

Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be “used for any purpose other than private study, scholarship, or research.” If a user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of “fair use” that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select “Pages from: first page # to: last page #” on the print dialog screen



The Van Houten library has removed some of the personal information and all signatures from the approval page and biographical sketches of theses and dissertations in order to protect the identity of NJIT graduates and faculty.

ABSTRACT

RELATIONSHIP ANALYSIS: IMPROVING THE SYSTEMS ANALYSIS PROCESS

**by
Joseph Thomas Catanio**

A significant aspect of systems analysis involves discovering and representing entities and their inter-relationships. Guidelines exist to identify entities but do not provide a rigorous and comprehensive process to explicitly capture the relationship structure of the problem domain. Whereas, other analysis techniques lightly address the relationship discovery process, Relationship Analysis is the only systematic, domain-independent analysis technique focusing exclusively on a domain's relationship structure.

The quality of design artifacts, such as class diagrams, and development time necessary to generate these artifacts can be improved by first representing the complete relationship structure of the problem domain. The Relationship Analysis Model is the first theory-based taxonomy to classify relationships. A rigorous evaluation was conducted, including a formal experiment comparing novice and experienced analysts with and without Relationship Analysis. It was shown that the Relationship Analysis Process based on the model does provide a fuller and richer systems analysis, resulting in improved quality of and reduced time in generating class diagrams. It also was shown that Relationship Analysis enables analysts of varying experience levels to achieve a similar level of quality of class diagrams. Relationship Analysis significantly enhances the systems analyst's effectiveness, especially in the area of relationship discovery and documentation resulting in improved analysis and design artifacts.

**RELATIONSHIP ANALYSIS:
IMPROVING THE SYSTEMS ANALYSIS PROCESS**

by
Joseph Thomas Catanio

**A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
In Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Information Systems**

Department of Information Systems

May 2004

Copyright © 2004 by Joseph Thomas Catanio

ALL RIGHTS RESERVED

APPROVAL PAGE

**RELATIONSHIP ANALYSIS:
IMPROVING THE SYSTEMS ANALYSIS PROCESS**

Joseph Thomas Catanio

~~Dr. Michael P. Bieber, Dissertation Advisor~~ Date
Associate Professor of Information Systems, NJIT

~~Dr. Jane Cheng, Committee Member~~ Date
Professor of Computer Information Systems, Bloomfield College

Dr. Fadi P. Deek, Committee Member Date
Professor of Information Systems, NJIT

Dr. Il Im, Committee Member Date
Assistant Professor of Information Systems, NJIT

~~Dr. Vassilka Kirova, Committee Member~~ Date
Adjunct Professor of Computer Science, NJIT

Dr. Ravi Paul, Committee Member Date
Assistant Professor of MIS, East Carolina University

BIOGRAPHICAL SKETCH

Author: Joseph Thomas Catanio
Degree: Doctor of Philosophy
Date: May 2004

Undergraduate and Graduate Education:

- Doctor of Philosophy in Information Systems,
New Jersey Institute of Technology, Newark, NJ, 2004
- Masters of Science in Computer Science,
New Jersey Institute of Technology, Newark, NJ, 1999
- Bachelor of Science in Electrical Engineering,
Rutgers University, New Brunswick, NJ, 1987

Major: Information Systems

Presentations and Publications:

- Catania, J., & Bieber, M. (2004). Web Engineering: Principles and Techniques, Chapter Entitled: Relationship Analysis: A Technique to Enhance Systems Analysis for Web Development, Idea Group Publishing, (forthcoming).
- Catania, J., Galnares, R., Zhang, L., & Bieber, M. (2004). "Ubiquitous Metainformation and the WYWWYWI Principle," *Journal of Digital Information*, (forthcoming).
- Catania, J., Yoo, J., Bieber, M., & Paul, R. (2004). "Relationship Analysis in Requirements Engineering," *Requirements Engineering Journal*, (forthcoming).
- Catania, J., & Bieber, M. (2003). "Relationship Analysis: A Plan to Enhance Systems Analysis," 2nd Annual Symposium on Research on Systems Analysis and Design, Florida International University.
- Catania, J., Bieber, M., Im, I., Paul, R., Yoo, J., Ghoda, A., Pal, A., & Yetim, F. (2003). "Relationship Analysis: A Research Plan for Enhancing Systems Analysis For Web Development," *Proceedings of the 36th Hawaii International Conference on System Sciences*, IEEE Press, Washington, D.C.

Dedicated to GOD

ACKNOWLEDGMENT

I would like to express my deepest and sincere appreciation to Dr. Michael Bieber, my advisor, for his guidance, encouragement, friendship, time, and moral support throughout this research. Special thanks are given to Dr. Jane Cheng, Dr. Fadi Deek, Dr. Il Im, Dr. Vassilka Kirova, and Dr. Ravi Paul for actively participating in my committee. In addition, I would like to thank both Dr. Roxanne Hiltz and Dr. Fadi Deek for their financial assistance while I was a full-time student completing my dissertation.

Many thanks are extended to Dr. Michael Chumer, Professor Osama Eljabiri, and Dr. Vassilka Kirova for allowing me to recruit students from their classes for the experiment portion of my dissertation. Also, thanks to the students for their participation.

I also appreciate the time and effort of John Discepola and Ronald Lazeration, my expert judges. Also, special thanks to Michael Gorman.

In addition, I thank both Dr. Yuanqiong (Kathy) Wang and Dr. Il Im for their guidance and expertise for the data analysis portion of my dissertation.

TABLE OF CONTENTS

Chapter	Page
1 INTRODUCTION	1
1.1 Motivation	2
1.2 Dissertation Overview	3
1.3 Dissertation Outline	3
1.4 Contributions.....	4
1.5 Dissertation Boundaries	5
2 BACKGROUND LITERATURE.....	6
2.1 Software Engineering Development Process.....	7
2.1.1 Iterative Software Development	9
2.1.2 Agile Software Development.....	11
2.2 Software Project Management	17
2.3 Human Aspects	22
2.4 Analysis Framework	24
2.5 Structured Analysis	28
2.5.1 Data Flow Diagrams	32
2.5.2 Data Dictionary	33
2.5.3 Mini-specifications	34
2.5.4 Structured Walkthrough	35
2.5.5 Deficiencies	35
2.5.6 Summary	36
2.6 Object-oriented Analysis	37
2.6.1 Characteristics	40

TABLE OF CONTENTS
(Continued)

