A SEMANTICALLY-ENHANCED OBJECT-ORIENTED CASE TOOL FOR ENTERPRISE MODELING

by

WILLIAM TAN KHOON LEE

B.B.A.(Hon), Acadia University, 1989

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE in Business Administration

in

THE FACULTY OF GRADUATE STUDIES

(Faculty of Commerce and Business Administration)

We accept this thesis as conforming
to the required standard

THE UNIVERSITY OF BRITISH COLUMBIA

May 1997

© William Tan Khoon Lee, 1997
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Management Information Systems

The University of British Columbia
Vancouver, Canada

Date April 29, 1997

DE-6 (2/88)
Abstract

This thesis is motivated by the desire to build a semantically-enhanced Computer-Aided Software Engineering (CASE) tool that systems analysts would find useful in helping them build object-oriented enterprise models.

The tool would incorporate the unique ability to evaluate candidate models against a set of pre-defined semantic rules. The rules are those of the Object-Oriented Enterprise Modeling (OOEM) method proposed by Wand and Woo (1993).

A survey of current CASE Tools, three of which are reviewed in this thesis, consistently reveal an absence of a built-in semantic integrity checking utility built for or around the tools. The tools were indeed very powerful but were only able to detect syntactic violations in a model.

This thesis takes the view that it is possible, and indeed important for a CASE tool to assist users in building and checking for semantically consistent object-oriented enterprise models. The chapters in this thesis describe a range of activity - from theory strengthening to empirical testing - all with the singular objective of discovering if analysts would in fact find such a tool useful.

The empirical study revealed that analysts do welcome a CASE tool with semantic checking capability. However, much remains to be done to provide a user interface that would, among other things, assist analysts in structuring their thoughts and ideas on the computer. Work also remains to be done on improving the diagnostic utility's user interface.
# Table of Contents

Abstract ii

Table of Contents iii

List of Tables v

List of Figures vi

<table>
<thead>
<tr>
<th>Chapter 1</th>
<th>Introduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Motivation</td>
</tr>
<tr>
<td>1.2</td>
<td>Background</td>
</tr>
<tr>
<td>1.3</td>
<td>Research Questions</td>
</tr>
<tr>
<td>1.4</td>
<td>Contribution</td>
</tr>
<tr>
<td>1.5</td>
<td>Organization of the Thesis</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 2</th>
<th>CASE Tool Survey</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>2.2</td>
<td>The Tools</td>
</tr>
<tr>
<td>2.3</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 3</th>
<th>Object-Oriented Enterprise Modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>3.2</td>
<td>Introduction to OOEM</td>
</tr>
<tr>
<td>3.3</td>
<td>Object Hierarchy</td>
</tr>
<tr>
<td>3.4</td>
<td>The Object Template</td>
</tr>
<tr>
<td>3.5</td>
<td>Scope of Knowledge and Influence of Objects</td>
</tr>
<tr>
<td>3.6</td>
<td>The Modeling Rules</td>
</tr>
<tr>
<td>3.7</td>
<td>Building OOEM Models</td>
</tr>
<tr>
<td>3.8</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 4</th>
<th>Logical Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>4.2</td>
<td>OOEM and OOEMCT</td>
</tr>
<tr>
<td>4.3</td>
<td>Object Hierarchy Revisited</td>
</tr>
<tr>
<td>4.4</td>
<td>Metamodel of OOEM</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 5</th>
<th>System Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>Development Platform</td>
</tr>
<tr>
<td>5.3</td>
<td>System Design Overview</td>
</tr>
<tr>
<td>Chapter 6</td>
<td>Functional Design</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>6.2</td>
<td>Hierarchy of Functional Components</td>
</tr>
<tr>
<td>6.3</td>
<td>Conclusion</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 7</td>
<td>Empirical Study</td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>7.2</td>
<td>Definition of Usability</td>
</tr>
<tr>
<td>7.3</td>
<td>Objectives of the study</td>
</tr>
<tr>
<td>7.4</td>
<td>Methodology</td>
</tr>
<tr>
<td>7.5</td>
<td>Methodology Framework</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 8</td>
<td>Discussion of Results</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>8.2</td>
<td>Overview</td>
</tr>
<tr>
<td>8.3</td>
<td>Subject Background Questionnaire</td>
</tr>
<tr>
<td>8.4</td>
<td>Study Feedback Questionnaire</td>
</tr>
<tr>
<td>8.5</td>
<td>Summary for Study Feedback Questionnaire</td>
</tr>
<tr>
<td>8.6</td>
<td>Post Study Interview</td>
</tr>
<tr>
<td>8.7</td>
<td>Summary</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Chapter 9</td>
<td>Research Questions and Future Research</td>
</tr>
<tr>
<td>9.1</td>
<td>Research Questions</td>
</tr>
<tr>
<td>9.2</td>
<td>Future Research</td>
</tr>
<tr>
<td>9.3</td>
<td>Conclusion</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
</tr>
<tr>
<td>Appendix 1</td>
<td>Session Introduction</td>
</tr>
<tr>
<td>Appendix 2</td>
<td>How to Run a Session</td>
</tr>
<tr>
<td>Appendix 3</td>
<td>Inventory List</td>
</tr>
<tr>
<td>Appendix 4</td>
<td>Consent Form</td>
</tr>
<tr>
<td>Appendix 5</td>
<td>Case Study</td>
</tr>
<tr>
<td>Appendix 6</td>
<td>Investigator Remarks</td>
</tr>
<tr>
<td>Appendix 7</td>
<td>Questionnaire #1 Subject Background Questionnaire</td>
</tr>
<tr>
<td>Appendix 8</td>
<td>Questionnaire #2 Study Feedback Questionnaire</td>
</tr>
<tr>
<td>Appendix 9</td>
<td>Post Study Interview Record</td>
</tr>
<tr>
<td>Appendix 10</td>
<td>Table Definitions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Question</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Question 1: OOEMCT Made It General Easy to Build Model?</td>
<td>78</td>
</tr>
<tr>
<td>2</td>
<td>Question 2: OOEMCT Helped in Isolating and Identifying Constructs?</td>
<td>79</td>
</tr>
<tr>
<td>3</td>
<td>Question 3: OOEMCT Helped in Keeping Track of Constructs?</td>
<td>80</td>
</tr>
<tr>
<td>4</td>
<td>Question 4: OOEMCT Helped You Remember Where You Were?</td>
<td>81</td>
</tr>
<tr>
<td>5</td>
<td>Question 5: Was The Model Viewer Window Useful?</td>
<td>82</td>
</tr>
<tr>
<td>6a</td>
<td>Question 6: OOEMCT Made It Easy To Keep Track of Links?</td>
<td>83</td>
</tr>
<tr>
<td>6b</td>
<td>Consolidated Results from Question 6</td>
<td>84</td>
</tr>
<tr>
<td>7</td>
<td>Question 7: OOEMCT Made It Easy To Trace A Request?</td>
<td>84</td>
</tr>
<tr>
<td>8</td>
<td>Question 8: OOEMCT Checking Resulted In Fewer Mistakes?</td>
<td>85</td>
</tr>
<tr>
<td>9</td>
<td>Question 9: OOEMCT Useful In Building Bigger And More Complex Models?</td>
<td>86</td>
</tr>
<tr>
<td>10</td>
<td>Question 10: OOEMCT Could Help Build Models Quicker?</td>
<td>86</td>
</tr>
<tr>
<td>11</td>
<td>Question 11: OOEMCT Made It More Enjoyable?</td>
<td>87</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The Object Hierarchy of OOEM</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>An OOEM Internal Object Template</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>Default Assumption about Object Knowledge</td>
<td>17</td>
</tr>
<tr>
<td>4</td>
<td>An Upside-Down Tree Structure</td>
<td>24</td>
</tr>
<tr>
<td>5</td>
<td>OOEM Object Hierarchy</td>
<td>26</td>
</tr>
<tr>
<td>6</td>
<td>OOEMCT Metamodel</td>
<td>27</td>
</tr>
<tr>
<td>7</td>
<td>Modules in OOEMCT</td>
<td>35</td>
</tr>
<tr>
<td>8</td>
<td>Drop-down Combo Box Activated</td>
<td>37</td>
</tr>
<tr>
<td>9</td>
<td>An Example Of A Error Message</td>
<td>39</td>
</tr>
<tr>
<td>10</td>
<td>The Standard Interface Layout</td>
<td>41</td>
</tr>
<tr>
<td>11</td>
<td>“Hint” appearing over a form</td>
<td>44</td>
</tr>
<tr>
<td>12</td>
<td>Main Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>13</td>
<td>Recursive Algorithm</td>
<td>50</td>
</tr>
<tr>
<td>14</td>
<td>Hierarchy of Functional Sub-Systems</td>
<td>55</td>
</tr>
<tr>
<td>15</td>
<td>The Home Form</td>
<td>56</td>
</tr>
<tr>
<td>16</td>
<td>Guided Model Entry</td>
<td>57</td>
</tr>
<tr>
<td>17</td>
<td>Free Form Entry Interface</td>
<td>59</td>
</tr>
<tr>
<td>18</td>
<td>Direct Entry of Objects Form</td>
<td>60</td>
</tr>
<tr>
<td>19</td>
<td>Link Entries Form</td>
<td>61</td>
</tr>
<tr>
<td>20</td>
<td>Reports Interface</td>
<td>62</td>
</tr>
<tr>
<td>21</td>
<td>The Object Report Window</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>The Requests Report Window</td>
<td>64</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------</td>
<td>----</td>
</tr>
<tr>
<td>23</td>
<td>The Diagnostic Report Window</td>
<td>65</td>
</tr>
<tr>
<td>24</td>
<td>The Model Report Window</td>
<td>66</td>
</tr>
<tr>
<td>25</td>
<td>Scroll Buttons, Edit, and List Boxes</td>
<td>90</td>
</tr>
<tr>
<td>26</td>
<td>Proposed Graphical User Interface for OOEMCT</td>
<td>102</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

This thesis is motivated by the desire to build a semantically-enhanced Computer-Aided Software Engineering (CASE) tool that systems analysts would find useful in helping them build object-oriented enterprise models.

The tool would incorporate the unique ability to evaluate candidate models against a set of pre-defined semantic rules. The rules are those of the Object-Oriented Enterprise Modeling (OOEM) method proposed by Wand and Woo (1993) and discussed in detail in Chapter 3.

Consequently, the tool will be referred to as the OOEM CASE Tool (or OOEMCT) throughout the remainder of this thesis.

Because OOEM’s unique contribution to Object-Oriented Analysis (OOA) is in its rule-driven approach to modeling, OOEMCT is similarly endowed with a unique ability to store and process these rules against domain information supplied by systems analysts.

OOEMCT will use its special ability to process rules in two ways. First, analysts will be able to use the rules to systematically guide them when building new models. Second, analysts may also use the rules to evaluate candidate models stored in OOEMCT’s data base. The evaluation criteria used will naturally be the candidate model’s adherence to, or departure from, the modeling rules of OOEM.
The objective of this thesis is to test the usefulness and usability of a semantically-enhanced object-oriented CASE tool built upon the modeling rules of OOEM. The dimensions of this objective are two-fold:

1. To explore the theoretical and technical feasibility surrounding the development of a rule-processing CASE tool, and
2. To test the usefulness and usability of the CASE tool within its intended user audience.

Usefulness and usability are critical measures of user acceptance and therefore a tool's success at fulfilling its intended purpose.

The successful development of a computerized, and therefore algorithm-driven, CASE tool is particularly poignant in demonstrating the viability of OOEM's rule-driven approach to object-oriented enterprise modeling. A tool with this capability would help users from diverse backgrounds build models of higher quality and consistency. This was previously impossible.

1.2 Background

A survey of current CASE Tools, three of which are reviewed in this thesis, consistently reveal an absence of a built-in semantic integrity checking utility built for or around the tools. The tools were indeed very powerful and were able to detect what are essentially syntactic violations in the model. For example, entities that exist with no relation to other constructs in the model are easily identified. However, the tools were unable to detect semantic gaps in models. For example, the tools could not distinguish if an object declared in the model by a user was, in fact,
an object in its own right. Users take full responsibility for identifying objects and the tools simply work on the assumption that the user is justified in his or her decision.

This thesis takes the view that it is possible and indeed important, for a CASE tool to assist users in building and checking for semantically consistent object-oriented enterprise models. But a tool capable of performing such semantic checks would first require a set of rules or guidelines upon which these checks may be based.

One appropriate candidate is the OOEM methodology developed by Wand and Woo (1993). Their methodology is unique in that it offers a set of clearly defined semantic rules that analysts may use to guide model building or check models for internal and external validity from an object-oriented modeling viewpoint. Take for example one competing modeling approach that suggests looking for candidate objects in the problem domain by identifying things with “crisp boundaries” (Rumbaugh et. al. 1991). In instances such as these where object identification is so completely dependent on subjective interpretation, there is little wonder then that so few CASE tools even attempt to incorporate a utility that checks for semantic correctness.

However, this situation provides an excellent opportunity for OOEM to demonstrate its power. The task at hand is therefore to determine if it is indeed possible and desirable to incorporate semantic rule processing in a CASE Tool for object-oriented enterprise modeling. Hence the genesis of OOEMCT and this thesis.
1.3 Research Questions

The development of OOEMCT raises several fundamental questions regarding the formalization of OOEM; in particular the rules as they are applied in the stricture of a computerized environment. The questions are as follows:

1. Can the rules of OOEM remain ontologically expressive and yet be sufficiently formalized to be captured within a computerized tool?

2. In applying the rules, can the tool effectively detect situations where candidate models have not adhered to the rules of OOEM?

3. If OOEMCT is indeed effective in storing and applying the rules of OOEM, will users find such a tool (as implemented) useful? In other words, as Landauer (1995) puts rather succinctly,

   "It is not our job to produce clever computer applications that reduce productivity." (p. 663)

These questions are posed in an attempt to ascertain the viability and value of investing even more resources into the development of OOEMCT into a fully featured product that would rival other commercially available CASE tools in terms of usefulness and usability.

The experience gained from the development of OOEMCT will provide valuable insights into the euphony of metaphysics and the rigor of computer science as they come together in one place.

Results from the user study would also contribute to our learning of what functionality users would expect of such a tool in their work. To a very limited extent, the study will probe the
complex nature of user interface design and cognitive modeling from the responses of study subjects.

1.4 Contribution

The development of OOEMCT offers a unique opportunity to test the application of the ontologically expressive rules of OOEM in an algorithm-driven computerized environment. The stricture of a programming environment will undoubtedly force the formalization of, heretofore, subjective interpretations and extrapolations of events and entities in the modeled domain.

It furthers allows us to gain an understanding of users’ expectations from such a tool and starts us early along the road in a heuristic approach towards understanding the cognitive processes users encounter when using a tool of this nature. This is possible only through direct and active experience with an early prototype like OOEMCT.

