction
  - i.e., ask the user first!!

*Avoid common design pitfalls*
- Pay attention to details
- Do not finish prematurely
- Design iteratively
- Start with a fresh perspective
- Hide implementation details
Other Motif™ Guidelines

Application Design Principles
- Choosing components
- Layout
- Interaction
- Component design

Window Manager Design Principles
- Configurability
- Window support
- Window decoration
- Window navigation
- Icons

Designing for International Markets
- Country-specific data formats
- Icons, symbols, and pointer shapes
- Translating screen text

Conclusions
- Guidelines are guidelines—not rules.
- Guidelines are used to increase usability
- Always consider compliancy issues
- Common set of guidelines
VITA

Kevin Andrew Mayo was born on March 30, 1964 to George and Rebecca Mayo. He received his Bachelor of Science in Computer Science from Virginia Polytechnic Institute and State University (Virginia Tech) in 1986. Following the B.S., Kevin continued on for a Masters of Science at Virginia Tech. At this point, Kevin had the opportunity to work with Dr. Sallie Henry in the field of Software Engineering and Software Quality Assurance Metrics. When the M.S. was finished in 1989, he switched gears to pursue a Ph.D. studying task analysis in Human-Computer Interaction and had the fortune to work with Dr. H. Rex Hartson.

During his tenure as a graduate student, Kevin was very involved with Graduate Teaching Assistant training at the University Level. He gave several lectures and represented Virginia Tech at a national GTA training conference. Kevin is very devoted to teaching quality.

Kevin enjoys many artistic endeavors, musical theater, reading, aviculture (both as hobbyist and breeder), and antique restoration. His future plans include continuing work in Human-Computer Interaction and getting onto the business of living.