Chapter	Page
2.6.2 Object Identification	42
2.6.3 Object Communications	45
2.6.4 Object Behavior	49
2.6.5 Object Operations	52
2.6.6 Summary	53
2.7 Use-case Analysis	55
2.7.1 Identification	57
2.7.2 Structuring a Use-case	59
2.7.3 Deficiencies	61
2.7.4 Summary	63
2.8 Domain Analysis	64
2.8.1 Feature Oriented Domain Analysis.....	66
2.8.2 Organization Domain Modeling	66
2.8.3 Standardizing Product Re-use.....	67
2.8.4 Summary	69
2.9 Requirements Analysis	70
2.9.1 Eliciting Requirements	73
2.9.2 Documenting Requirements	79
2.9.3 Verifying and Validating Requirements	82
2.9.4 Requirements Management	85
2.9.5 Summary	87

TABLE OF CONTENTS
(Continued)

Chapter	Page
2.10 Relationship Analysis	88
2.11 Analysis Quality	94
2.11.1 Quality Attributes of Natural Languages.....	94
2.11.2 Generating a Quality Specification.....	100
2.11.3 Quantifying the Specification Quality	102
2.12 Summary and Conclusion	106
3 RELATIONSHIP ANALYSIS THEORY	110
3.1 Creativity in Software Engineering	110
3.2 Conceptual Modeling.....	114
3.3 Existing Methodologies	117
3.4 Existing Theories	131
3.4.1 Ontological Theory.....	132
3.4.2 Classification Theory.....	136
3.4.3 Speech Act Theory.....	138
3.4.4 Structure of Intellect Theory (SI).....	140
3.5 Relationship Analysis Model (RAM).....	143
3.5.1 Unit Focus.....	145
3.5.2 Collection Focus	147
3.5.3 Comparison Focus	150
3.5.4 System Focus	152
3.5.5 Transformation Focus	153

TABLE OF CONTENTS
(Continued)

Chapter	Page
3.5.6 Implication Focus	155
3.5.7 Relationships Among the Relationships	157
3.5.8 Mapping RAM to RAF	158
3.6 Summary	161
4 RELATIONSHIP ANALYSIS APPLIED	163
4.1 Relationship Analysis Process (RAP).....	164
4.2 Relationship Analysis Template (RAT).....	165
4.3 Relationship Analysis Diagram (RAD)	167
4.4 Summary	169
5 EXPERIMENTAL DESIGN	170
5.1 Overview	170
5.2 Hypotheses	171
5.3 Method	178
5.4 Subjects	179
5.5 Procedures	179
5.6 Measures	180
5.6.1 Questionnaires	181
5.6.2 Expert Judges	183
5.7 Task	184
6 EXPERIMENTAL RESULTS AND DATA ANALYSIS	188
6.1 Subject Background Information	189

TABLE OF CONTENTS
(Continued)

Chapter	Page
6.1.1 Subject Pre-experiment Evaluation	189
6.1.2 Subject Distribution	191
6.2 Expert Judge Reliability.....	193
6.3 Experiment Hypotheses Analysis	194
6.3.1 Analysis Quality Grade Variable.....	195
6.3.2 Class Diagram Analysis Time Generation Variable.....	199
6.3.3 Total Analysis Time Generation Variable.....	202
6.3.4 Relationship Analysis Time Variable.....	204
6.4 Post-experiment Questionnaire Evaluation.....	205
6.4.1 Factor Analysis	205
6.4.2 Satisfaction.....	207
6.4.3 Analysis Ability	211
6.4.4 Task Comprehension	215
6.5 Summary of Hypotheses Analysis.....	218
7 DISCUSSION, CONCLUSIONS, AND FUTURE RESEARCH.....	219
7.1 Summary of Hypotheses Evaluation Results.....	219
7.2 Debrief Session Subject Comments.....	220
7.3 Experimentation Enhancements.....	221
7.4 Research Contributions.....	222
7.5 Future Research	223

TABLE OF CONTENTS
(Continued)

Chapter	Page
APPENDIX A CONSENT FORM	226
APPENDIX B TRAINING MATERIALS	229
APPENDIX C SURVEY INSTRUMENTS	250
APPENDIX D TASK LISTS.....	260
APPENDIX E COVER SHEETS	265
APPENDIX F PROBLEM STATEMENT	267
APPENDIX G PROBLEM STATEMENT SOLUTION	268
REFERENCES	271

LIST OF TABLES

Table		Page
2.1	Disciplines of Management	19
2.2	Categories of System Failures	20
2.3	Herzberg's Hygiene and Motivational Factors	23
2.4	Structured Analysis Framework Characteristics.....	37
2.5	Object-oriented Analysis Framework Characteristics	54
2.6	Use-case Recipe	58
2.7	Coleman's Use-case Template.....	61
2.8	Use-case Analysis Framework Characteristics	63
2.9	Domain and Re-useability Criterion	68
2.10	Domain Analysis Framework Characteristics	70
2.11	Relative Cost to Repair a Software Error in Different Stages	71
2.12	Major Software Verification and Validation Activities.....	82
2.13	Software Test Plan Structure	84
2.14	Requirements Analysis Framework Characteristics	88
2.15	Relationship Analysis Generic Relationships	89
2.16	Sample Brainstorming Questions	91
2.17	Relationship Analysis Framework Characteristics	93
2.18	Quality Attributes.....	96
2.19	Examples of Selected Quality Characteristics	99
2.20	Physical SRS Quality Attributes	103
2.21	Quality Attribute Metrics	105
3.1	Semantic Models.....	129

LIST OF TABLES
(Continued)

Table	Page
3.2 Mapping of Ontological Constructs to Conceptual Model Constructs.....	134
3.3 Relationship Analysis Model Using SI Nomenclature	144
3.4 Relationship Analysis Model (RAM)	145
3.5 Relationship Map Between Yoo’s Categories and RAM	159
3.6 Relationship Determination Questions	159
4.1 Generic Relationship Analysis Template.....	166
5.1 Factorial Design Experiment	178
5.2 Dependent Variable Measurement Methods.....	181
5.3 Pilot Study Summary	186
6.1 Subject Experience Score Details	190
6.2 Subject Experience Score Mean and Standard Deviation Calculations.....	190
6.3 Summary of Subject Distribution	191
6.4 Group Identification Numbers per Condition	191
6.5 Summary of Total Number of Groups per Condition.....	192
6.6 Correlation Between Judges Before Meeting	193
6.7 Correlation Between Judges After Meeting.....	193
6.8 Results Overview	195
6.9 Quality Grade Details	196
6.10 Quality Grade Normality Test Details	196
6.11 Quality Grade Mean and Standard Deviation Calculations.....	197
6.12 Quality Grade Main Effect 1 (Analysis Tool)	197

LIST OF TABLES
(Continued)