Finally, the tool fills an important gap in the holistic overture of OOEM as a significant contribution to the field of object-oriented approaches to modeling.

1.5 Organization of the Thesis

The remainder of this thesis is organized as follows: CASE Tool Survey, Object-Oriented Enterprise Modeling, Logical Design, System Design, Functional Design, Empirical Study, Discussion of Results, Research Questions and Future Research, Bibliography, and Appendices.

Chapter 2 presents a survey of three prominent CASE Tools. The first is based on a structured
analysis approach while the other two are based on object-oriented approaches. The objective of
this chapter is to highlight the originality of our research and thence the uniqueness of OOEMCT.

Chapter 3 briefly outlines the theoretical foundation, constructs, and assumptions used when
modeling with OOEM. A structured approach to model building with OOEM “by hand” is also
presented. The objective of this chapter is to introduce OOEM and present a commonly used
heuristic approach to building models with OOEM rules.

Chapter 4 discusses the logical design of OOEMCT. The chapter offers new assumptions and
adaptations to the theory that came about as a result of implementation requirements. This
chapter presents our findings in terms of the theoretical feasibility of building OOEMCT.

Chapter 5 delves into the details of the design and implementation of OOEMCT including a
discussion of interface design and human-computer interaction issues. The algorithm used in
processing the internally stored rules of OOEM is also presented in this chapter. The objective of
this chapter is to introduce readers to the rationale behind the “look-and-feel” of OOEMCT while
discussing issues of a technical nature.

Chapter 6 explains the functional features of OOEMCT. It’s purpose is to explain the various
functions built into OOEMCT, how each is invoked, and the expected response from the system
when each is invoked.

*Introduction:* 6
Chapter 7 fully describes the empirical study used in evaluating OOEMCT's usefulness and usability. Study methodology as well as data collection and analysis techniques are presented. The purpose of this chapter is to explain the terms usefulness and usability and describe in detail the study setup, conditions, and constraints.

Chapter 8 contains a discussion of the results from the empirical study. It presents a number of interesting hypotheses concerning data collected and observations from the empirical study. The purpose of this chapter is to present an interpretation of the findings and highlight the major learning experiences derived from the empirical study.

Chapter 9 wraps up the thesis by revisiting the research goals and questions. The focus of this chapter is to tie our current research goals with our learning experience and suggest a framework for future research efforts.

A Bibliography follows Chapter 9 and it contains a complete list of literary resources used in this thesis.

Finally, all instructions, forms, questionnaires, and the actual case-study used during the empirical study are included in the Appendices. Table definitions for OOEMCT are also attached.
Chapter 2

CASE Tool Survey

2.1 Introduction

This chapter will survey three prominent CASE tools now commercially available on the market. The purpose of this survey is to provide an overview of their principal features and in so doing illustrate how OOMCT intends to position itself in terms of its functionality in the CASE tool market.

2.2 The Tools

The first CASE tool is based on a structured approach to analysis: that of Entity-Relationship modeling. The second and third CASE Tools are based on object-oriented approaches to modeling.

2.2.1 Texas Instrument's Composer

Composer is a tool that supports a range of software engineering activity through several smaller, more focused toolsets. The toolsets cover planning, analysis, design, and construction activities and are aptly named as such (Composer, 1996).

The Planning Toolset is used for establishing the coarse grained scope of single or multiple projects. It does this by using high level data and activity analysis that is often concurrent with business process design activity. The results are application development projects that are clearly defined in terms of business requirements.
The Analysis Toolset focuses on a specific area of the business defined earlier in the planning phase. Users (team members) concentrate on a particular area of the business by refining three components: a data model, an activity model, and an interaction model. The result is a Business Model which is essentially a conceptual representation of the business devoid of implementation details. Information captured through any of the toolsets is shared and commonly accessible through a central data repository known as the Encyclopedia.

The Design Toolset crosses the threshold of planning into the realm of implementation. Details such as procedure flows, user interfaces, and database management systems are considered. The result is a conceptual representation of the physical implementation of a business system.

Finally, the Construction Toolset uses information collected during analysis and design to automatically construct complete application systems. The Workstation Construction Toolset allows developers to generate, test, modify, and re-test an entire system or sub-system.

Perhaps the most significant component of Composer is its Encyclopedia. The Encyclopedia is a sophisticated repository that ensures model concurrency and integrity across multiple toolsets and users without contention. It does this by centrally storing definitions of objects and their relationships so as to achieve extensive completeness and consistency checking based on methodological guidelines. The checking is deferred until requested so as not to enforce an order of work. Checking may also be focused on a particular modeling object, a diagram or the...

---

1 The check is to ensure that user-defined objects are complete and consistent across the toolsets. This is basically a check for syntax and data consistency.
2.2.2 MetaCase Consulting’s MetaEdit Plus

*MetaEdit Plus* is an information systems and business process modeling tool. The tool covers information systems planning, architecture definition, analysis, design, and code generation activities within an integrated tool structure (MetaEdit+, 1995).

*MetaEdit Plus* is a highly flexible tool. The tool itself supports a number of popular object-oriented modeling methodologies such as those by Booch, Rumbaugh, and Coad/Yourdon.

The *Matrix Editor* within the tool provides for the automatic cross-sectional analysis of a system, to show very clearly where system and subsystem boundaries are within a model. Much like a pivot table, elements in a model can be selectively juxtaposed in a spreadsheet-like matrix table for maximum information yield and clarity.

The *Object Repository* helps facilitate object reuse while maintaining and enforcing consistency between business processes and system descriptions.

Finally, *MetaEdit Plus* is also a meta-CASE tool. In other words, it will allow users to incorporate custom methodologies that the tool will then support.

*CASE Tool Survey: 10*
2.2.3 Protosoft’s Paradigm Plus

*Paradigm Plus* is presented by its developers as a powerful object-oriented development tool that provides support for the entire software development lifecycle (Paradigm Plus, 1995).

Like many other object-oriented modeling tools, it too features support for a broad array of methodologies, platforms, networks, implementation languages, and customization options. The presence of an *Object Repository* completes the suite of commonly found sub-components in a tool of this genre.

To be specific, *Paradigm Plus* features a *Diagram Editor* that users can use to produce and attach drawings composed from built-in graphic primitives such as lines, circles, and polygons. Each primitive or symbol carries the ability to “explode” into one or more sub-diagrams thus enabling diagram leveling to be performed.

The *Automatic Diagram Layout and Generation* tool features the ability to automatically align diagram elements and through the “automatic diagram” feature, it further allows users to explore objects and their relationships to other objects in a method-specific diagram.

The *Repository Browser* is used to view and edit all objects and relationships stored in the object repository.

The *Table Editor* allows for easy inspection, comparison, and editing of object properties.
Most relevant to this thesis is a *Scripting Language* in *Paradigm Plus* that provides for completeness checking\(^2\).

### 2.3 Conclusion

The preceding review support our belief that many, if not all, of the currently available CASE tools are unable to provide a check for semantic integrity for models built with the tool. Besides the three CASE tools reviewed above, the other tools surveyed were: ICONIX Software Engineering’s *ICONIX PowerTools* (ICONIX, 1995), Project Technology’s *BridgePoint CASE* (BridgePoint, 1995), and Mark V Systems’ *ObjectMaker* (ObjectMaker, 1993). Although every tool surveyed provided a syntactic check\(^3\) or even a meta-tool editing capability, none provides a pre-built rule-processing component or even allowed users to easily write or attach a rule-processing module to their tool.

Clearly then a tool with semantic rule processing capability like OOEMCT would stand alone.

---

\(^2\) Like *Composer*, the check is to ensure that user-defined objects are complete and consistent across the toolsets. This is basically a check for syntax and data consistency.

\(^3\) BridgePoint CASE states quite clearly that “It guides your construction of models that are always syntactically correct.”
Chapter 3

Object-Oriented Enterprise Modeling

3.1 Introduction

This chapter is intended to be a primer for those new to OOEM methodology. Research in OOEM is currently ongoing at the University of British Columbia but those interested in learning more about the OOEM methodology should read Wand and Woo, 1993 and Zhao 1995.

The following sections in this chapter will deal with a range of issues about OOEM beginning with a brief introduction to OOEM all through to a framework for building OOEM models “by hand”.

3.2 Introduction to OOEM

Disappointed with current object-oriented analysis techniques that lack strong theoretical foundation, Drs. Wand and Woo published a position paper in 1993 championing the assimilation of ontological principles with object-oriented modeling technology for enterprise modeling.

The marriage of Ontology and object-oriented modeling is a natural one since, befittingly, one of the principal tenets of Ontology proposes that the world is composed of things\(^4\) (Bunge, 1977).

\(^4\) One might think of “things” as also “objects”.
Many object-oriented modeling methodologies now appearing seem to be, more often than not, “object wrappers” around traditional structured methodologies (e.g. Bailin, 1989 and Rumbaugh, et al. 1991). OOEM therefore strives towards presenting a methodology that is theoretically sound and can lay true claim to object-orientation.

3.3 Object Hierarchy

Earlier research efforts on OOEM (Wand and Woo, 1993; Zhao 1995), had identified and divided objects into two broad categories: Clients and Internal Objects. Clients are, in fact, External Objects and are not considered a part of the system directly under study. Hence little information about Clients was expected to be available (see Section 3.5: Scope of Knowledge and Influence of Objects below). Their role was seen as, more or less, one of “triggering” events and activities (of objects) within the domain of interest. Figure 1 below depicts the object hierarchy of OOEM.

![Object Hierarchy Diagram](image)

Figure 1 The Object Hierarchy of OOEM
3.4 The Object Template

An object representation in OOEM is captured in an object template as illustrated in Figure 2 below. The template is a reproduction of an earlier work by Zhao 1995. While the template is said to apply only to Internal Objects modeled in the View it may, in fact, be used for modeling External Objects where more information about the latter is available (see Section 3.5: Scope of Knowledge and Influence of Objects). The OOEM object template embodies all object properties essential to modeling objects mapped in the problem domain viz object name, sets of Services, Interface Attributes, Internal Attributes, and Requests Generated.

<table>
<thead>
<tr>
<th>Object Name</th>
<th>Services</th>
<th>Interface Attributes</th>
<th>Internal Attributes</th>
<th>Request Generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Service 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Service 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 2** An OOEM Internal Object Template (Zhao, 1995)
The following is how an OOEM Internal Object template and its properties should be interpreted.

In Figure 2 the template depicts an OOEM Internal Object with a complement of only two Services. A Service is an operation internal to an object and its activation is deemed necessary toward satisfying a pending Request either in whole or in part. A Request, on the other hand, is a mandate or petition received either from a source outside the domain of influence (a Client) or from a source internal to the system (i.e. an Internal Object). In order to invoke a Service, an incoming Request always requires knowledge of the receiving object’s Interface Attribute. One might think of an Interface Attribute as a shared protocol between the two elements.

Each Service is invoked via an Incoming Interface Attribute and may, in certain circumstances, return a Returning Interface Attribute (e.g. Service 1) to complete the call. The performance of a Service may also result in the propagation (or generation or spawning) of new Requests to other objects in the model.

Objects may also possess Internal Attributes that Services access in the course of completing their operations. Internal Attributes contain state information about Objects. Objects are sole custodians of their Internal Attributes and so modification of an object’s Internal Attributes by other objects is only possible through invoking the custodian object’s Services by way of a Request.
3.5 Scope of Knowledge and Influence of Objects

One distinction between Internal and External Objects in OOEM is the premise that while the former is expected to always yield full knowledge of its Internal Attributes and Services, the latter may choose to withhold such information from scrutiny. Figure 3 below depicts the default assumption used in OOEMCT with regard to the scope of knowledge and influence of objects.

One should note, however, that there is nothing in the assumption expressed in Figure 3 that will preclude the use of full knowledge about External Objects should it become available during the modeling process.

3.6 The Modeling Rules

The OOEM methodology sets itself apart from other modeling methodologies by offering a set of seven rules that semantically checks and enhances the mapping integrity of a candidate model. The following sections will present a brief outline of each of the seven rules. Again, a fuller discussion of these rules can be obtained from reading Wand and Woo, 1993 and Zhao 1995.
3.6.1 Rule #1: The Scope Identification Rule

The Scope Identification Rule, as its name implies, is intended to help analysts focus their attention only on the essential elements in the problem domain. Specifically, it helps determine which objects should be included and which properties (attributes and actions) of those objects should be included. A principal activity in applying this rule is the delineation of objects internal and external to the modeled system. Equally important is the identification of Requests originating from each object external to the system (Client).

3.6.2 Rule #2: The Object Identification Rule

The Object Identification Rule specifies very clearly the circumstances under which an object may (or may not) be included in the model. In order to be included in the model, an object must provide or request a service as part of its role in system activities.

3.6.3 Rule #3: The Service Inclusion Rule

The Service Inclusion Rule similarly stipulates the conditions under which a service may be included in an object. To be included, a service must be directly invoked by at least one request in the model.

3.6.4 Rule #4: The Attribute Inclusion Rule

The Attribute Inclusion Rule exists to ensure that attributes included in a model must serve one of two purposes. It must either serve as an interface for other objects (i.e. known to another object) or it must be used (accessed) by an internal service of the custodian object.
3.6.5 Rule #5: The Attribute Ownership Rule

The Attribute Ownership Rule ensures encapsulation by making it necessary for all attributes appearing in a model to be traced to a unique custodian object. In other words, all attributes appearing in a model must have an owner and only that owner may have the privilege to modify the attribute.

3.6.6 Rule #6: The Aggregation and Decomposition Rule

The Aggregation and Decomposition Rule states that a composite object may only be included in a model if it provides a service not already provided by any of its component objects. Also, a composite object may only display properties that are not already properties exhibited by its component objects.

3.6.7 Rule #7: The Generalization and Specialization Rule

The Generalization and Specialization Rule aims at simplifying a model by identifying objects with common services and organizing them into a broader object class. This more “general” class will then provide services that are common to all its specialized (or sub-class) objects. Specialized objects should then only possess services that are unique to themselves. An example might be tellers at a bank. While all tellers share certain common characteristics and abilities, each category of teller (e.g. quick deposit tellers, current account tellers, and investment tellers) will each have their own special ability. For example, all tellers accept cash but only current account tellers can dispense checks.
3.7 Building OOEM Models

Building an OOEM "by hand" need not necessarily be an unstructured and haphazard activity.

This section will first provide a structured approach to building an OOEM model and then present some trouble shooting tips to deal with important "frequently asked questions" that might be encountered.