Table	Page
6.13 Quality Grade Main Effect 2 (Experience Level).....	197
6.14 Quality Grade Interaction Effect (Part 1).....	198
6.15 Quality Grade Interaction Effect (Part 2).....	198
6.16 Class Diagram Time Details	200
6.17 Class Diagram Time Normality Test Details.....	201
6.18 Class Diagram Time Mean and Standard Deviation Calculations.....	201
6.19 Total Analysis Time Details	202
6.20 Total Analysis Time Mean and Standard Deviation Calculations.....	203
6.21 Relationship Elicitation Time Details.....	204
6.22 Relationship Elicitation Time Mean and Standard Deviation Calculations ...	205
6.23 Summary of Cronbach’s Alpha, Questions, and Factors.....	206
6.24 Satisfaction Normality Test Details.....	209
6.25 Satisfaction Mean and Standard Deviation Calculations.....	209
6.26 Satisfaction Main Effect 1 (Analysis Tool)	209
6.27 Satisfaction Main Effect 2 (Experience Level).....	210
6.28 Satisfaction Interaction Effect (Part 1)	210
6.29 Satisfaction Interaction Effect (Part 2)	211
6.30 Analysis Ability Normality Test Details.....	212
6.31 Analysis Ability Mean and Standard Deviation Calculations	212
6.32 Analysis Ability Main Effect 1 (Analysis Tool).....	212
6.33 Analysis Ability Main Effect 2 (Experience Level)	213

LIST OF TABLES
(Continued)

Table	Page
6.34 Analysis Ability Interaction Effect (Part 1).....	214
6.35 Analysis Ability Interaction Effect (Part 2).....	214
6.36 Task Comprehension Normality Test Details.....	215
6.37 Task Comprehension Mean and Standard Deviation Calculations.....	215
6.38 Task Comprehension Main Effect 1 (Analysis Tool).....	215
6.39 Task Comprehension Main Effect 2 (Experience Level)	216
6.40 Task Comprehension Interaction Effect (Part 1)	217
6.41 Task Comprehension Interaction Effect (Part 2)	217
6.42 Hypotheses Summary	218
B1.1 Order Training Unit Template.....	232
B1.2 Order Training Collection Template	233
B1.3 Order Training Comparison Template	234
B1.4 Order Training System Template.....	235
B1.5 Order Training Transformation Template.....	236
B1.6 Order Training Implication Template	237
B1.7 Customer Training Unit Template	240
B1.8 Customer Training Collection Template.....	241
B1.9 Customer Training Comparison Template.....	242
B1.10 Customer Training System Template.....	243
B1.11 Customer Training Transformation Template.....	244
B1.12 Customer Training Implication Template.....	245

LIST OF FIGURES

Figure	Page
2.1 The Waterfall Model.....	8
2.2 Component Interaction Characteristics.....	26
2.3 Component Interaction Analysis Framework	27
2.4 SADT Activity Diagram Constructs.....	30
2.5 DeMarco-Yourdon Constructs.....	31
2.6 Object Structure	38
2.7 Sequence Diagram	46
2.8 Collaboration Diagram.....	47
2.9 Object Interaction Diagram (Less Detail).....	48
2.10 Object Interaction Diagram (More Detail)	49
2.11 State-chart Diagram	51
2.12 Use-case (UML Notation).....	60
3.1 The Role of a Conceptual Model in Systems Development.....	115
3.2 Guilford's Structure of Intellect Model	141
3.3 Unit Focus.....	146
3.4 Collection Focus	149
3.5 Comparison Focus	151
3.6 System Focus	153
3.7 Transformation Focus	155
3.8 Implication Focus.....	157
4.1 Generic Relationship Analysis Diagram (RAD).....	168
6.1 Quality Grade for Groups	199

LIST OF FIGURES
(Continued)

Figure	Page
B1.1 Use-case Analysis Training.....	230
B1.2 Order Training RAD	238
B1.3 Training Class Diagram 1	239
B1.4 Customer Training RAD	246
B1.5 Training Class Diagram 2	247
B1.6 Training Class Diagram Final	248
B1.7 Training Class Diagram Final Without Attributes	249
G1.1 Solution Use-case Analysis Diagram.....	268
G1.2 Solution Class Diagram Without Attributes	269
G1.3 Solution Class Diagram	270

CHAPTER 1

INTRODUCTION

The literature indicates that the best way to improve the software development life-cycle is to improve it during the early stages of the process (Sommerville, 2001) (Faulk, 2000) (Wieringa, 1998) (Booch et al., 1998); in particular, during the elicitation, analysis and design phases. The literature describes that the more effective techniques to elicit requirements of a computer or information system involve collaboration (Jurison, 1999) (Gill & Pidduck, 2001) (Haywood, 1998) (Paré & Dubé, 1999) and collaborative efforts have been demonstrated to aid analysts by improving the problem solving process (Wilson et al., 1993) (Sabin & Sabin, 1994) (Nosek, 1998) (Selvin, 1999). A technique known as use-case analysis is a widely accepted method to eliciting requirements for software systems. Use-case analysis is a scenario-based technique that captures the desired system functionality from the user's perspective in a team-oriented, user-inclusive strategy. Once the features of the system are identified, a more technical description or analysis of the problem solution is performed. The research described in the background literature chapter discusses various software engineering analysis techniques. As a result, a gap in the system analysis process has been identified, namely how to explicitly identify and document the relationship structure of a problem domain. This dissertation presents a rigorous and systematic process based on theory to identify and document the relationship structure of an application domain. While not an integral aspect, the experiments will be conducted using groups of analysts.

1.1 Motivation

During the system analysis phase, components are determined through the identification process of the system's entities and relationships. Informal guidelines exist to help identify entities or objects (Chen, 1976) (Rumbaugh, 1991) (Booch, 1994). In addition, prior to Yoo's dissertation on Relationship Analysis (RA), no guidelines existed to analyze an application domain in terms of its relationship structure (Yoo, 2000). The determination of an application domain relationship structure is an implicit process. No defined processes, templates, or diagrams exist to explicitly and systematically assist in eliciting relationships or documenting them in Class Diagrams or Entity-Relationship (E/R) Diagrams (Beraha & Su, 1999). However, relationships constitute a large part of an application domain's implicit structure. Completely understanding the domain relies on knowing how all the entities are interconnected. Relationships are a key component lightly addressed by E/R and class diagrams. These diagrams capture a limited subset of relationships and leave much of the relationship structure out of the design and system model. While analyses and models are meant to be a limited representation of a system, the incomplete relationship specification is not by design, but rather a lack of any methodology to determine them explicitly (Bieber & Yoo, 1999) (Bieber, 1998). As a result, many analyses miss aspects of the systems they represent. RA addresses these concerns. It provides a way of identifying the relationship structure of a problem domain and helps fill a void in the systems analysis process.