3.7.1 A Structured Approach

Although not mandatory, the process of building an OOEM model would benefit greatly from adhering to the specific sequence of activity as prescribed below:

1. Identify and isolate all candidate Clients in the problem domain;
2. Going through every candidate Client in turn, for each Client;
   2.1 Identify and isolate all Requests originating from each Client;
   2.2 Going through every Request in turn, for each Request;
      2.2.1 Identify the object receiving the Request;
      2.2.2 Identify the Service invoked in the object;
      2.2.3 Identify the Interface Attribute used to invoke the Service;
      2.2.4 Identify the Response\(^5\) returned by the Service;
      2.2.5 Identify all Internal Attributes used (accessed) by the Service (if any);
      2.2.6 Identify all Requests spawned by the Service (if any);
      2.2.7 If there are Requests spawned by the Service, repeat step 2.2.

---

\(^5\) A Response models information returned to the requesting object as a result of the initial request.

Object-Oriented Enterprise Modeling: 20
3.7.2 Troubleshooting

Should one encounter problems while using the above algorithm, then the following list of troubleshooting tips can be used to help determine the nature or cause of the problem. This list is by no means exhaustive but intended to cover only the most frequently encountered questions while building an OOEM model.

Step 1. If one cannot identify and isolate any candidate Clients in the problem domain, then the suitability of the case for use with OOEM needs to be reviewed. In other words, the tool does not seem to fit the problem. Consult Rule #1 The Scope Identification Rule.

Step 2.1. If a Client does not generate a Request, then reconsider if the object might be of another object type i.e. an Internal Object or Provider. A Provider is an object that is external to the system yet is not a Client of the system. An example of a Provider is a credit card company that regularly submits a list of invalid card numbers to vendors to help them avert fraud. Note that the credit card company’s operations might not be in the scope of interest if one were to study a vendor’s operation, yet the list it provides may be vital to the processes under scrutiny for the vendor. So a Provider is neither a Client (e.g. card holders) nor an Internal Object. Finally, consider if the object might not be a Client at all. Consult Rule #1 The Scope Identification Rule.

Step 2.2.2. If a Service cannot be identified for the receiving object then reconsider if the Request was sent to the right object. Consult Rule #3 The Service Inclusion Rule. Alternatively, consider if the object might be a component (or specialized) object. If this is the case then the Service is probably available from the aggregate (or generalized) object. Consult Rule #6 The
Aggregation and Decomposition Rule and Rule #7 The Generalization and Specialization Rule.

Finally, consider if the object might be a Provider. See Section 4.3 Object Hierarchy Revisited.

Steps 2.2.3 and 2.2.4. If a particular Attribute has already been identified previously as belonging to another object, then consider if it might be the case where the receiving object needs to spawn a new request to the custodian object (object currently owning the attribute) requesting it to provide the information it (the receiving object) now requires. Consult Rule #5 The Attribute Ownership Rule. Alternatively consider if it might be better to change the custodian object for the Attribute.

3.8 Conclusion

The chapter briefly discussed the theoretical foundation, constructs, and assumptions of modeling with OOEM. It has also presented a structured approach to model building without machine assistance and a list of suggestions to deal with commonly experienced difficulties while building an OOEM model. The next chapter will delve into the issues of incorporating the knowledge and practice of OOEM model building into a computerized CASE tool.
Chapter 4

Logical Design

4.1 Introduction

This chapter continues from the theoretical underpinnings of OOEM described in the previous chapter. It will discuss the logical design of OOEMCT including extensions and adaptations to OOEM held necessary in order for the tool to become technically feasible.

4.2 OOEM and OOEMCT

This thesis is concerned with the theoretical foundation of OOEM in so far as it affects the implementation of OOEMCT and, more critically, the perceived usefulness of models generated by the Tool. In this regard, it is particularly important to note that:

1. Ontology does not concern itself with business semantics. An ontologically correct model, in itself, does not guarantee the model's integrity from a business meaning perspective.

2. OOEM is fundamentally driven by services of objects within the system propagating requests to one another. This is always instigated by a request or requests received from external client objects (Zhao, 1995). There is no Ontological equivalent for this phenomenon.

The implications of the first statement is that analysts must use personal judgement, diligence, domain knowledge, and experience in building and/or interpreting models deemed acceptable by OOEMCT.
The second statement implies that analysts must be mindful of the series propagation of requests in OOEM - and more critically, OOEMCT - and the rather intricate web of dependencies they create within a model. To explain this another way, one could think of an OOEM model’s structure as not unlike that of the upside-down tree structure used so often to teach first-year data structure students in computer science (see Figure 4 below).

![An Upside-Down Tree Structure](image)

In Figure 4, Node A can be thought of as a Client despatching requests to objects represented by Nodes B and C (see Section 3.3 Object Hierarchy below for object types used in OOEMCT). Paths 1 and 2 to Nodes B and C respectively, can be thought of as Request paths. Nodes B and C, in trying to satisfy the requests, spawn additional requests to other objects represented by Nodes D, E, F, and G. And because of their almost identical structure, OOEM models and other tree structures suffer from the tree management woes (e.g. pruning and grafting) already so well studied and documented in data structure text books.
In summary, the design of OOEMCT would have to accommodate frequent changes to the model in as easy and as unintrusive a manner as possible.

4.3 Object Hierarchy Revisited

Although the object hierarchy defined previously in Figure 1 was adequate for modeling "by hand", during the development of OOEMCT it became apparent that some objects clearly exhibited characteristics that defied classification in the two previous categories. These objects were external to the domain and outside the scope of interest yet they made NO request to the system. Their role was simply that of providing a service in response to requests from the system (e.g. information requested from a credit bureau). Hence it was thought appropriate to place these objects into a new and distinct category by themselves. This new category has become known as Provider Objects or simply Providers. Figure 5 below depicts the new object hierarchy of OOEM as implemented in OOEMCT.
4.4 Metamodel of OOEM

The metamodel of OOEM upon which OOEMCT was built is presented below in Figure 6. The metamodel is an abbreviated diagram (for parsimony) and does not show the specialization of external objects and their associated relationships.
The metamodel depicts the constructs of OOEM as boxes and the participatory relationships between them as arrows. The accompanying cardinal numbers at each end of a relationship serve to define the boundary of participation for each construct. Finally, the object hierarchy (signified by an inverted three-pronged fork) serves to more clearly illustrate object participation in BOTH their generalized and specialized roles.

The relationship Generate, for example, shows that objects in general may generate no Requests or any number of Requests. However, Clients must generate at least one Request. Requests, on the other hand, must be generated by at least one object in the case of Clients or spawned by a Service in the case of Internal Objects. In a further example, the relationship “Sent” shows that
objects may receive no Requests or any number of Requests. Requests, on the other hand, may be sent by any number of objects.

Besides providing an easy to understand mapping of the OOEM constructs, the metamodel also represents a smooth transition of the OOEM rules from its Ontological origins to a new digitized domain. A complete explanation of the metamodel and its connection with the seven rules of OOEM follows.

4.4.1 Rule #1: The Scope Identification Rule

The Scope Identification Rule, as its name implies, exists to define the boundary of interest in the candidate model. The Rule emphasizes the separation of activities and events between those within the sphere of interest and influence (the enterprise) and those without (the environment). Objects outside the system are said to generate “Requests” to the system. The system then enters a state of instability until the requests are satisfied by events and activities that take place within the system.

The metamodel encapsulates this rule by requiring that objects be assigned to one of two categories or roles; they must either be Internal Objects or External Objects. Once identified, objects are obliged to assume their common and specialized participatory roles as defined in the metamodel. In fulfilling these “participatory obligations” objects implicitly define the scope of the analysis.

At this point, it is important to differentiate between the OOEM terms Perspective and View. A
Perspective is described as “the aspects of the system necessary to describe the organizational activities invoked as a consequence of the requests of one external thing (client).” while a View is “the union of perspectives for a set of given clients.” (Wand and Woo, 1993)

Hence, in the ensuing sections of this chapter, it will become clear that if candidate objects included in the View fulfill their assigned participatory roles within the intent of the metamodel, the scope identification rule will be met naturally.

### 4.4.2 Rule #2: The Object Identification Rule

The Object Identification Rule specifies that each object must provide or request a service as part of system activities to be included in the View.

We will first deal with Internal Objects. The participatory relationship Provide ensures that all Internal Objects provide at least one Service.

Working in tandem, the participatory relationship Generate ensures that all Clients generate at least one Request in order to be included in the View.

The observant will note that this leaves Providers largely unrestricted and undefined. But since Providers assume a “peripheral” role in the analysis effort anyway, OOEMCT is programmed to encourage analysts to rethink and recast the role of Providers whenever encountered. In other words, there is a very strong likelihood that the Provider might be better cast as a Client or Internal Object.
4.4.3 Rule #3: The Service Inclusion Rule

The Service Inclusion Rule ensures that every Service included in an object is directly invoked by at least one Request in the View.

In the metamodel, this rule is primarily enforced by the relationship Invoke and its cardinal specification.

As further peripheral constraints, all Internal Objects to be included in the View must provide at least one Service⁶ (see Section 4.4.2, Rule #2: The Object Identification Rule). Also, a Request must either be generated by at least one Client or spawned by exactly one Service. The relationships Generate and Spawn and their cardinal specifications jointly enforce this. These constraints ensure that every Service and Request appearing in the View is not spurious.

Pieced together, the stipulations successfully express the imperative of this OOEM rule: that every service included in an object is directly or indirectly invoked by at least one request in the View.

4.4.4 Rule #4: The Attribute Inclusion Rule

The Attribute Inclusion Rule states that a property of a thing not “known” to other things or that never changes should not be included. This requires that attributes be subdivided into two categories; Interface Attributes and Internal Attributes. The basis for division is as follows.

⁶ Note, however, that External Objects are given the option of yielding or not yielding knowledge about their internal services and are therefore exempt from this rule.
Interface Attributes are attributes that are stipulated as being necessarily “known” to other objects so that they may become a “protocol” of communication. In the metamodel, the relationships Interface and Own help express this sub-rule. Note that each Request must deal with exactly one Interface Attribute which is, in turn, uniquely owned by an Internal Object in the View (see Rule #5 below). Internal Objects may, of course, own more than one Interface Attribute for a multiplicity of transactions to become possible.

Internal Attributes are attributes of a different nature. These attributes must be used or affected by at least one Service belonging to the object. In the metamodel, the relationship Access and its cardinal specification ensures that Internal Attributes within an object included in the View must be accessed by at least one Service belonging to that object. Note, however, that not every Service must access an Internal Attribute to be included in the View.

Finally, External Objects may choose to reveal or not reveal their properties.

4.4.5 Rule #5: The Attribute Ownership Rule

The intention of the Attribute Ownership Rule is to ensure data integrity by identifying a unique custodian of an attribute and granting only that custodian the ability to modify that attribute through its (the custodian’s) internal services. It is perhaps best explained through an analysis of three participatory relationships in the metamodel viz: Own, Access, and Provide.

A particular attribute must be uniquely Own-ed by an object in the View. This same object also Provide-s one or more Services that Access-es (i.e. use or affect) that attribute. Together with the
fact that constructs in a metamodel are not to be instantiated, the Attribute Ownership Rule then enforces (albeit in a slightly roundabout manner) the rule that attributes may only be modified by the internal services of their custodian objects.

4.4.6 Rule #6: The Aggregation and Decomposition Rule

The Aggregation and Decomposition Rule expresses the idea that composite objects should only be included if they provide services that are not already provided by any of its components. The metamodel through the participatory relationship *Provide* and its cardinal specification helps ensure that each *Service* can only be attributed to a single *Internal Object* in the *View*. This "internal layer" of semantic integrity may be quite easily controlled and enforced programmatically in OOEMCT. In other words, OOEMCT will check the *Services* defined in aggregate objects and ensure that component objects do not already offer an identical *Service*.

4.4.7 Rule #7: The Generalization and Specialization Rule

The Generalization and Specialization Rule is, in OOEMCT, handled in a very similar manner to the Aggregation and Decomposition Rule explained in the preceding section. This rule similarly seeks to consolidate duplicate services into a super-class object that centrally manages these *Services* for sub-class objects. Again, OOEMCT will check the *Services* of all sub-class objects in the model and prompt the user for a deliberate action to consolidate each replicated *Service*. 

*Logical Design: 32*
4.5 Conclusion

During the development of OOEMCT, it became clear that the constructs of OOEM as they stood and their subjective interpretation when applied against domain information was now wholly inadequate for the stricture of a programming environment. Hence, it became necessary to develop strategies that would bridge that gap during the logical design stage so physical design could begin.

The major achievements during this stage were the:

- Recognition that Ontological correctness and correct business semantics were not necessarily congruent;
- OOEM models were basically upside-down tree structures in nature;
- The recognition of a second category of external objects i.e. Provider objects;
- The development of a metamodel and thence the transfer of the Ontologically expressive rules of OOEM into a digitally processed environment.

The next chapter will deal with the system design of OOEMCT with these changes incorporated.
Chapter 5

System Design

5.1 Introduction

This chapter will elaborate on the development of OOEMCT from a rather technical perspective covering aspects of implementation including choice of development platform, system design overview, selection of interaction style, selection of data store, and designing for usability and usefulness.

5.2 Development Platform

The wide spectrum of development tools available on Microsoft's Windows 95 operating system and its overwhelming popularity in user communities made it a natural choice for the development platform of OOEMCT's prototype.

Naturally, the application will, to a large extent, be shaped by the constructs and facilities available in Windows 95 as the host operating system environment.

5.3 System Design Overview

The OOEMCT's design can, conceptually, be divided into three major sub-components as depicted in Figure 7 below. The first two sub-components (the Interface Sub-Component and Processing Sub-Component) are further divided in three modules each while the third (the Data Storage Sub-Component) stands alone.
The subsequent sections of this chapter will deal with each sub-component in greater detail.

5.3.1 The Interface Sub-Component

This section deals with three major issues of design used for the Interface sub-component; that of selecting an interaction style, designing for usability and usefulness, and designing for cognitive advantage. The learning gained from studying these issues will be consistently applied in the development of each of the three modules in this sub-component.

5.3.1.1 Selection of Interaction Style

According to Shneiderman (1995) the first thing a designer should be aware of is a taxonomy of five primary interaction styles: menu selection, form fill-in, command language, natural language, and direct manipulation. The suitability of a particular style, or combination of styles,
largely depends on the task, computer concept, and interface device in question.

The second thing a designer should be aware of, according to Dr Shneiderman, is that no formal rules exist to help the designer pick the most effective style or combination of styles for a given task! The only answer lies in experimentation to validate and refine a particular design. Although only a secondary objective, the empirical study we present in Chapter 7 makes an attempt to identify and document the interface issues that the form fill-in style brings to users of OOEMCT.

In the guidelines that accompany Dr. Shneiderman’s taxonomy, the form fill-in approach is said to be particularly useful in that it:

- Simplifies data entry;
- Requires modest training;
- Is convenient to offer assistance; and
- Shows context for activity.

The form fill-in style suffers in that it consumes screen space and requires typing skills. However, wherever possible, the use of the menu selection style is made to minimize keyboard entry\(^7\). For example, Figure 8 below shows a drop-down combo-box\(^8\) in action. Users click on the combo-box to “drop-down” a list of items from which to choose. If the item is available, a

\(^7\) According to Dr. Shneiderman’s guidelines, the menu selection style is recommended where reduced keystrokes is desirable.

\(^8\) A combo-box is a context sensitive list of item choices from which a user may choose. The box expands (or “drops”) and collapses upon user command; usually a click on the box.

System Design: 36
single mouse click takes the place of a series of tedious, redundant, and error-prone keystrokes.

As is already obvious from Figure 8 above, the implementation of a “form” in OOEMCT is basically an autonomous instance of a window in Microsoft Windows 95.

The other three interaction styles were not considered for use in the design of OOEMCT because the command language style is suited to power users and requires substantial training and memorization. The natural language style, on the other hand, is suited to situations where there is heavy use of syntax. Finally, direct manipulation, though widely appealing requires the
commitment of substantial design and technical resources in an area where its potential usefulness and application approach is as yet unexplored. As such, these three styles were not selected for use with OOEMCT.

5.3.1.2 Designing for Usability and Usefulness

A search through the literature revealed a consensus among experts in human-computer interface design - the only reliable method for generating quality user interfaces is to test prototypes with actual end users and modify the design based on users' comments and performance (e.g. Gould 1985; Landauer (1995); Myers (1992); Swartout 1982).

Nevertheless, research and experience gained in human-computer interface design have yielded a set of broad guidelines that should serve to guide the software engineer from the outset.

5.3.1.2.1 Interface Design Guidelines

It is not difficult to understand why researchers in human-computer interface design such as Molich and Nielsen (1990) advocate the use of interface design guidelines during the design phase of a software engineering project. At the design phase, following such guidelines impose little extra effort. To support a high level of software usability, a set of usability guidelines was used throughout the entire development effort for OOEMCT. The guidelines are outlined in the next section.

5.3.1.2.2 Usability Checklist

The checklist used in the design of OOEMCT consists of ten criteria, and the definition of each is
given below, quoted directly from Molich and Nielson, (1990)\(^9\). According to the authors, the checklist was used successfully in a study of interface problem recognition involving seventy-seven designers and programmers from industry and academia. The authors also claim that the list corresponds to similar principles described by other researchers.

Note that in the definitions, the use of the term “dialogue” is in regard to the dialogue between the user (human) and the computer.

1. **Simple and Natural Dialogue**

   "Dialogues should not contain irrelevant or rarely needed information. Every extraneous unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. All information should appear in a natural and logical order." (p. 339)

Figure 9 below is an example of an error message generated when a user attempts to add a second receiving object to a request that already has a receiving object assigned. This is, of course, impossible since a request may be sent to only one object.

![Error Message](image)

**Figure 9** An Example Of A Error Message

\(^9\) The first paragraph in each criterion is the direct quote. The second paragraph explains how OOEMCT’s design complies with that criteria.
2. **Speak the User’s Language**

“The dialogue should be expressed clearly in words, phrases, and concepts familiar to the user rather than in system-oriented terms.” (p. 339)

*Figure 9* again, is an example of the preferred use of simple English over technical terms.

3. **Minimize the User’s Memory Load**

“The user’s short-term memory is limited. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. Complicated instructions should be simplified.” (p. 339)

*Figure 10* is an example where relevant contextual information is always provided so the memory load is taken off the user and s/he does not have to remember the elements critical to the current required action.
4. **Be Consistent**

"Users should not have to wonder whether different words, situations, or actions mean the same thing." (p. 339)

"Be Consistent" includes consistency between subsystems (self-consistency) and consistency across independent systems with common user populations (compatibility). In fact, the object-oriented development environment of Delphi, (the development tool used for OOEMCT) through its strong inheritance and rich palette of pre-defined interface objects for Windows 95, already ensures to a large extent, that the resulting application is self-consistent as well as compatible with other applications running in the Windows environment.
In Figure 10, for example, the standard controls such as the Next and Back buttons always mean the same thing in every form. This consistency was very well implemented through the use of object inheritance employed throughout the development process. In other words, a "parent" form with all common controls and functionality was built and then reused as a "child" wherever required (with added customization if necessary). A single change to the "parent" form would then ripple through every "child" form in the entire application.

5. Provide Feedback

"The system should always keep users informed about what is going on, by providing him or her with appropriate feedback within reasonable time." (p. 339)

An example will be the Model Viewer window shown in Figure 16. Through the Model Viewer window, users in the Guided mode are given near instantaneous feedback about the various elements they have entered into the model.

6. Provide Clearly Marked Exits

"Users often choose system functions by mistake and will need a clearly marked 'emergency exit' to leave the unwanted state without having to go through an extended dialogue." (p. 339)

In Figure 10, for example, the "Home" button appears on every form to let users return to the top-most level and exit OOEMCT quickly at any time.

7. Provide Shortcuts

"Clever shortcuts - unseen by the novice user - may often be included in a system such that the system caters to both inexperienced and experienced users." (p. 339)

In Microsoft Windows, this is a fundamental user interface programming requirement.
OOEMCT complies with this programming standard by similarly assigning a single character in each button's label to a "hot key" sequence. The assigned character would appear as an underscored character in the button's label. Frequent users of Microsoft Windows would immediately recognize that holding down the "Alt" key while pressing the underscored character key will provide the equivalent of a mouse click on that button.

8. Provide Good Error Messages

"Precise error messages provide the user with exact information about the cause of the problem. Constructive error messages provide meaningful suggestions to the user about what to do next." (p. 339)

In Figure 9, for example, the error message is very explicit about the cause of the problem. Other error messages have been designed to be equally informative. Wherever relevant, constructive suggestions are also provided.

9. Error Prevention

"Even better than good error messages is a careful design that prevents a problem from occurring in the first place." (p. 339)

Again, a fundamental user interface programming requirement in Microsoft Windows specifies that buttons invoking "irrelevant" functionality in a specific context be disabled (dimmed). In Figure 10, for example, buttons such as "Edit" and "Delete" are disabled if a user has not yet made an entry. This, of course, is because it makes absolutely no sense to allow editing or deletion to a non-existing entry!
10. Help and Documentation

"Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, be focused on the user's task, list concrete steps to be carried out, and not too large." (p. 339)

As OOEMCT is an early prototype, no help is available as yet but "hints" are used wherever possible to help the user understand the function of each interface element. Figure 11, below shows a "hint" appearing momentarily when the mouse cursor moves over the combo-box (the box where users are expected to type in an entry).

![Figure 11 "Hint" appearing over a form](image-url)
5.3.1.3 Designing for Cognitive Advantage

Although the preceding interface design guidelines are invaluable in helping us start early along the road to better interface design, the guidelines themselves are, nevertheless, only "superficial" in their coverage of the complex human-computer interaction processes that take place each time a user sits in front of his or her computer.

The guidelines fall far short of the demands imposed when two active agents, each capable of initiating an exchange of information, are put together in a symbiotic relationship that involves the completion of complex intellectual tasks.

Studying aspects of the interface between user, computer, and task, Card and Moran (1995) identified a sequence of ever-broader interfaces. These interfaces begin with the simple physical interaction between the user and the devices and displays and carries on through to the grappling of complex intellectual tasks. The sequences or "strands" are presented verbatim as follows:

1. *The physical interface*: The user interacts with a system by means of physical input devices, such as a keyboard and a mouse, and output devices, such as a high-quality graphical display.

2. *The cognitive interface*: The user has certain characteristics as an information processor, such as a limited working memory, that together with the goals he is trying to achieve determine his behavior.

3. *The conceptual interface*: The computer system is also a complex information processing mechanism, and the user needs to have some kind of mental model of it in order to effectively interact with it.
4. **The task interface:** Systems are designed to help their users do tasks, not only small, routine tasks, but also larger, difficult intellectual tasks that are the object of the grand visions of personal workstations. (Card and Moran 1995, p. 587)

While the physical interface sequence (or strand) was, for practical reasons, beyond the realm of our control, we were nevertheless very concerned with the mental and cognitive issues involved in the development of OOEMCT. However, a decision to postpone explicit studies into these areas was made because the contribution of each interface sequence was, by itself, worthy of separate investigation and beyond the scope and resources available to this thesis.

5.3.2 The Processing Sub-Component

This section covers two important aspects of the Processing Sub-component; Outline of Processing Algorithms and Emplacing the Diagnostic Facility.

5.3.2.1 Outline of Processing Algorithms

The processing algorithms for OOEM may be broadly divided into the three modules as explained in **Section 5.3 System Design Overview** and **Figure 7** above. The three modules are the **Model Entry Processing** module, the **Model Entry and Link Processing** module, and the **Report Generation Processing** module. An outline of the processing algorithm for each module follows.

5.3.2.1.1 The Model Entry Processing Algorithm

The **Model Processing Module** uses an algorithm that adopts a depth-first search of “Nodes”
(Objects) in the model. Using Figure 4 above as an example, OOEMCT in Guided mode will assist users towards identifying Nodes A, B, and D and then Nodes A, C, and F sequentially. Users may, of course, choose to divert their attention at any “branch” to pursue a path of their interest or they may resume a previous search path from any branching point.

5.3.2.1.2 The Model Entry and Link Processing Module

The Model Entry and Link Processing Module was designed to allow the greatest flexibility in model entry. Hence there is no pre-programmed activity constraint and the user assumes full responsibility for all aspects of model construction with the exception only of linking Requests to Objects. In compliance with OOEM Rules 2, 3, 4, and 5 (see Sections 3.6.2 to 3.6.5) and error prevention design guidelines, the process of linking Requests to Objects will require the user to navigate through a pre-defined order of forms that will help guide and structure information input.

5.3.2.1.3 The Report Generation Processing Module

The Report Generation Processing Module is itself broken into three sub-modules; one for each report type (see Section 6.2.4 Report Retrieval below for complete list of report types).

The algorithm for generating the Objects Report window picks up all objects stored in the model and reports on the object type, Services, Interface Attributes, and Internal Attributes used by each Service of every identified object. It is simply an inventory of all objects in the model.

The algorithm for generating the Requests Report window picks up all Requests stored in the
model. It does not reveal the originator nor destination object of a \textit{Request}. Again, it is simply an inventory of all \textit{Requests} in the model.

The algorithm for generating the \textit{Diagnostic Report} window is the most comprehensive of the set. It checks the integrity of the entire model according to five of the seven rules of OOEM as explained in detail in \textbf{Sections 3.6.1 to 3.6.5}\textsuperscript{10}. The algorithm is designed to generate ONLY an exception report and provides the following error messages as classified by the rule(s) it violates. Note that although error messages are classified according to the five rules, the nature of an error may sometimes make it non-exclusive to that rule.

1. \textbf{Rule #1: The Scope Identification Rule}
   
   - The Following Client(s) Made NO Requests To The System
   - The Following Object(s) Did NOT Receive Any Request For Service
   - The Following Request(s) Had NO Origin
   - The Following Request(s) Were NOT Sent To Any Object
   - The Following Request(s) Have No Interface Attribute
   - The Following Request(s) Did Not Invoke Any Service

2. \textbf{Rule #2: The Object Identification Rule}
   
   - The Following Client(s) Made NO Requests To The System
   - The Following Object(s) Did NOT Receive Any Request For Service

\textsuperscript{10} However, as explained in \textbf{Section 5.3.1.2.2: Usability Checklist, Item 9: Error Prevention}, the design of OOEMCT relies, where possible, on error prevention rather than on error identification e.g it only allows an Internal Attribute to be modified by its owner: Rule #5.
3. **Rule #3: The Service Inclusion Rule**
   - The Following Service(s) Have NO Owner
   - The Following Service(s) Were NEVER Invoked

4. **Rule #4: The Attribute Inclusion Rule**
   - The Following Interface Attribute(s) Were NEVER Invoked
   - The Following Internal Attribute(s) Were NEVER Accessed

5. **Rule #5: The Attribute Ownership Rule**
   - The Following Interface Attribute(s) Have NO Owner
   - The Following Internal Attribute(s) Have NO Owner

Besides reporting semantic inconsistencies in the model, the algorithm for generating the *Diagnostic Report* window also detects situations when a *Request* is sent to a *Provider Object*. If this is the case, it generates the message, “A Request Was Sent To A Provider Object. Please Check That Object Is Not Really An Internal Object”. If a model is Ontologically correct, the algorithm reports that, “The Model Is Ontologically Correct”.

Finally, the algorithm for generating the *Model Report* window traverses the model “tree” (see *Section 4.2 OOEM and OOEMCT* and *Figure 4*) searching for model details as outlined in pseudo-code below. The algorithm is presented in two parts: the Main Algorithm in *Figure 12* and the Recursive Algorithm in *Figure 13*. Note that these algorithms are identical in concept to those first published by Zhao, 1995 p 37-38 but adapted for implementation by OOEMCT.
example, the recursive algorithm in this thesis makes no distinction of object type because of varying depths of knowledge about \textit{External Objects} (see \textbf{Section 3.5 Scope of Knowledge and Influence of Objects}). The algorithms have also been greatly simplified for wider readership.

```
Start of Main Algorithm

While there are Clients:
  While Client has outstanding Requests:
    Do Propagation Trace (Recursive Algorithm)
  End While
End While

\textbf{Figure 12 Main Algorithm}
```

```
Start of Recursive Algorithm

For each Request
  Identify Target Object
  Identify Invoked Service
  Identify Interface Attribute
  Identify Internal Attribute
  Identify All Spawned Requests By Service
  For Spawned Request
    Recur Recursive Algorithm

\textbf{Figure 13 Recursive Algorithm}
```
5.3.2.2 Emplacing the Diagnostic Facility

The implementation of the diagnostic facility could have been achieved through one of four alternative approaches:

- The facility could be constantly engaged;
- The facility could be selectively turned on or off;
- The facility could be made to focus only on specific selections;
- The facility could be deferred and activated only on user request.

A decision was taken to adopt the last approach, that is, to defer activating the diagnostic facility until a specific user command was invoked.

This decision was made on the premise that engaging the diagnostic facility constantly may serve, instead, to distract the user during model construction. Obviously, an incomplete model would almost always be internally inconsistent. If the diagnostic facility were to be constantly engaged, then an unobtrusive way to keeping the user aware and informed of system states must be devised. Although this was appealing at first, it was decided to postpone the development of this facility until more is learned about users' reaction to it from the empirical study.