However, RA has not yet been fully developed and exists as an informal process. In addition, it is not based on a theoretical foundation. The motivations of this research are to utilize the concepts learned from Yoo's studies and develop RA from the ground

up. This dissertation will create a rigorous and systematic process based on theory to identify the relationship structure of an application domain.

1.2 Dissertation Overview

My dissertation, *Relationship Analysis: Improving the Systems Analysis Process*, will present a rigorous and systematic technique to identify and document the relationship structure of an application domain.

This research addresses a major deficiency in today's software engineering analysis techniques, namely a systematic process to identify and document relationships in a system being modeled. A significant aspect of systems analysis and design involves discovering and representing entities and their relationships. However, existing techniques leave the relationship determination as an implicit process. The proposed Relationship Analysis Process (RAP) provides a rigorous and systematic process to explicitly identify and document the relationship structure of the application domain. In support of the process, a Relationship Analysis Template (RAT) and Relationship Analysis Diagram (RAD) are utilized.

1.3 Dissertation Outline

The dissertation includes seven chapters. Chapter 1 introduces the dissertation topic by describing it in terms of motivation, contributions, and boundaries. Chapter 2 consists of a literary review of various software engineering analysis techniques. This extensive research identified a void in system analysis that is addressed in this dissertation.

Chapter 3 provides the theoretical background to RA by developing a new RA model (RAM), grounded in theory, utilizing Guilford's Structure of Intellect Theory (Guilford, 1956) (Guilford, 1967). Chapter 4 describes the Relationship Analysis Process as a rigorous and systematic technique. Chapter 5 outlines the experimental design to test comparisons of system analysis with and without Relationship Analysis in a controlled environment. Chapter 6 discusses the experimental results and data analysis. Chapter 7 describes the conclusions and future research. Consent forms, survey instruments, and task lists are included in the Appendix sections followed by the references.

1.4 Contributions

The contributions provided by this dissertation are geared towards eventual widespread use of Relationship Analysis and its incorporation into mainstream software engineering methodologies. Relationship Analysis has the potential to become an invaluable elicitation and analysis technique regardless of the software engineering approach taken during the analysis process. As a result, a deeper understanding of the application domain is expected. This dissertation builds from the concepts initiated by Yoo's dissertation on identifying relationship structures of application domains. His work provided a first cut to Relationship Analysis (RAF) and the proof-of-concept that Relationship Analysis is both feasible and will provide a major contribution. This dissertation will utilize concepts learned from RAF and develop it into a rigorous and systematic process based on theory to identify and document the relationship structure of an application domain.

The proposed Relationship Analysis Process (RAP) integrates into current object-oriented analysis (OOA) processes to fill an important gap of how to explicitly identify and document the relationship structure of an application domain. The dissertation will produce the following:

- A Relationship Analysis Model (RAM) based on theory
- A rigorous and systematic user-centered Relationship Analysis Process (RAP)
- A domain-independent Relationship Analysis Template (RAT) for eliciting information about the relationship structure of an application domain
- A Relationship Analysis Diagram (RAD) that documents the relationship structure of a domain, which would greatly assist in developing class diagrams

The RAP, RAT, and RAD should enhance the systems analyst's effectiveness, especially in the area of relationship discovery and documentation. Further contributions are presented in Chapter 7.

1.5 Dissertation Boundaries

Relationship Analysis is not a design technique. Rather, it is a method-independent analysis technique, which provides useful input to the system's design phase. This dissertation does not discuss how designers use the information provided by the RA analysis technique. Instead, the dissertation shows how the explicitly identified relationships help designers with varying experience levels to create class diagrams.

RA has been developed utilizing computer systems centric application domains. Although it may be generic enough to apply to other problem domains, this dissertation does not address those possibilities.

CHAPTER 2

BACKGROUND LITERATURE

The term Software Engineering evolved from the need to apply a systematic process to the development, operation, and maintenance of a software product. To meet the need for high quality, low cost software systems, delivered quickly, firms balance the issues of quality, cycle time, and effort to determine tradeoffs to be managed and improved during the software development life-cycle process (Harter et al., 1998). The development of software systems generally requires a group effort utilizing specialized skills and knowledge. This collaborative approach requires effective team management (Donnelly et al., 1998). The best way for a group to share a common terminology is through formal written specifications that describe the problem to be solved (IEEE, 1998) (Faulk, 2000) (Thayer & Dorfman, 2000) (Sommerville, 2001). These documents outline the scope of the project and the problem domain as well as capture the user requirements and overall desired system functionality. The process of documenting the requirements is the first step to the software development life-cycle. The primary reason why some software systems are unsuccessful is that requirements analysis is poorly done or not performed at all (Abdel-Hamid & Madnick, 1989). Therefore it stands to reason that analysis is a key component to the software development life-cycle process and improving this phase should help to improve the entire process. This chapter is a literary review of a wide assortment of analysis techniques that can be used to help analyze and design software systems. Although there are many techniques, the underlying commonality or objective of different analysis techniques is to gain a better understanding of the problem domain.