The other two alternative approaches were similarly deferred until more is understood about what users require (or desire) from this facility.
5.3.3 The Data Storage Sub-Component

It was decided early in the development process to build OOEMCT’s data store upon a relational database system. This approach was adopted because relational database packages for microcomputers are ubiquitous and the technology is thus widely understood and easily maintained even by relatively new users. Out of the popularity of relational database comes a rich set of data manipulation tools that makes viewing and manipulating stored information very much easier than with a proprietary data format. Finally, the popularity of Microsoft’s Open Database Connectivity standard (ODBC) and other similar database interface drivers are allowing many development tools to access and manipulate a great number of relational database products. For example, a modern Prolog development tool with ODBC capability may be used to build an inference engine and “plugged-in” to OOEMCT should one be required in the future.

The tables actually used in the implementation correspond closely to that of the metamodel in Figure 6. A definition of these tables is enclosed in the Appendix 10.

5.3.4 The Need For An Empirical Study

Having searched the literature in human-computer interaction design, the decision to proceed with an empirical study can best be summarized by Landauer (1995) who believes that:

“Where it is possible, the use of theory will be constrained and modest, because theories will be imprecise, will cover only limited aspects of behavior, will be applicable only to some parts of some systems, and will not necessarily generalize; as a result, they will yield little advantage over empirical methods.“ (p. 659)
Myers (1992) collaborates:

"It is notoriously difficult to design a user interface that will be easy to use, and there are no design strategies that will guarantee that the resulting user interface will be learnable, easy to use, and "user-friendly." (p. 2)

Finally, software engineers and programmers, who perform much of the actual system design work, rightly focus their attention and creative energies on the mechanism, on how to write the program efficiently and elegantly, and how to maximize performance and maintainability - not on usefulness or usability. It comes as no surprise then that Landauer (1995) considers personal observation of one's own system to be one of the worst ways to assess usability.

With these very thoughts in mind, we set out in Chapter 7 to explain, in detail, an empirical study carried out to test the usefulness and usability of OOEMCT. But first, the next chapter will deal with the functional specifications of OOEMCT.
Chapter 6

Functional Design

6.1 Introduction

This chapter will discuss the functional components of OOEMCT beginning with an overview of system functions from a hierarchical perspective. The discussion will then traverse the hierarchy to explain each functional component in detail including interface components and access routes.

6.2 Hierarchy of Functional Components

The functional components of OOEMCT can be divided into three principal operational modes: 

*Guided* mode, *Free Form* mode, and *Report Retrieval* mode.

In the *Guided* mode, as its name suggests, novice users receive an abundance of help, feedback, and user intervention-free system execution during model construction. Error-prevention is a high priority in this mode and that responsibility rests heavily on OOEMCT.

The *Free Form* mode, on the other hand, is intended for use by experienced users. Users are able to move freely between different activities during model construction. This affords greater flexibility and productivity by, for example, allowing related activities to be grouped together. Of course, error-prevention now becomes the responsibility of the user as the computer frees nearly all restrictions and controls in the model building process.

Finally, the *Report Retrieval* mode offers users a series of four reports about the stored model.
Information about stored Objects is available through the Objects Report window as is information about Requests through the Requests Report window. The Model Report window presents information about the inter-relationship of stored elements and finally, the Diagnostic Report window presents information about possible inconsistencies in the model. All report windows may be saved in ASCII file format.

A detailed explanation of all functional components follow below. But first, Figure 14 below presents a graphical depiction of the hierarchy of the functional subsystems in OOEMCT.
6.2.1 The Home Form

Borrowing from a metaphor used in web browsers, the Home form is the top level interface that greets the user when OOEMCT is invoked. The Home form is the gateway to one of three principal sub-system functions available in OOEMCT: Guided model entry, Free Form model entry and Reports retrieval. The Home form is presented in Figure 15 below.

One caveat is, however, necessary. Unless the user is extremely familiar with the tool, switching between the Guided and Free Form modes midway during the construction of a model is strongly discouraged. According to Norman (1983), he observed that “mental models do not

Figure 15 The Home Form
have firm boundaries: similar devices and operations get confused with one another.” He further warns that “Our conceptualization of a target system should not be confused with the mental model that a user creates of that system”. Without further study into the mental models developed by users of OOEMCT in each mode of usage, we cannot determine if switching between modes is advisable.

6.2.2 Guided Model Entry

When the Guided mode is chosen, the interface is as presented in Figure 16 below.

![Guided Model Entry](image)

**Figure 16** Guided Model Entry

In the Guided mode, users are led through a predefined sequence of forms that will enable them to input data in a structured step-wise manner. Naturally, the sequence of data entry is arranged according to the algorithm presented in Section 3.7.1 A Structured Approach. Explanation or help is given on each form so the user is always kept aware of the required action and its purpose.
Also, a viewer window (see Figure 16 above) keeps the user "mapped" to his or her progress in the model building process. An explanation of the contents of the viewer window follows.

In Figure 16 above, the layout of viewer window's contents follows the format below:

<table>
<thead>
<tr>
<th>Client Name</th>
<th>Request Name =&gt; [Object Name] &lt;Service Name&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spawned Request Name =&gt; [Object Name] &lt;Service Name&gt;</td>
</tr>
</tbody>
</table>

Notice from Figure 16 that each Request originating from a Client is displayed in a new block (or paragraph). For example, Client 1 made two Requests (Req 1 and Req 2) and they were placed on separate blocks (the first and second blocks in the viewer window). Notice also that whenever a Service spawns a Request, the new line generated is indented once from the one containing the spawning service. For example, in the first block, Sp Req 1 was spawned by Svc 1 of Int Obj 1 (directly above) and was hence indented once from the preceding line. Similarly, in the same block, Sp Req 2 was spawned by Svc 2 directly above and was indented twice because the line containing the spawning service had already been indented once before. Where a Service spawns multiple Requests, each Request so spawned is given the same "rank" and indented to the same degree. For example, in the third block, Sp Req 4 and Sp Req 5 are indented to the same degree because they were both spawned by Svc 6 of Int Obj 3.

The goal of providing model feedback through the viewer window was, of course, to give the user an easy, efficient, and hopefully error-free way of building OOEM models.
6.2.3 Free Form Model Entry

If, however, the user chooses to enter data via the *Free Form* mode he or she will be presented with the interface as in Figure 17 below.

![Free Form Entry Interface](image)

**Figure 17 Free Form Entry Interface**

The *Free Form* mode as its name implies, allows more experienced users to directly enter objects, *Services*, *Requests* (and *Responses*), and *Attributes* without thinking about how they may be associated in the model. This provides the user with a quicker, more flexible, and direct mode of model entry sacrificing much of the guided help structure available in the *Guided* mode.
Users in the Free Form mode may, of course, input information in any order.

Free Form mode users, in this version of the OOEMCT prototype, will not be able to obtain a viewer window much like the one available in the Guided mode. To view the model, users will have to leave the data input area and access the Reports mode of OOEMCT.

As can be seen from the Free Form mode interface in Figure 17 above, the Free Form mode itself is again divided into two further sub-functions: Direct Entry and Link Entries. An example of a Direct Entry form is available in Figure 18 below.

![Figure 18 Direct Entry of Objects Form](image-url)
When the user has completed entering *Objects, Requests, Services, and Attributes* into the system, s/he may then link them together to form the complete model. For example, *Clients* and *Requests* may be linked or associated by pushing the relevant button. **Figure 19** below shows a "Client and Request" Link Entries form. In **Figure 19**, *Client 1* may be linked to either (or both) *Requests* displayed in the scroll box (Req 1 and Req 2) simply by highlighting a *Request* and clicking on the "Link" button. If the user changes his/her mind, a *Request* may similarly be detached (unlinked) by following the above procedure except that the "UnLink" button should be depressed instead.

![Figure 19 Link Entries Form](image-url)

*Functional Design: 61*
6.2.4 Report Retrieval

Note that the *Home* form is also able to deliver, at any point, reports about the model in one of four report formats: *Objects*, *Requests*, *Diagnostic*, and *Model*. When invoked, the *Reports* button brings the user to the second level interface as depicted in Figure 20 below.

![Figure 20 Reports Interface](image-url)
When a user chooses an *Objects* report, s/he is presented with a report window as presented in Figure 21 below.

![Figure 21 The Object Report Window](image)

The *Objects Report* window contains an inventory of all objects in the model and includes information about their *Services*, *Interface Attributes*, and *Internal Attributes* where provided by the analyst.
Similarly, the *Requests Report* window presented in **Figure 22** below contains an inventory of all *Requests* in the model.

![Requests Report Window](image)

**Figure 22** The Requests Report Window
The *Diagnostic Report Window* presented in **Figure 23** below provides the user with a list of "bugs" in the model. Similar to a debug list generated at the end of compiling a program, the *Diagnostic Report* window displays a list of exceptions except that unlike a debug list, the errors flagged are not usually syntactic or logical errors but semantic errors in terms of the OOEM Rules presented in Chapter 3. The *Diagnostic Report* window is also different from a debug list because it provides a brief explanation of why the exception was flagged and therefore where the analyst might begin looking for the offending entry. An explanation of its contents presented in **Section 5.3.2.1.3 The Report Generation Processing Module.**

**Figure 23** The Diagnostic Report Window
The Model Report window is the last of the series of four report windows available in OOEMCT. Its content is identical to the viewer window available to users of the Guided Model Entry mode. Presented in Figure 24 below, the Model Report window displays the complete model details according to the format presented and explained in Section 6.2.2 Guided Model Entry above.

![Figure 24 The Model Report Window](image)

### 6.3 Conclusion

This chapter covered the various functions of OOEMCT and highlighted the features of each function. The next chapter will describe an empirical study to test the utility of OOEMCT. As
the section on defining usability will explain, utility and usability are inseparable. Major problems in usability will, no doubt, affect the utility of OOEMCT. In view of this, the study will also make note of how successful our interface design efforts have been.
Chapter 7

Empirical Study

7.1 Introduction

Whether the objectives of OOEMCT, and that of this thesis as a whole, were met cannot be successfully answered without an empirical study into its usability. Besides a sound theoretical and technical foundation, the development of any software tool that involves human interaction is immediately fraught with many other complex behavioral considerations. Therefore, besides testing OOEMCT against the broader objectives of this thesis, it is hoped that the usability study would also uncover user cognitive and behavioral issues that surface when a tool of this nature is encountered for the first time.

This chapter will proceed to discuss in detail the preparation and approach of the usability study for OOEMCT.

7.2 Definition of Usability

In order to study the usability of a software system, it is important to first understand how usability fits into the overall issue of system acceptability.

System acceptability is the question whether the system is good enough to satisfy all the needs and requirements of users and others, such as the users' clients and managers (Nielson, 1993). This acceptability is made up of both social acceptability and practical acceptability.
An example of a system which is practically acceptable, but may not be socially acceptable is one that successfully detects fraudulent applications for unemployment benefits. Such a system is not socially acceptable because it would entail the collection, storage, and close scrutiny of very private information about individuals. Practical acceptability is comprised of cost, support, reliability, etc., as well as “usefulness”. Usefulness is really the issue of whether the system may be used to achieve some desired goal, and it can be described in terms of two categories: utility and usability. Utility is the question of whether the functionality of the system can in principle do what is needed, and usability is the question of how well the users can actually use that functionality.

### 7.3 Objectives of the study

The objectives of this study are

1. To evaluate the utility of the OOEM CASE Tool. 
   The ability (functionality) of the tool in evaluating a candidate model’s semantic integrity according to the rules of Wand and Woo’s OOEM methodology will be tested.

2. To make specific recommendations for improvement of the CASE Tool usability. Because utility and usability are closely interrelated, observations and feedback on the CASE Tool’s usability will be noted for use in future iterations of the tool.

For a more in-depth discussion of the CASE Tool’s interface design criteria, please refer to **Chapter 5** where issues of design are available.
7.4 Methodology

This study is intended to be an exploratory investigation of utility-related issues of OOEMCT. It was not designed to obtain feedback that allow precise mathematical or statistical analysis. In fact, this approach is supported by Wixon, Holtzblatt, and Knox (1990) who take the position that the only basis for hope in the hermeneutic and contextual research positions lies not only in the rejection of theory but also in the rejection of systematic data collection and analysis.

This study then will focus its efforts on the qualitative identification of key utility-related problems that may serve as the basis for specific recommendations in future research and development work for OOEMCT.

7.5 Methodology Framework

The methodology framework comprises subject selection, study session design, preparation of subjects, task scenario, and data collection and analysis.

7.5.1 Subject Selection

A total of nine subjects were enlisted. The use of a rather small number of subjects in this study was unavoidable as the pool of graduate students who had attended the systems analysis course and had not yet graduated was very small. But the small sample size did not necessarily erode the effectiveness of the study. Past research had revealed that a pool of four to five subjects is adequate for identifying major usability issues11 (Egan et al., 1990; Good et al., 1984; Nielsen, 1986).

11 As already outlined earlier in this paper, usability and utility are closely interrelated facets of a software’s usefulness.

Empirical Study: 70
The process of selecting subjects was not randomized. The research design deliberately chose subjects who had been taught and had experience building OOEM models with only simple software drawing tools. The subjects were, in fact, students who had completed Dr. Woo's course on Systems Analysis and Design at the University of British Columbia.

This was a realistic expectation since the target user population for OOEMCT would be systems analysts trained in various modeling techniques possibly including that conceived by Drs. Wand and Woo (i.e. OOEM). It was felt that the subjects' previous experience with OOEM would play an important role in contributing to their observations regarding the usability and practicality of OOEMCT as a modeling tool. The exclusion of novices would also add a degree of stability and consistency in test results.

Because subjects were previously required to complete course assignments (while they were attending Dr. Woo's course) using software running under Microsoft's Windows environment, a degree of consistency in user interface and software modeling tool experience was also achieved through this method of selection.

It is believed that the selection criteria would strengthen the external validity of the findings from this study as the target user population for the OOEMCT should possess some of these qualities.
7.5.2 Study Session Design

The study began with a pilot study to test the test, so to speak. Feedback from the pilot study was, of course, used to correct the design and approach of the study proper.

The remaining eight subjects were randomly split into two groups. Four subjects in the first group were assigned work using the Free Form mode of OOEMCT while the remaining four were instructed to work in the Guided mode. Each subject worked independently.

Subjects were first given training through participation in a live demonstration with the investigator and then assigned a case study to solve using OOEMCT.

7.5.2.1 Use of Questionnaires and Logs

Two questionnaires and an interview were used in the study. The first questionnaire basically sought to gauge the background and experience of the subject since, for some, a year or more had passed since they completed the systems analysis course. They may have attended further courses in systems analysis or were, in some other way, exposed to newer or better tools. The second questionnaire was a series or Likert Scale questions used to register the subjects’ immediate reaction to the tool in terms of its utility and value.

The interview was conducted just before the end of the session and its main purpose was to elicit qualitative responses about the tool’s utility in general and its interface design in particular. It was also an opportunity for subjects and the investigator to clarify any issues they may have about the study in general.
In addition, the investigator also kept a log of questions and feedback that spontaneously arose either during the demonstration or during the study itself.

### 7.5.2.2 Session Structure

In order to ensure that each test session was replicated and conducted with as high a degree of consistency as possible, two sets of documentation were used. All documents described in this section are attached in Appendices 1 to 9.

The first is an investigator’s package which included an inventory of the equipment and materials required for each test and a checklist outlining the steps required in preparing for the test and the sequence of action while conducting the test session.

The second is a subject’s package which included an explanation of the test and its objectives. A consent form, background questionnaire, demonstration case study sheet, and experiment case study sheet were also assembled in the package (but distributed only at the appropriate time).

### 7.5.2.3 Role of the Investigator

The role of the investigator was to conduct the test and collect as much feedback as possible about OOEMCT’s ability to meet the subjects’ needs and expectations.

The investigator did not assist the subjects in interpreting the case study while the study was in session. The subjects were encouraged to think and work independently. No discussion or persuasion that may introduce bias into the study was engaged during the entire test. Every
attempt to preserve the subjects’ independent operation and perception of the tool was made.

However, the investigator did interject whenever the user attempted an operation that would halt the system or in some other way significantly affect the progress of the study. Where a user was confused or “lost” remarks as outlined in the Investigator’s Remarks (see Appendix 6) were used to resolve problems in a multi-level structure that seeks to match the help level given with the gravity of the problem.

7.5.3 Preparation of Subjects

The subjects were first trained in the use of the tool by individually participating in a brief demonstration (about ten to fifteen minutes) with the investigator using a very simple case study. Subjects were taught only either the Free Form mode or Guided mode as relevant to them. They were also allowed to ask questions about the tool during the demonstration and the study proper. No attempt was made to obscure any aspect of the tool during the demonstration.

7.5.4 Task Scenario

Subjects, immediately after viewing the demonstration, were presented a totally unfamiliar case study developed especially for this study. To ensure that all subjects had the same goals in mind, the case study used was identical in every case (see Appendix 5). Subjects were instructed to use OOEMCT as they sought to model the problem domain. No time limit was set and it was not required that subjects produced a “working model” at the end of the session. When the study was over, subjects were requested not to discuss the proceedings and contents with anyone until the results of the study were published.
7.5.5 Data Collection and Analysis

As already discussed in Section 7.4 Methodology, the emphasis on collecting qualitative information overrides that of quantitative precision in a study of this nature. As such, the study used four different data collection techniques in order to obtain a more comprehensive and collaborative set of results. The techniques used were questionnaire, direct observation, video taping and interview. The two questionnaires used (Questionnaires 1 and 2) and the Post Study Interview Record are attached in the Appendices 7, 8, and 9 respectively. Video taping was used primarily as a cross-reference check and memory aid hence no transcriptions were made for this study.

A summary of the data collected from the study, including notes collected from direct observation, will be presented in the next chapter.
Chapter 8

Discussion of Results

8.1 Introduction

Although this study is essentially empirical research, it did not set out to produce definitive statistical support for the findings for reasons already mentioned in the previous chapter. Rather, the intention is for the results to lay the groundwork for future research in OOEMCT.

8.2 Overview

A total of nine subjects took part in the study: one pilot and eight actual study sessions. The purpose of the pilot session was to highlight as many potential problems with the study design as possible, prior to the running the actual test sessions. As there were no major changes made to the research methodology, findings from the pilot study have been included in the analysis of results.

8.3 Subject Background Questionnaire

Five of the nine subjects had between two to three years of experience with operating systems that used windows. Three others had about five years of similar experience while the last had about ten years of exposure to windowed operating systems.

Of the nine subjects seven had little\textsuperscript{12} or no working experience while one had two years of

\textsuperscript{12} Less than a year of experience.
working experience in the role of systems analyst. All eight had little opportunity to work with CASE tools other than those taught in the systems analysis course at The University of British Columbia. The ninth subject had over 16 years of working experience in system analysis.

All nine subjects expressed confidence in their ability to work with computers and all but one expressed the opinion that a computerized CASE tool would be more helpful than working with just “pen and paper” when building a model.

8.4 Study Feedback Questionnaire

Only eight subjects out of nine completed the Study Feedback questionnaire (Appendix 8). Of the eight, one completed the questionnaire only partially. The ninth did not complete the questionnaire as the study for that subject was abandoned midway. The subject had elected to stop participating in the study quite early.

8.4.1 Question 1

Question 1 asked subjects if they felt OOEMCT had made it generally easy for them to build a model. The results are presented in Table 1 below.

As the results show, five subjects thought it was. At the same time, it is not difficult to understand why two subjects who used the Free Form mode thought it was not. The Free Form mode offered minimal assistance and novice users found it daunting at times.

\[13\] Self-reporting was used.
Table 1 Question 1: OOEMCT Made It General Easy to Build Model?14

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>■■</td>
<td>■■■</td>
<td>■■■■</td>
</tr>
<tr>
<td>Neutral</td>
<td>■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Disagree</td>
<td>■■</td>
<td>■</td>
<td>■</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

8.4.2 Question 2

Question 2 asked subjects if OOEMCT had been of help to them in isolating and identifying objects, requests, attributes, and services as they entered them into the system. The results are presented in Table 2 below.

From the results, it would seem OOEMCT had been especially useful when it came to identifying objects, requests, and to a lesser degree, services. However, two subjects found isolating and identifying attributes to be especially troublesome. When probed, one of the subjects revealed that she had forgotten what interface and internal attributes actually were and the system offered no help in explaining these to her in detail. Finally, one of the subjects felt that OOEMCT had not been of any help as he had hand drawn the model on paper before beginning to use the tool. The fact that several other subjects had also hand drawn diagrams revealed a serious gap in OOEMCT’s ability to accommodate the “authoring” process that occurs very early in the development of a model. Section 8.6.3: Cognitive Issues: Idea Structuring elaborates on this finding below.

---
14 This is a truncated question for easy reference. The actual question is in the text.

Discussion of Results: 78
Table 2 Question 2: OOEMCT Helped in Isolating and Identifying Constructs?

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Objects</td>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requests</td>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attributes</td>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Services</td>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8.4.3 Question 3

Question 3 asked subjects if OOEMCT had been of help to them in keeping track of (remembering) objects, requests, attributes, and services they entered into the system. The results are presented in Table 3 below.

The responses were basically the same as that of the previous question with one interesting exception. Two subjects using the Free Form mode did not feel the tool was of help. It is not
immediately apparent why they responded as they did. One hypothesis is the fact that the *Free Form* mode did not offer a model viewer side-by-side with the form in use (users could, however, leave the form and evoke the model viewer separately). This hypothesis is interesting since users were always able to view all items they had entered when they were working in the *Direct Entry* forms (see Figure 18).

Table 3 Question 3: OOEMCT Helped in Keeping Track of Constructs?

<table>
<thead>
<tr>
<th>Constructs</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Agree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Objects</td>
<td>Neutral</td>
<td>⬤</td>
<td>⬤</td>
<td>⬤</td>
</tr>
<tr>
<td></td>
<td>Disagree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Requests</td>
<td>Neutral</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td></td>
<td>Disagree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Attributes</td>
<td>Neutral</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td></td>
<td>Disagree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Agree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Services</td>
<td>Neutral</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td></td>
<td>Disagree</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤</td>
<td>⬤⬤⬤⬤⬤</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

Discussion of Results: 80
8.4.4 Question 4

Question 4 asked subjects if OOEMCT had helped them remember where they were in the process and what they had to do next. The results are presented in Table 4 below.

Again the results seem to indicate that the Guided mode was preferred probably because of the additional assistance offered in that mode.

Table 4 Question 4: OOEMCT Helped You Remember Where You Were?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td>••••</td>
<td>••••</td>
</tr>
<tr>
<td>Neutral</td>
<td>•</td>
<td>•</td>
<td>•</td>
</tr>
<tr>
<td>Disagree</td>
<td>••••</td>
<td>••</td>
<td>••</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

8.4.5 Question 5

Question 5 asked subjects if they found the Model Viewer window helpful. The results are presented in Table 5 below.

Users in the Free Form mode were also able to access the Model Viewer window but had to leave the Direct Entry form in order to do so. Perhaps their “sitting on the fence” response reflect their positive feelings about having the window but their disappointment at not being able to constantly access that window. Originally, it was thought that putting the model viewer outside the Direct Entry forms would reduce the possible confusion users might experience because the model viewer could only display model details when the request propagation “sequence” was in

Discussion of Results: 81
place. In other words, the model viewer could only display the tree (described in Figure 4) if it was complete from the “root” up.

**Table 5 Question 5: Was The Model Viewer Window Useful?**

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td>☐☐☐☐</td>
<td>☐☐☐☐</td>
</tr>
<tr>
<td>Neutral</td>
<td>☐☐☐☐</td>
<td></td>
<td>☐☐☐☐</td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

### 8.4.6 Question 6

Question 6 asked subjects if OOEMCT had made it easy for them to keep track of the links between objects, requests, attributes, and services in the system. Detailed results are presented in Table 6a and consolidated results in Table 6b below.

The results show that the number of subjects who agreed and those who disagreed were not very different for the Free Form group. In the Guided group, however, those who agreed outnumbered those who disagreed two to one. One explanation could be the Model Viewer window that was available only in the Guided mode.
Table 6a Question 6: OOEMCT Made It Easy To Keep Track of Links?

<table>
<thead>
<tr>
<th>Links to Requests</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objects to Attributes</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Objects to Services</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requests to Attributes</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requests to Services</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Attributes to Services</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>[ ]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

Discussion of Results: 83
### Table 6b Consolidated Results from Question 6

<table>
<thead>
<tr>
<th>Links</th>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Links</td>
<td>Agree</td>
<td>8</td>
<td>16</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>Neutral</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Disagree</td>
<td>9</td>
<td>8</td>
<td>17</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td></td>
<td><strong>42</strong></td>
</tr>
</tbody>
</table>

### 8.4.7 Question 7

Question 7 asked subjects if OOEMCT made it easy for them to trace a request from its origin to the end. The results are presented in Table 7 below.

The results again show about an even distribution between those who agreed and those who did not. However, the interesting thing to note here is that the polarized opinions are split neatly down the middle between those who used the Free Form mode and those who used the Guided mode.

### Table 7 Question 7: OOEMCT Made It Easy To Trace A Request?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>●●●</td>
<td>●●●</td>
<td>●●●</td>
</tr>
<tr>
<td>Neutral</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Disagree</td>
<td>●●●</td>
<td>●●●</td>
<td>●●●</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>
8.4.8 Question 8

Question 8 asked subjects if OOEMCT’s ability to check a model’s integrity helped them make fewer mistakes. The results are presented in Table 8 below.

Although only about half of the subjects were clearly in favor of the integrity check, it still seems encouraging to keep enhancing and testing this feature because only one subject was clearly against the idea. From the post test interview, two subjects suggested positive enhancements for the diagnostic utility and their dissatisfaction with the current implementation must have accounted for their reluctance to fully accept the diagnostic utility’s value as it stood.

Table 8 Question 8: OOEMCT Checking Resulted In Fewer Mistakes?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>■■</td>
<td>■■</td>
<td>■■■■</td>
</tr>
<tr>
<td>Neutral</td>
<td>■</td>
<td>■■</td>
<td>■■■</td>
</tr>
<tr>
<td>Disagree</td>
<td>■</td>
<td></td>
<td>■</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

8.4.9 Question 9

Question 9 asked subjects if they thought OOEMCT could potentially help them build models of greater size and complexity than they would with just “pen and paper”. The results are presented in Table 9 below.

A clear majority of the subjects thought the tool would clearly help them when models of greater size and complexity were encountered. Note that two subjects from the Free Form group clearly

Discussion of Results: 85
disagreed with the idea. The reason is for their negative response is not difficult to guess.

**Table 9** Question 9: OOEMCT Useful In Building Bigger And More Complex Models?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

**8.4.10 Question 10**

Question 10 asked subjects if they thought OOEMCT could potentially help them build models quicker than they would with just “pen and paper”. The results are presented in Table 10 below.

The distribution of responses seem to indicate that the subjects were not sure if this were the case. This is perhaps not surprising since there were no time limits enforced and subjects were probably oblivious to the time it took them to complete the exercise.

**Table 10** Question 10: OOEMCT Could Help Build Models Quicker?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>
8.4.11 Question 11

Question 11 asked subjects if they thought OOEMCT made it more enjoyable to build models.

The results are presented in Table 11 below.

Quite like Question 9, a majority of the subjects found the tool enjoyable to use with the notable exception of two subjects from the Free Form group.

Table 11 Question 11: OOEMCT Made It More Enjoyable?

<table>
<thead>
<tr>
<th>Response</th>
<th>Free Form</th>
<th>Guided</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agree</td>
<td>••</td>
<td>••••</td>
<td>••••••</td>
</tr>
<tr>
<td>Neutral</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disagree</td>
<td>••</td>
<td>••</td>
<td>••</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td></td>
<td>8</td>
</tr>
</tbody>
</table>

8.5 Summary for Study Feedback Questionnaire

The Free Form mode users were generally less positive about OOEMCT than the Guided mode users. This is not really surprising since the Free Form mode offered less help with both textual and visual information then the Guided mode. This in turn is also understandable since, generally, many of the subjects had became "rusty" with the OOEM methodology and had very little working experience as systems analysts. In effect, this meant they had to grapple with essentially three learning tasks at once; systems analysis in general, OOEM methodology, and the tool.
8.6 Post Study Interview

Feedback from the post study interviews was divided into four categories. The first category (Design Issues) will deal with comments about tool that do not fundamentally affect cognitive processing activity. The second category (Cognitive Issues) will, of course, deal with comments that directly address a need to improve the task interface as outlined earlier in Section 5.3.1.3: Designing for Cognitive Advantage. The third category (Utility Issues) will deal with feedback about specific features of the tool. The last category (Miscellaneous Issues) will be the catchall category for feedback that did not fall into any of the three preceding categories. Each category is again sub-divided internally into broadly identifiable “themes” of interest. It must, however, be emphasized that the criteria used in assigning items of discussion to each of the above categories is arbitrary and meet only the needs of this presentation. In no way was the user study equipped to identify and isolate the critical thinking processes that were involved at each stage of using the tool. We now proceed to the first category.

8.6.1 Design Issues - Usability Checklist

Feedback from subjects indicate that the usability checklist (see Section 5.3.1.2.2: Usability Checklist) had been very useful in the design of the graphical user interface (GUI). This section first deals with the constructive comments and then the “reinforcement” comments. Please note that the non-committal tone used in this section is not to be construed as one conveying a sense of irresolution. Rather, it underscores the need for further rigorous testing with larger groups of subjects before a firm commitment to design changes can be endorsed. The constructive comments received about the general design of the GUI were:

- More detailed help (even an online tutorial) would have been helpful in the Free
In the Free Form mode, the direct entry form for objects should indicate the object type against the object name in the list box. In the current implementation, users have to click on the desired object in the list box and observe the object type from the radio buttons immediately above the list box (see Figure 18).