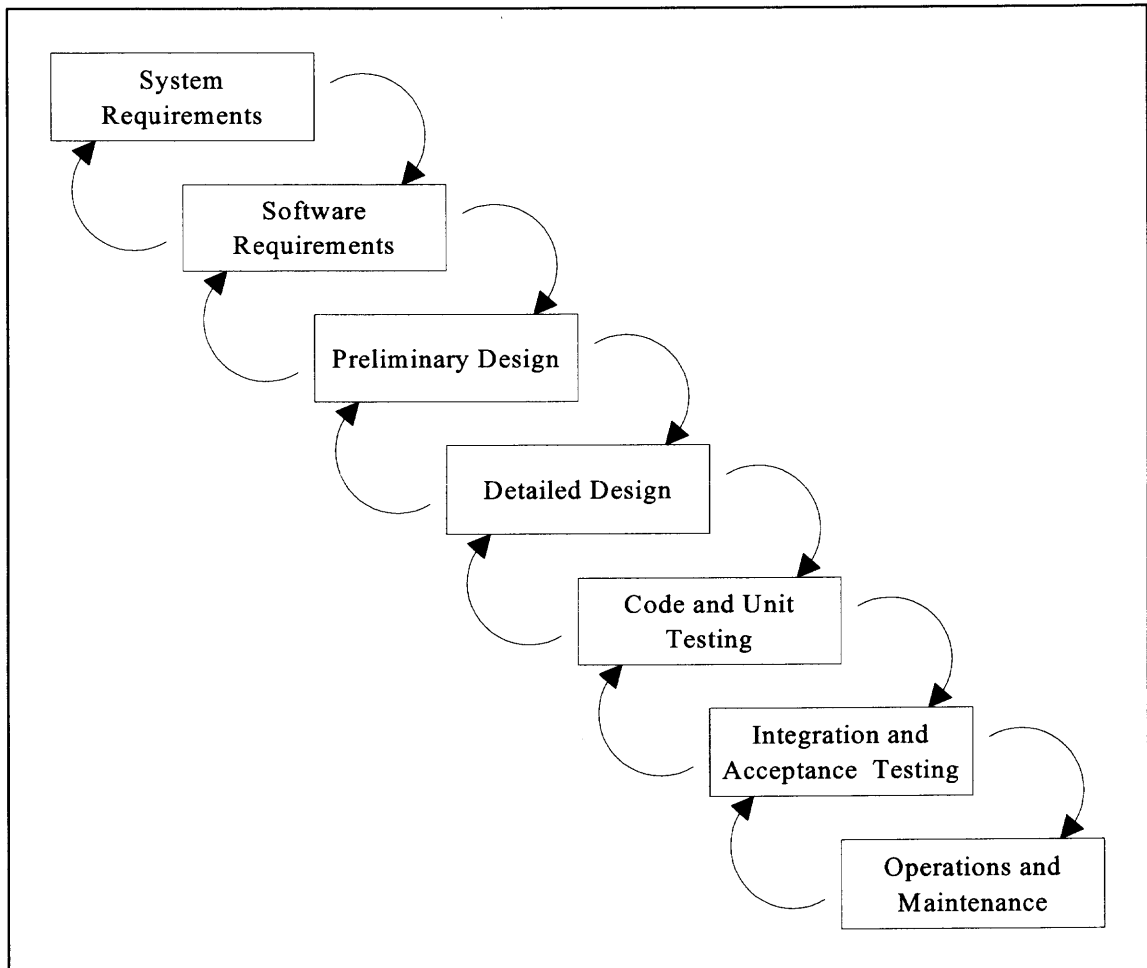
2.1 Software Engineering Development Process

The Institute of Electrical and Electronics Engineers, Inc. (IEEE) defined the term software engineering in 1993 as the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. Software engineers have developed methodologies that assist them to develop software products. These methodologies encompass the entire software development life-cycle process. The software development process or software life-cycle begins with a statement of software requirements and ends with the product being retired (IEEE, 1998) (Blum, 1994). Therefore, the software process is the progression from the identification of some application specific domain need to the creation and delivery of a software product to fulfill that need. To understand the need, one must first understand the application domain. Analysis exists at the application domain level and conceptual models are used to explain the application need and describe domain concepts. Models are prescriptive in nature and are intended to provide clear and concise software requirements, which are then utilized to construct the software system. The most fundamental software development process activities are specification, development, validation, and evolution (Sommerville, 2001). These activities can be realized using a varying number of techniques and methods and the scope of this paper is to focus on the specification activity of the software development process.

There are different paradigms or approaches that software engineers can use to develop systems both effectively and efficiently. These paradigms organize the software development process activities in different ways. However, regardless of the method

chosen, it involves the specification, development, validation, and evolution activities. The baseline management paradigm strategy is used to coordinate and control the software development process (Thayer & Dorfman, 2000). Baseline management is based on the waterfall software development life-cycle model (Royce, 1970), which partitions the project into manageable pieces.

Figure 2.1 The Waterfall Model



Each component of the Waterfall model has its own process steps requiring people with various levels of expertise to successfully develop the software product solution. Taken verbatim, the Waterfall development model requires that the set of system and software requirements be determined and remain static before design and

implementation begin. Following the steps of the Waterfall model in such a sequential manner is too rigid and is a contributing factor to the software industry being plagued by cost overruns, late deliveries, poor reliability, and user dissatisfaction (Abdel-Hamid & Madnick, 1991). Therefore, variations to the Waterfall model have been created to be more iterative in process and more incremental in tangible assets.

2.1.1 Iterative Software Development

Most software engineering projects use the Waterfall model process steps as a guideline to follow in the overall life-cycle to coordinate and control the software development process. But instead of following a sequential developmental process, a more iterative approach is utilized. The iterative development approach addresses the shortcomings of the Waterfall model and provides an iterative process model to software development. Other common names for the iterative process are: incremental, evolutionary, staged, and spiral. The essence of the iterative process is that system and software specifications are developed jointly with the software code (Sommerville, 2001). This approach fosters a design for change attitude to software development. During the software development process, requirements change. The literature indicates that requirements change because users either do not know or find it difficult to describe the functionality needed in the software system. In addition, users needs evolve over time resulting in system and software requirements to also change to encompass these new needs. Changing users needs alter the value of software features rapidly, therefore making the software life-cycle process unpredictable. Since software development depends on requirements that are constantly changing, it is difficult to create a predictable plan. Iterative development

is an adaptive process because it can handle changes in required features. This leads to a style of planning and control where long-term plans are very fluid, and short-term plans relating to an iteration are stable. Iterative development provides a firm foundation in each iteration that is the basis for later plans.

When using incremental development, initial software releases contain limited functionality but are constructed in a way to facilitate incorporating new requirements. Davis argues that the incremental approach reduces the overall development time, allows software to be easily enhanced, and helps to meet users needs (Davis, 1988). Both rapid and evolutionary prototyping addresses the issue of ensuring the software development product meets users needs (Gomaa, 1990) (Boehm, 1988). The approach is to construct a series of partial implementations that can be used to elicit user feedback and provide users with working models of limited functionality. The feedback is used to modify the software requirements specification and incorporate these needs into the written specification and software asset. The benefits of using an iterative software process are (Gordon & Bieman, 1995):

- Improves System Usability
- Improves Match Between System Functionality and Users Needs
- Improves Design Quality
- Improves Maintainability
- Reduces Development Effort

These advantages are due to the ability of the iterative life-cycle to adapt to changing needs during the software development process. Consequently, higher quality software systems that meet users needs result. Providing incremental software releases affords the development team time to evolve and grow requirements, designs and implementations. The concept of developing software systems in an iterative fashion,

producing incremental assets based on change, is an adaptive process (Highsmith, 1999). Adaptive software development differentiates itself from the traditional process-oriented methodologies in that it is a people-oriented process (Cockburn, 1999). Software development that uses adaptive techniques is now known as an agile methodology.