- In the viewer window (Guided mode), interface and internal attributes should be displayed where available.

- When pursuing a depth-first traversal during model construction, there was no quick way to resume to the point of last (or desired) branching to pursue another thread of request propagation (see also Section 8.6.2: Cognitive Issues: Navigational).

- The forms and the text used in the tool might have been too small\textsuperscript{15}.

- The scroll buttons (see Figure 25) were not labeled and were sometimes confused for the “< Back” or “Next>” navigation buttons. Also, the relevant button should be disabled if there were no more items on the list in either “direction”.

- The way an item “drops” from the combo box into the list box when “added” might be confusing to some users (see Figure 25 below). Perhaps the selected item should remain displayed in the combo box even after being added.

\textsuperscript{15} This may be corrected by lowering the resolution of the screen if desired.

\textit{Discussion of Results: 89}
The following positive comments were also received from subjects:

- The elements of the form were described as being "very friendly", "easy to understand", and "easy to use".
- Keyboard shortcuts were appreciated.
- Messages in dialog boxes were short and non-technical in content.
- The combo box, when "dropped", very helpfully displayed a complete inventory of items relevant for that category. This allowed the user to check if a particular

Discussion of Results: 90
8.6.2 Cognitive Issues: Navigational

This section deals with feedback that concerns aspects of the cognitive interface relating to movement and positioning within the global context of the model.

Subjects unanimously indicated that a graphical interface for the tool would have been useful with one subject noting that it would have been especially useful at the "fine-tuning" stage. One of the reasons\textsuperscript{16} cited for this was that it would have made navigation easier. The viewer window, which presented a global context of the model, was not originally intended as a navigational aid was thus abandoned by most users with one indicating that it was of absolutely no use.

As such, navigation is a major issue in the design of OOEMCT. Detailed feedback from subjects relating to the subject of navigation is presented below.

- Splitting the entry of objects, services, interface attributes and internal attributes into four separate forms was confusing. One has to navigate through three or four forms to obtain a complete picture of the object.

- In the \textit{Free Form} mode, linking requests to objects, services, and interface attributes involved navigating through too many forms and thus became confusing.

- Putting the service form immediately after the internal object form was unnatural.

\textsuperscript{16} A second reason for wanting a graphical interface is presented in Section 8.6.3: Cognitive Issues: Idea Structuring.

Discussion of Results: 91
The interface attribute form should come immediately after the internal object form instead.

- When pursuing a depth-first traversal during model construction, there was no visual or mental "anchor" that would help in mapping the current level one was at in the "tree" (against other branches of pending requests). Also, having to manually "back-out" of a depth-first search to the last (or any "branch") was tedious and counter-intuitive.

- Navigation through direct interaction with the viewer window would have been desirable.

### 8.6.3 Cognitive Issues: Idea Structuring

Without fail, every subject had either implicitly or explicitly indicated that a graphical interface for the tool would have been very desirable. What is even more intriguing is that about half the subjects drew diagrammatic representations of their model on paper before beginning to work on the computer. This raises three interesting hypotheses:

- The state of the art in modern windows-based software tools has raised the expectations of users significantly. This is true even for prototypes.

- The way subjects were taught to build models in class was through diagrammatic expression.

- The desire for a graphical interface may not be entirely based on visual appeal or ease of manipulation alone.

Our contention is that all three hypotheses are true. However, we believe that in our case, the last
hypothesis is of special interest and significance to our learning since it has direct bearing on the design of OOEMCT’s GUI.

Recall that in Section 5.3.1.3: Designing for Cognitive Advantage, we had to defer investigation into the cognitive tasks and processes that are involved when using OOEMCT. We now have a better insight into what these cognitive tasks and processes might be; especially during the initial stages of model construction.

We believe that the clamor for a graphical interface can be attributed to the subjects’ need for a “canvas” upon which to sketch ideas. But it does not seem to stop here. Subjects also felt that the tool was “too rigid” in that it forced them to declare the object type\(^{17}\) at the outset, start only with a client, and would not let them change object type midstream. This presents a serious hurdle to the idea structuring and restructuring process that is constantly going on until the model is completed. It is wholly conceivable that an analyst may discover the “true” Client of a system only towards the end of the modeling process when everything becomes clearer.

A number of subjects also observed that the “tree” structure of request propagation would make model revision or tuning extremely difficult as a single change could upset the intricate network of propagated requests downstream.

Relating these ideas, we realized that subjects were basically articulating the need for a more effective task interface (see Section 5.3.1.3: Designing for Cognitive Advantage). This translates

\(^{17}\) And presumably attribute type.
into the need for OOEMCT to serve in several roles at various stages of the model development process. In the initial stages, the tool would be expected to help users gather information, extract and discover ideas, structure them, and finally compose them into a comprehensible model - a process known as idea structuring (Card and Moran, 1995). Next, OOEMCT will be expected to evaluate the model and that in turn sparks off further rounds of iterative idea structuring and restructuring until the model is internally consistent or until the user chooses to halt the process in the interest of time or other constraints.

The iterative pursuit of a representative model and the ensuing changes it entails cannot but support the fact that objects and their attributes in OOEMCT must be permitted to change type midstream and change type repeatedly. This is natural since the scope of a model constantly shifts with a better understanding and appreciation of the problem domain. This conclusion is consistent with Green’s (1989) cognitive dimension framework where one dimension expresses the notion of premature commitment where the user is forced to make choices too soon (see Section 9.2.2: Object Typing and Request Origination).

Related to this is the fact that the tool must also allow users to start building a model from any point (or branch) up. That is, it should not be mandatory or even necessary to start building a model from a Client up. Again, this is natural since the scope of a model may change such that a previously identified Client might no longer be. Finally, a way of simplifying the editing of an intricate network of propagated requests must be found.

As a final note, one subject felt that the tool should not require users to specify object types at all.
but instead make recommendations to the user as to the role of each candidate object in the model might be.

8.6.4 Cognitive Issues: Shift in Request Generation

According to OOEM methodology, clients despatch requests into the system. Thereafter, it is the services invoked that spawn propagating requests throughout the system (see Section 3.7.1: A Structured Approach). This shift in the request "originator" role seemed to confound a number of subjects on two levels.

First, subjects felt that objects should generate (or spawn) requests, not services. The idea of services, basically a "non-entity", generating requests did not quite sit well with a few respondents - especially the novices. It took some explanation to convince the subjects. But that in turn raised a second question.

If services within objects are responsible for generating requests, how is it possible for clients to generate requests through phantom (unknown) services? This, in itself, raises another two issues. First, it would take additional cognitive effort during the idea structuring stage (see Section 8.6.3: Cognitive Issues: Idea Structuring) to differentiate and deal with the two constructs differently. Second, it hinders the fluid reassignment of object type since a service must be "invented" and inserted into a client in order to change its type. Besides being a hindrance during idea structuring this also poses a challenge from a software engineering (technical) perspective.

Discussion of Results: 95
8.6.5 Cognitive Issues: Memory Aid

Two subjects in the *Free Form* mode found it difficult to remember the items they had entered into the model; especially when it was time to link them together. One of the subjects suggested that multiple forms be left open so users could cross-reference item types against one another whether one was entering items or attempting to link them.

8.6.6 Utility Issues: Diagnostic Utility

Every subject felt that the diagnostic utility was a positive feature in OOEMCT - adding confidence to the finished model. However, they had the following comments to add:

- The utility should be constantly engaged and used to "drive" the modeling process by immediately identifying violations and requesting "appropriate" actions.
- The utility should be constantly engaged because, if uninterrupted, one assumes everything is fine until the end when it then becomes immensely difficult to make changes to the model.
- The diagnostic report at the end resembles an "error listing" in a program compilation. One is not sure where to begin in the debugging process and often some of the errors are in fact cascading errors resulting from a defect upstream. A method of organizing and prioritizing the violations or guidelines to help resolve the violations in an organized way would have been useful.
- The utility would probably be more useful in larger models.
- Some aspects of the diagnostic report would be obsolete if a graphical interface were available e.g. a missing request between two objects would be obvious even without the diagnostic.
The utility should provide more information. It was probably not enough to just highlight the area(s) of model weaknesses. One suggestion was for an expert system-like explanation facility. Another suggested that the utility goes as far as to recommend specific actions to take.

8.6.7 Utility Issues : Guidance Mode

Subjects - particularly novices - enjoyed working with the hand-holding provided by the tool in the Guided mode. One subject, a novice, expressed much satisfaction with the structured help the tool offered while building the model.

8.6.8 Miscellaneous Issues

This section will discuss minor issues of interest from the user study.

- When “pressed” by the stricture of the Guided mode subjects may “invent” services or interface attributes to satisfy the system without appearing to be totally committed to the need from the facts of the case.

- The fact that OOEMCT allows users to trace requests to specific services was, at least in one instance, appreciated and held as being "informative".

- In one instance, a subject using the Free Form mode found it easier to first identify a request and then link it with the spawning service. This was in direct contradiction with the design approach and hence the interface controls.

- In instances where the diagnostic utility “approved” a model, subjects tended to place much confidence in the evaluation. None questioned the report further or even examined the model again!
8.7 Summary

This chapter has dealt at length with the findings from the empirical study. The study revealed four important facts about OOEMCT. They are:

- Users use the interface to sketch their ideas;
- Users did NOT want to commit to object types prematurely;
- Users expected to see object-to-object communication (not request-to-service connection);
- Users wanted to receive feedback from the diagnostic utility but in an unintrusive way.

The next chapter will deal with whether the findings support or refute the goals and objectives of this thesis.
Chapter 9

Research Questions and Future Research

9.1 Research Questions

The intention of this thesis was to answer the research questions expressed earlier in the first chapter. These questions are reproduced below and an attempt is made to answer them using the findings from the empirical study for support where applicable.

1. Can the rules of OOEM remain ontologically expressive and yet be sufficiently formalized to be captured within a computerized tool?

Yes. OOEMCT is able to capture a large part, if not all, of the semantics embodied in the first five of the seven rules of OOEM. The Guided mode, for example, allowed novice users to build OOEM models of at least good acceptability even though they had not used the methodology for a while.

2. In applying the rules, can the tool effectively detect situations where candidate models have not adhered to the rules of OOEM?

Yes. OOEMCT is able to process stored models and provide an exception report that points directly to where a semantic rule had been broken. One subject deliberately used the Diagnostic Report window to repeatedly amend his model until it was entirely correct. He did not have difficulty understanding the diagnostic report.
3. If OOEMCT is indeed effective in storing and applying the rules of OOEM, will users find such a tool (as implemented) useful?

Yes. Questions 8 to 11 discussed in Chapter 8 seem to indicate a positive response to OOEMCT especially from those who used the Guided mode.

4. Finally, what other key questions will this research raise?

Chapter 8 is filled with questions that are interesting and valuable to the future research and development of OOEMCT. The next section highlights three key questions that urgently need answering in the next iteration of OOEMCT.

9.2 Future Research

This section is divided into three sub-sections each dealing with a specific area that requires further research and development.

9.2.1 Graphical User Interface

From Chapter 8, we learn that users found the absence of a graphical user interface (GUI) to be the major stumbling block. Yet in designing a GUI, two major hurdles must be overcome.

- How should the GUI accommodate frequent “pruning and grafting” of the model “tree” during idea structuring and restructuring?
- How should the GUI allow a user to quickly appreciate and organize the propagation structure of the model? In other words, how would the GUI
effectively communicate the rather intricate web of propagated requests in the model?

In this investigator’s opinion, future research into the design of GUI should consider a non-random or “free-space” placement of symbols (that represent the constructs of OOEM) on the canvas. This is the case with most other CASE tools on the market. In Figure 26 below, a proposed graphical visualization of the communication model is presented from a previously completed case study. The reader should not be over concerned with the facts of the case study (or lack thereof!).
Figure 26 Proposed Graphical User Interface for OOEMCT
What is important from studying this model is the concept of X-Y axes that help users appreciate the sequential propagation of requests (X axis) and order of processes (Y-axis). This investigator believes that a “controlled” layout such as this one, will not only communicate temporal and spatial characteristics about requests and processes quickly, it will also make structuring and restructuring the model much easier. In other words, pruning and grafting trees can be greatly simplified if a sub-tree could be easily identified, isolated, and pruned. In a layout such as the one in Figure 26, sub-trees can be easily pruned and grafted to new locations with much less effort both cognitively and operationally. This layout is very similar to manipulating files and folders in a graphical file system browser like Windows 95's Explorer. Just as folders, and the sub-folders contained therein, can be moved by simply dragging and dropping the top-most folder to a different “branch”, so too can objects and their propagated requests be moved as such.

9.2.2 Object Typing and Request Origination

During the idea structuring stage, users indicated that they did not wish to commit to an object type immediately preferring to defer the decision until they were ready. Although the proposed layout discussed above might allow this, one serious hurdle visible in the OOEM metamodel prevents this.

Recall that users felt uneasy that clients generated requests initially and then this transformed into services spawning requests after the triggering request from the client “enters” the domain of interest in the model. This makes object typing impossible to change midstream as one should recall that clients offer no information about their internal properties. In other words, one would have to invent a dummy service for a client if one wishes to later recast the client as an internal
In the metamodel, the relationships *Generate* and *Send* might need to be removed leaving only the relationship *Spawn* to handle all manner of request origination.

### 9.2.3 Diagnostic Feedback

Recall from *Section 5.3.2.2: Emplacing the Diagnostic Facility* that there were four different strategies to engage the diagnostic facility. In the empirical study users had indicated that they would like to be made aware of their “mistakes” or “loose ends” they must take care of before it was too late. On the other hand, it is not unreasonable to assume that if the facility were overly intrusive, users would be distracted. As such a happy median must be found.

### 9.3 Conclusion

The objective of this thesis has been met. As far as theory is concerned, OOEM lacked a second external object type (*Provider*) to cope with modeling domain diversity.

Usability wise, the next iteration of the OOEMCT prototype must ensure that:

- Users can use the interface to sketch their ideas;
- Users need NOT commit to object types prematurely;
- Users see object-to-object communication (not request-to-service connection);
- Users receive feedback from the diagnostic utility in an unintrusive way.

Together with the other issues raised in *Chapter 8*, there is fertile ground for much future
research in OOEMCT. From feedback collected in the empirical study and personal experience there remains much optimism in this investigator that OOEMCT in its finished form will be well received in the modeling community.
Bibliography


Appendices
Session Introduction
OOEM CASE Tool Study
Fall 1996

This study consists of the following steps:

1. You will be asked to sign the Consent Form.

2. You will be asked to complete Questionnaire #1.

3. You will be shown a demonstration of the CASE Tool using a simple example. You may ask questions or make comments during the demonstration.

4. You will be then be given a one-page case study from which you will attempt to build a model using the OOEM CASE Tool. There is no time limit to work through the case study.

5. I will sit through the session with you and you may ask questions or make comments about the CASE Tool. You may also seek clarification about information found in the case study but I will not be able to help you interpret information from the case study e.g. I cannot say if a certain entity is an object or not.

6. After you have finished the case study, you will be asked to complete Questionnaire #2.

7. I will then finish off the session with a short interview and you may ask me any question you have about the study during that time.
Appendix 2

How to Run a Session
OOEM CASE Tool Study
Fall 1996

Before The Day Itself:

1. Book the computer laboratory and the microcomputer.
2. Call subjects to confirm their participation.

On The Day Itself:

3. Make sure the consent form, all the questionnaires and case study notes are ready.
4. Bring up the program and make sure the program is working properly.
5. Make sure the room is ready i.e. the chairs are ready and there are no distractions.
6. Set up the video camera and test.

Subjects Arrive:

7. Give them the Session Introduction Sheet and bring them to a writing table.
8. Have them sign the Consent Form and complete Questionnaire #1.
9. When they finish, bring them to the computer and give them the demonstration case study notes and run the demonstration when ready.
10. Start the video camera before running the tutorial.
11. Give them the case study notes for the actual study (The Treasury Case) and let them begin the study when they are ready. Make sure the Investigator’s Remark form is ready.
12. When they have finished, move them away from the computer and give them Questionnaire #1.
13. Interview the subject using the Post Study Interview Form as a guide.
14. Session complete, subject may leave. Switch off the video camera.
How to Run a Session
OOEM CASE Tool Study
Fall 1996

After Subject Leaves:

15. Collect all documentation and make sure the subject's name appear on each one.

16. Collate the interview feedback and categorize the responses.
Appendix 3

Inventory List
OOEM CASE Tool Study
Fall 1996

1. How To Run A Session
2. Session Introduction
3. Consent Form
4. Case Study : The Treasury Case (with writing material)
5. Investigator’s Remarks
6. Questionnaire #1 : Background Information
7. Questionnaire #2 : Study Feedback
8. Questionnaire #3 : Post-Study Interview Questionnaire
9. Video Camera, Microphone, and Video Tape
Thank you for taking time off to participate in this study.

We are seeking your help in the evaluation of the Object-Oriented Enterprise Modeling Computer-Aided Software Engineering Tool (OOEMCT) developed at the Faculty of Commerce and Business Administration. OOEMCT, like other Computer-Aided Software Engineering (CASE) tools, is aimed at helping analysts build models of the domain under study so they may better understand and thence refine or resolve problems that may exist.

Your contribution and feedback will form the basis of the evaluation. As well, video taping may be done during the study or parts of it. This will facilitate the collection of important information so we may later review it. The video will NOT be used for other purposes without your prior consent in writing. The result of the evaluation may play an important part in subsequent changes or improvements to the system.

Please keep in mind that this study is to assess the system and NEVER you. In fact, reporting of the results of this study will be aggregated and your responses CANNOT be traced back to you or any other participant. So we would appreciate your frank opinion.

Once the session is over, you are requested to NOT discuss the study with your friends and classmates before December 15, 1996.

Participation in this study is completely voluntary and you may discontinue your participation at any time during the session.

If you agree to the above terms and conditions, please sign your name below.

_____________________________  ______________________  ________________
Name                                    Signature           Date

Thank you for your participation!
Appendix 5

Case Study
OOEM CASE Tool Study
Fall 1996

The following is a (fictitious) account of the treasury activities at a local bank.

Every morning, the treasury manager receives statements of requirements from various departments of the bank (e.g. domestic and international banking divisions) outlining their requirements for various types of currency to meet anticipated demand from their clients and/or the bank itself (e.g. in international bill settlement or fund transfers).

The treasury manager then consolidates the requirements from the various departments and works out the treasury's position. Next he advises the dealers, who each specialize in a particular currency, about how much of a particular foreign currency the bank will need and when the funds are needed to cover requirements. The dealers will then update their individual activity books based on the information and take positions (trade) accordingly.

Dealers in the treasury department are charged with buying and selling (excess) foreign currency to meet the needs of the bank. Deals are mostly "spot" or "futures" and dealers try to hedge against fluctuation risks in order to obtain the best possible rate of exchange for the bank. Each dealer maintains an activity book and all deals done are recorded on transaction records. A transaction record contains all the necessary details about a deal.

Treasury clerks collect the transaction records from dealers as and when deals are done. The clerks perform arithmetic checks on the figures as well as other validity checks such as the date, type of deal, and broking house used. If there is doubt, the clerks will refer to the dealers for clarification.

If the transaction records appear to be in order, the clerks will record the deals in the appropriate deal book e.g. US dollar deals, Italian Lira deals, etc. At the end of each trading week, the deal books are handed over to the treasury manager who checks and verifies the transactions and signs the closure of the books for the week. From the books, the treasury manager also prepares a weekly report of treasury activities for the vice president of the bank.

The transaction records are then handed to the accounting department who files them. At the end of each month, the accounts department prepares a balance sheet for each dealer outlining their financial position. Dealers are required to show a "healthy" balance sheet at each quarter.
Appendix 6

Investigator Remarks
OOEM CASE Tool Study
Fall 1996

Level 0: Minimal Information/Help
1. What are you trying to do?
2. Why did you just do that?
3. What do you think you can do next?
4. What are you planning to do next?
5. What is your goal?
6. Do you think you made a mistake?
7. Have you tried anything else?
8. Why didn’t that work?
9. Please speak up.
10. What are you thinking?

Level 1: More Information/Help
11. You are in the correct form. Have you tried everything in this form?
12. Please try clicking other push buttons in this form.
13. What do you think this button does?

Level 2: Suggestions Given
14. Have you tried this button?

Level 3: Instructions Given
15. Press this button.
Appendix 7

Questionnaire #1
Subject Background Questionnaire
Ooem Case Tool Study
Fall 1996

General:

Name: __________________________

Computer Experience Questions:

Please circle your choice where appropriate.

1. • Have you ever used a windowing system e.g. Apple MacIntosh, Microsoft (MS) Windows?
   Yes / No (if No, go directly to next question)

   • If yes, which one(s) and for how many years?
     (please circle and write length of use)

     MS Windows 3.11_____yrs      MS Windows 95_____yrs
     Apple MacIntosh_____yrs      X-Windows_____yrs
     Other_______________________/_____ yrs

2. How comfortable are you with using a windowing system?

   Very Uncomfortable 1 2 3 Neutal 4 5 Very Comfortable 6 7

Personal Assessment Questions:

3. I feel I need an experienced person nearby to assist me when I use the computer.

   Strongly Disagree 1 2 3 Neutral 4 5 Strongly Agree 6 7

4. I feel confident about using the computer to store important information.

   Strongly Disagree Neutral Strongly Agree
Questionnaire #1
Subject Background Questionnaire
OOEM CASE Tool Study
Fall 1996

CASE Tool and Modeling Experience Questions:

5. Have you taken a Systems Analysis or similar course other than COMM 436/536 or BAIT 506?
   • Yes / No (If No, go directly to next question)
   • If yes, at what level?
     Undergraduate/ Graduate/ Professional / Other

5. How long was that course?
   _______ months Part-time or Full-time?

5. What CASE/modeling tool(s) were you taught in the course?
   (Please circle and/or write name of tool)
   SilverRun DFD
   Other

6. Have you ever been worked as a Systems Analyst or similar role?
   • Yes / No (If No, go directly to next question)
   • If yes, for how many years? ________ years

7. Have you ever been taught to use Dr(s) Wand and Woo object-oriented enterprise modeling method (with 7 ontology-based rules)?
   Yes / No

Appendices: 117
8. From the list of CASE Tools below, indicate your level of proficiency against any CASE Tool you know below.

Please circle “1” for novice user; “2” for mid-level user; “3” for advanced user.

BridgePoint { 1 / 2 / 3 } Designer 2000 { 1 / 2 / 3 } Iconix { 1 / 2 / 3 }

MetaEdit { 1 / 2 / 3 } ObjectMaker { 1 / 2 / 3 } OMT Select { 1 / 2 / 3 }

OOA Tool { 1 / 2 / 3 } Paradigm { 1 / 2 / 3 } Playground { 1 / 2 / 3 }

ROSE { 1 / 2 / 3 } SilverRun { 1 / 2 / 3 }

Other (specify) ______________________________________ { 1 / 2 / 3 }
(Please specify if you have used a CASE Tool whose name does not appear in the list above)

9. Having used a CASE Tool before, I would rather use just pen and paper than a computerized CASE Tool to build a model.

Strongly Disagree 1 2 3 Neutral 4 5 6 Strongly Agree 7

Appendices: 118
Appendix 8

Questionnaire #2
Study Feedback Questionnaire
OOEM CASE Tool Study
Fall 1996

General:

Name: ________________________________

Mode used: Guided / Free Form

Tool Utility:

1. Using the CASE Tool, I found it generally easy to build a model.

   Strongly Disagree  Neutral  Strongly Agree
   1  2  3  4  5  6  7

2. This question is broken down into 4 related sub-questions. They differ only in the bold text.

   - Using the CASE Tool, I found it easy to isolate and identify Objects from the case study notes.

   Strongly Disagree  Neutral  Strongly Agree
   1  2  3  4  5  6  7

   - Using the CASE Tool, I found it easy to isolate and identify Requests from the case study notes.

   Strongly Disagree  Neutral  Strongly Agree
   1  2  3  4  5  6  7

   - Using the CASE Tool, I found it easy to isolate and identify Attributes from the case study notes.

   Strongly Disagree  Neutral  Strongly Agree
   1  2  3  4  5  6  7

   - Using the CASE Tool, I found it easy to isolate and identify Services from the case study notes.

   Strongly Disagree  Neutral  Strongly Agree
   1  2  3  4  5  6  7
3. This question is broken down into 4 related sub-questions. They differ only in the bold text

- Using the CASE Tool, I found it easy to keep track of (remember) the **Objects** I had entered into the system.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Using the CASE Tool, I found it easy to keep track of (remember) the **Requests** I had entered into the system.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Using the CASE Tool, I found it easy to keep track of (remember) the **Attributes** I had entered into the system.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Using the CASE Tool, I found it easy to keep track of (remember) the **Services** I had entered into the system.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4. Using the CASE Tool, I found it easy to remember where I was and what I had to do next.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5. Using the CASE Tool, I found the Model Viewer window helpful.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Questionnaire #2
Study Feedback Questionnaire
OOEM CASE Tool Study
Fall 1996

6. Using the CASE Tool, I found it easy to keep track of the links (relationship) between the following in the system... (please circle inside the table)

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2  3</td>
<td>4   5 6 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requests</th>
<th>Attributes</th>
<th>Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objects</td>
<td>1 2 3 4 5 6 7</td>
<td>1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Requests</td>
<td>1 2 3 4 5 6 7</td>
<td>1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Attributes</td>
<td>1 2 3 4 5 6 7</td>
<td>1 2 3 4 5 6 7</td>
</tr>
<tr>
<td>Services</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Using the CASE Tool, I found it easy to trace a request from its origin to the end.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2  3</td>
<td>4   5 6 7</td>
</tr>
</tbody>
</table>

8. Using the CASE Tool, I was able to check the integrity of the model as I went along and that resulted in less errors.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2  3</td>
<td>4   5 6 7</td>
</tr>
</tbody>
</table>

9. Using the CASE Tool, I will be able to build models of greater size and complexity than I would with just pen and paper.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2  3</td>
<td>4   5 6 7</td>
</tr>
</tbody>
</table>

10. Using the CASE Tool, I will be able to build models quicker than I would with just pen and paper.

<table>
<thead>
<tr>
<th>Strongly Disagree</th>
<th>Neutral</th>
<th>Strongly Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Appendices: 121
Questionnaire #2
Study Feedback Questionnaire
OOEM CASE Tool Study
Fall 1996

11. Using the CASE Tool, it was fun to build the model.

   | Strongly Disagree | Neutral | Strongly Agree |
---|-------------------|---------|----------------|
   | 1                 | 2       | 3              |
   | 4                 | 5       | 6              |
   | 7                 |         |                |

Appendices: 122
Appendix 9

Post Study Interview Record
OOEM CASE Tool Study
Fall 1996

General:

Name:__________________________________________________

Mode used: Guided / Free Form

User Interface:

1. What did you like about the interface?

   ______________________________________________________
   ______________________________________________________
   ______________________________________________________
   ______________________________________________________

2. Were there specific areas you especially liked?

   ______________________________________________________
   ______________________________________________________
   ______________________________________________________
   ______________________________________________________
3. What did you NOT like about the interface?

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

4. Were there specific areas you feel need improvement and why?

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

5. Why did you draw a pictorial diagram of the model before you began using the CASE Tool (for those who did)?

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

__________________________________________________________________________

Appendices: 124
Appendix 10

Table Definitions

The Object Table Definition

![Table Definition Diagram]

- **Field Name**: ObjectID, ObjectName, ObjectType
- **Type**: A
- **Size**: 50, 9
- **Key**: * (index)

Enter a field name up to 25 characters long.
Table Definitions

The Generate Table Definition

[Image: Restructure Paradox 5.0 for Windows Table: TGenerate.DB]

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Size</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>GenerateID</td>
<td>T</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>ObjectID</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequestID</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enter a field name up to 25 characters long.

Pack Table

Save  Save As  Cancel  Help
Table Definitions

The Request Table Definition
Table Definitions

The Service Table Definition

Restructure Paradox 5.0 for Windows Table: TService.DB

<table>
<thead>
<tr>
<th>Field</th>
<th>Field Name</th>
<th>Type</th>
<th>Size</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ServiceID</td>
<td>+</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ServiceName</td>
<td>A</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ObjectId</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table properties:
- Secondary Indexes
- Define
- Modify

ByObjectID
ByServiceName

Enter a field name up to 25 characters long

Pack Table

Save | Save As | Cancel | Help

Appendices: 128
Table Definitions

The Attribute Table Definition

Restructure Paradox 5.0 for Windows Table: TAttrib.DB

Field roster:

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Size</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 AttributeID</td>
<td>+</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>2 AttributeName</td>
<td>A</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>3 AttributeType</td>
<td>A</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>4 ObjectID</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Enter a field name up to 25 characters long.

Pack Table

Save  Save As  Cancel  Help
Table Definitions

The Access Table Definition

Restructure Paradox 5.0 for Windows Table: TAccess.DB

Field roster

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Type</th>
<th>Size</th>
<th>Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>AccessID</td>
<td>+</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>ServiceID</td>
<td>N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AttributeID</td>
<td>N</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table properties:

Secondary Indexes

Define | Modify

ByAttributeID
ByServiceID

Enter a field name up to 25 characters long.

Pack Table

Save | Save As | Cancel | Help

Appendices: 130