2.1.2 Agile Software Development

Agile software development is a people-oriented process and there is a constant reworking of the management plan, requirements documents, design documents, and implementations with each iteration. This approach helps to identify risk early on in the development process instead of at the end. The earlier in the development process an error occurs and the later the error is detected, the more expensive it is to correct (Faulk, 2000). The two most prominent methodologies that fall under the agile life-cycle concept are Extreme Programming (XP) and The Rational Unified Process (RUP). It can be argued that the RUP is a heavyweight process and not a lightweight agile process since the RUP can also be used in a traditional waterfall style. The RUP is discussed in greater detail in subsequent paragraphs of this section.

The XP approach addresses the values communication, simplicity, feedback, and courage (Beck, 2000). XP can only work if people keep up the line of communication. This takes the place of any formal product specification and forces user involvement and user participation. Both are needed to ensure usage and acceptance (Barki & Hartwick, 1994) (Lederer et al., 1998) (Baroudi et al., 1986) (Saleem, 1996). Feedback facilitates communication and provides the current state of the system. The very nature of feedback nurtures open communication among all those involved in the project. Simplicity refers

to the development of the simple solution without worrying about thinking ahead. The argument for this stance is that spending time to implement a complicated feature that has not been requested and probably will not be used is a waste of time. As long as the solution provided is component-oriented it should allow for extensibility, and these features if requested in the future, can be incorporated. Both simplicity and communication complement each other. The greater the communication the clearer the overall picture of what the problem is and what has to be done to develop a solution. Just as the simpler the solution the less communication necessary to make that picture clear. Lastly, courage is an XP value that is more prevalent in this paradigm than others. This is primarily due to the lack of a formal specification that can be used to measure against. Management needs to have the courage to trust the team to get the job done. In addition, the developers need to have the courage to refine and simplify complex designs.

XP achieves design differently than other software development methodologies. Other methodologies stress rigorous up front design and analysis steps before coding begins. XP on the other hand starts with coding as soon as possible. An XP solution is comprised of the activities: coding, testing, listening, and designing (Beck, 2000). These activities are realized using twelve XP practices: planning, small releases, metaphor, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, 40-hour week, on-site customer, and coding standards. XP also recognizes that no one person can possibly understand all components and all aspects of the entire project, therefore ruling out centralized authority. Instead, XP institutes a decentralized decision making posture. The manager runs the process but allows those who are specialists in certain areas to make those decisions. The manager acts as a facilitator who

oversees the entire process and shares decision-making responsibilities with those who are familiar with a concept. The XP management tool is a chart metric. The chart is posted in a conspicuous location that tracks estimated development time and current calendar time. This allows the team to track product development and ascertain if they are on target or not. This approach allows the team to manage themselves without having a manager to dictate what has to be done and by when. It also aids in sharing responsibility with all team members and supports the idea that people want to do a good job and fosters that mind set. One of the most appealing activities is XPs emphasis on testing. Programmers write tests as they develop code. The tests are integrated into a continuous integration and build process that results in a highly stable platform. The platform evolves with successive iterations and yields a design process that is disciplined and adaptable.

The RUP is also a disciplined process that provides a framework that can accommodate a wide variety of processes. The process is iterative, object-oriented, controlled and can be used for the traditional waterfall heavyweight style or an agile manner of development depending on how it is tailored to the development environment of the organization. At its core, the RUP consist of nine fundamental workflows.

- Business Engineering: Understanding the needs of the business
- Requirements: Translating business needs into the behaviors of a computer system
- Analysis and Design: Translating requirements into software architecture
- Implementation: Developing software that fits within the architecture and conforms to the required behaviors
- Test: Ensuring that the required behaviors are correct
- Configuration and Change Management: Keeping track of all the different versions
- Project Management: Managing schedules and resources
- Environment: Setting up and maintaining the development environment
- Deployment: Everything necessary to release the final product

In contrast to a sequential development environment model, these activities are executed concurrently in an iterative fashion utilizing a phased approach throughout the lifetime of a product. Inception, elaboration, construction, and transition comprise the phases. In an iteration, all of the RUP activities are executed but as the project matures, the emphasis on certain activities increases or decreases depending on the project phase or stage. This permits change to be easily incorporated into the RUP iterative approach. Whereas XP works best with a small project scope and small groups, RUP's framework can accommodate both smaller and larger scale software projects. The literature indicates that XP is a minimal RUP process used in an agile manner (Booch et al., 1998) (Smith, 2001) (Martin, 2002). Similarly, Martin's dX (XP upside down) process fully complies with RUP's framework in an agile manner. Both the XP and dX methodologies utilize the RUP framework to software development but in an agile manner. Larman is also a supporter of agile RUP and uses the Unified Modeling Language (UML) to sketch out the design of work to be done during each iteration (Larman, 2001).

The UML is an industry-standard and accepted modeling language that is used to specify and document the data and processes of software system development in an object-oriented manner. It is unified because Grady Booch and James Rumbaugh merged their individual object-oriented modeling techniques into a single method known as the UML. In addition, they incorporated Ivar Jacobson's use-cases technique into the UML. A use-case describes how users and the system work together to realize the identified feature (Leffingwell, 2000). Booch, Rumbaugh, and Jacobson adopted four goals in the creation of the UML (Boggs & Boggs, 2002).

- To represent complete systems using object-oriented concepts
- To establish an explicit coupling between concepts and source code

- To take into account scaling factors inherent to complex and critical systems
- To create a modeling language usable by both humans and machines

These four overall design goals were at the heart of the first release in October 1995. It has evolved and expanded since, but the focus has always been on the object-oriented modeling language rather than the object-oriented method. Modeling is a core component of all the activities that lead up to the deployment of a good software system. A successful software organization can consistently deploy a good quality software system that meets the users needs and is usable. Therefore the best way to deploy a software system that meets the users needs and expectations is to properly capture the desired user requirements. In addition, to develop a quality software system it must have a solid architecture that can handle change. This implies that the development process used, must be systematic and adaptable to changing needs by users, business, and technology. Modeling and in particular the UML helps organizations achieve and realize this goal. Models provide the software system blueprints and are comprised of both general and detailed plans. Modeling helps designers to visualize a system and understand both static and dynamic behavioral characteristics. Models provide templates that can be used in constructing a system as well as a way to document decisions (Booch et al., 1998). The UML notation is platform independent and is designed to serve as an object-oriented modeling language, no matter how it is deployed. The UML can be used throughout the development life-cycle and across different implementation technologies. Similar to the openness school of thought followed by the creators of Linux, the UML is not a proprietary notation. Tool developers and training agencies may use it freely and Rational Rose is an object-oriented developmental tool that utilizes the UML. A Rational Rose model is a collection of diagrams that represent the software system from various

perspectives. The Rose model supports eight UML diagrams: use-case, activity, sequence, collaboration, class, state-chart, component, and deployment. These process diagrams are a graphical representation of the software system. They aid both users and developers in product planning and development and depict the system's static and dynamic state.

Static Views:

- Use-case Diagrams: Captures system functionality as seen by the users and is built at the project start.
- Class Diagrams: Outlines the vocabulary of the software system and are created and modified throughout the development process. These diagrams show the interactions between classes and their relationships amongst each other. It is comprised of a section that shows the class name while the other shows the class's attributes. Class diagrams are useful to people in different ways. Developers use class diagrams to develop classes. Analysts use them to understand the system details. Lastly, class diagrams depict the overall design of the system to architects.
- Component Diagrams: Describes the physical structure of the implementation and aids in organization and release building. There are two types of components, executable and library. A component diagram shows the compile-time and run-time dependencies. These components are connected by dashed lines and show their dependency relationships.
- Deployment Diagrams: Depicts the system's platform configuration and are useful to understand the physical layout of the system.

Dynamic Views:

- Sequence Diagrams: Represents the time-oriented dynamic control flow by showing the flow of functionality through a use-case.
- Collaboration Diagrams: Shows the message-oriented dynamic control flow between objects.
- State-chart Diagrams: Shows the event-oriented dynamic software system behavior. This provides a method to model the states in which an object exists. The state start is depicted by a black dot and shows its initial state at creation. The stop state is depicted as a black dot in a circle and shows the object's state just before destruction.
- Activity Diagrams: Shows the activity-oriented dynamic software system behavior. These diagrams depict the workflow process. They show the workflow start, end, and the order the activities occur.

Different people performing different roles on the project team use the static and dynamic view diagrams. Thus, the Rational Rose tool can be used by the entire project team and is a way for the team to systematically analyze and design a project.

Both XP and RUP are methodologies that utilize an agile software development approach. Its processes are iterative and represent an adaptive approach to software development that is people-oriented and rejects the assumption that people are replaceable components. Treating people as replaceable resources has its origins in Taylor's Scientific Management approach (Taylor, 1967). A Taylor model may be appropriate in a factory type setting but software development is a creative process and to hire and retain a competent staff requires a people-oriented management process.

2.2 Software Project Management

The initial process steps of the life-cycle process involves gathering and analyzing both system and software requirements. This involves the collection of information from the customer for whom the software engineering project is to be created. The collected information is to describe and outline the basic functionality of the software system to be designed. The description outlines what is to be designed based on criteria set forth by the customer. Requirements gathering is one of the first steps in any software life-cycle and builds the foundation that subsequent process steps rely. Some factors that affect the quality of these requirements involve experience, effort, time, and domain knowledge. Much of Boehm's research in this area has yielded interesting correlations between schedule and effort (Boehm & Egyed, 1998). The primary observation was that the more time spent creating a requirements document did not necessarily produce a high quality

document. His research has shown that a more important quality driver affecting a requirements document was that of previous experience and domain knowledge. Since subsequent software development process steps rely on the requirements phase, it seems prudent for an experienced domain expert to spend the time upfront and carefully generate meaningful, highly usable requirements documents. These documents are then used as a basis to generate design documents, testing and evaluation plans, and user guides. They are also used to produce a schedule used by the management team to track progress.

A project, software or some other type, needs to be planned and controlled through some form of management process to increase the likelihood of its successful completion. The management process binds and coordinates the activities surrounding a project's life-cycle. Software project management is an extension to project management in general and therefore shares the same characteristics and concerns. Thus, good product management of software projects is vital to its successful implementation. Project management involves planning, organizing, controlling, and leading a series of activities realized in a group environment to accomplish a particular goal. These four aspects of the project manager's role are known as the disciplines of management and are listed in Table 2.1 (Donnelly et al., 1998). It is the responsibility of the project manager to balance these disciplines within the atmosphere of the organization. Project management acts in the best interests of the organization utilizing the available resources to get the job done.

Table 2.1 Disciplines of Management

Disciplines of Management	
Plan	Write mission statement
	Define objectives
	Set priorities
	Assign responsibility for objectives
	Use forecasts to set budgets
Organize	Division of labor
	Delegation of authority
	Span of control
Control	Establish standards
	Monitor
	Reward/punish
	Job Appraisal
Lead	Motivate
	Develop
	Encourage

Mostly all software development projects involve a team effort to elicit user requirements, document and analyze those requirements, implement a solution, test and verify the solution, deliver the completed solution to the customer, and lastly, maintain that system. Collaborative efforts have been demonstrated to aid analysts by improving the problem solving process (Wilson et al., 1993) (Sabin & Sabin, 1994) (Nosek, 1998) (Selvin, 1999). The benefits of the collaborative development environment are also supported by Deek's research (Deek, 1999) (Deek et al., 2001).

During these collaborative activities it is possible for both business conditions and technologies to change. These changes, coupled with the possibility that the customer often changes the system requirements and basic functionality, has further complicated the software development process. As a result, the software industry is plagued by cost overruns, late deliveries, poor reliability, and user dissatisfaction (Abdel-Hamid & Madnick, 1991). According to a Standish Group International report, approximately one third of all information system projects failed, more than half completed over budget, and

only 16% were completed on time and within budget (Cafasso, 1994). Some causes of project failures are technology related, however the more predominate causes are communication issues and ineffective leadership. Block's analysis outlines twelve categories to classify most software system failures and they are summarized in Table 2.2 (Block, 1983).

Table 2.2 Categories of System Failures

Failure	Cause	Result
Resource	Conflicts of people, time, and project scope due to insufficient personnel.	Incorrect systems with poor reliability, difficulty with maintenance, and dissatisfied users.
Requirement	Poor specification of requirements.	Leads to developing the wrong system with many changes in requirements downstream.
Goal	Inadequate statement of system goals by management.	Leads to developing the wrong system by leading to requirement failures.
Technique	Failure to use effective software development approaches, such as structured analysis and design.	Causes inadequate requirements specification, poor reliability, high maintenance costs, scheduling, and budget problems.
User Contact	Inability to communicate with the system user.	Causes inadequate requirements specification, and poor preparation for accepting and using the information system.
Organizational	Poor organizational structure, lack of leadership, or excessive span of control.	Leads to poor coordination of tasks, schedule delays, and inconsistent quality.
Technology	Failure of hardware / software to meet specifications; failure of the vendor to deliver on time, or unreliable products.	Cause schedule delays, poor reliability, maintenance problems, and dissatisfied customers.

Table 2.2 Categories of System Failures (Continued)

Failure	Cause	Result
Size	When projects are too large, their complexity pushes the organization's systems development capabilities beyond reasonable limits.	Caused by insufficient resources, inadequate requirements specifications, simplistic project control, poor use of methodology, and poor organizational structure.
People Management	Lack of effort, stifled creativity, and antagonistic attitudes cause failures.	Time delays and budget overruns occur, project specifications are poor, and the system is difficult to maintain.
Methodology	Failure to perform the activities needed, while unnecessary activities are performed.	This type of failure can lead to any of the consequences of system failure.
Planning and Control	Caused by vague assignments, inadequate project management, and tracking tools.	Work assignments may overlap, deliverables may be poorly defined, and poor communication may result.
Personality	These are caused by people clashes.	Passive cooperation and covert resistance, with possible acts of vengeance.

The purpose of project management is to provide focus for using resources to achieve a specific objective within the constraints of time, cost, and performance. To help achieve objectives, managers should understand that the predominate causes of project failures are due to the human elements of communication issues and ineffective leadership. Since the four disciplines of management are geared towards effective people management skills, management can directly address the human aspects of project failure. By addressing these human aspects component of the software project management process, a manager can help reduce their effect on project failure and increase the likelihood of a successful software project.

2.3 Human Aspects

One way to measure the success of a software project is through good client relations as the ultimate measure of a project's success (Jurison, 1999). User involvement helps acquire a good working relationship with the client. This is achieved with an open line of communications with the client. Barki and Hartwick, Lederer, Baroudi, and Saleem have all done research in the area of user involvement on software system acceptance. They have all determined that user involvement enhances system usage, user satisfaction, and ultimately system acceptance (Barki & Hartwick, 1994) (Lederer et al., 1998) (Baroudi et al., 1986) (Saleem, 1996). Thus, effective communication skills can facilitate the process leading to a successful project.

An individual with effective communication and people skills aids all members of the team by acting as a facilitator between the software development team and the client during all phases of the software development process. In addition, many organizations face a staffing challenge. A manager also needs to facilitate communications among the members of the development team in an effort to keep everyone on the same page. Information technology workers of all types have realized that their skills are highly transferable to different companies and industries (Gill & Pidduck, 2001). Employees experiencing work exhaustion or job burnout have higher intentions of leaving their job (Moore, 2000). To help prevent high turnover, a manager needs to find ways to retain their professional work staff.

By understanding and utilizing the Herzberg motivation hygiene model of management, a manager should be able to leverage this model and apply it to help establish and retain a highly motivated project team.

Table 2.3 Herzberg's Hygiene and Motivational Factors

Hygiene or Dissatisfiers	Working Conditions
	Policies and administrative practices
	Salary and Benefits
	Supervision
	Status
	Job security
	Fellow workers
	Personal life
Motivators or Satisfiers	Recognition
	Achievement
	Advancement
	Growth
	Responsibility
	Job challenge

The model states employee motivation is achieved with challenging enjoyable work where achievement, growth, responsibility and advancement are encouraged and recognized (Herzberg et al., 1959). This theory addresses the concerns of job security, salary, working conditions, benefits, and others, which can affect an employee's productivity. By addressing these concerns, managers can eliminate possible sources of dissatisfaction, thereby increasing the chances of greater productivity.

Another way to improve the software development team is by utilizing participative management. The participatory management style approach allows various team members decision-making authority. The individual making a particular decision is the team expert in that particular area. The manager remains actively involved by facilitating the process. Field studies have shown that participation management has a positive influence on productivity and job satisfaction (Chung & Guinan, 1994). These are just some of the ways in which human aspects can be directly addressed and managed for the benefit of the individuals, software development team, and organization.

Information technology has also brought about change in the fundamental structure of the team concept. Communication technology has afforded organizations the ability to create virtual teams. Consequently, these team types have added a new level of complexity to team dynamics. Similar to the traditional approach, a virtual software team is a group of people whom work together guided by a common purpose to develop a software system. The difference, with respect to the traditional team, is that personnel and resources may be distributed over many different geographical locations. This brings new challenges to project management with regards to the social system. Organizational inertia, politics, and culture can all have major impacts on current efforts to develop effective virtual team dynamics (Haywood, 1998). Some of the key challenges facing the virtual team manager are building a cohesive team, keeping the synergy flowing, and monitoring the work of team members (Paré & Dubé, 1999).

Therefore, a software project team manager must consider both the technical and human components of the software development process during the life-cycle of a software project. The remainder of this paper addresses the technical element of the software development process, namely the analysis phase.

2.4 Analysis Framework

The analysis phase examines what the software system should do before subsequent phases decide how it is actually realized. The analysis determines the scope of the software system by completely describing what is to be created and techniques exist to specify a system. This section presents a new analysis framework to help describe various techniques used during the specification process. The goal of developing the

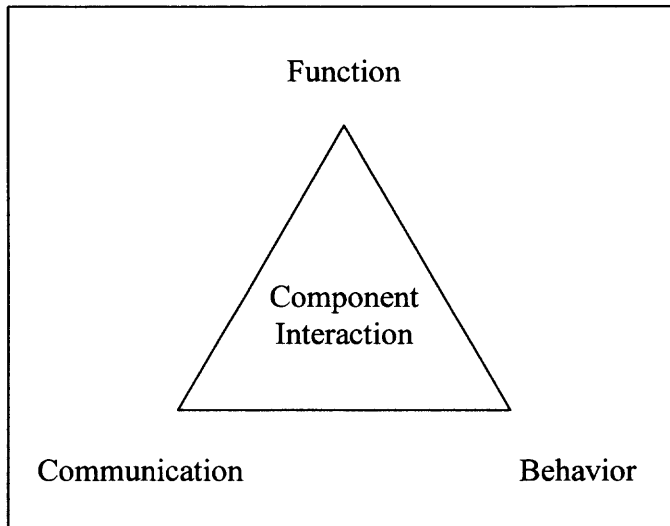
framework is to make it general enough to accommodate different analysis techniques and provide the ability to describe each technique in a systematic, repeatable method. Developing the framework in this manner makes it possible to describe issues, limitations, problems, and opportunities with various analysis techniques.

The analysis framework is based upon the component-oriented approach to the software system development process. Component-based development or component-based software engineering is a re-use based approach to software systems development (Sommerville, 2001). A component is an independent entity that provides services and may be described at different levels of abstraction. The analysis phase views the system at a high level of abstraction and identifies major system components. These components interact with each other to create a useful system function that describes what is to be done. The goal of the analysis is to describe the software system in its entirety by decomposing it into its relevant high-level abstract components. It is possible to describe a software system at different levels of abstraction and detail. The aggregate of all of these components comprise the software system. A more refined description yields a more detail-oriented software system description. There are many techniques and methods available to perform this software system decomposition and the intent of this paper is to compare and contrast the variety of techniques using a common analysis framework.

The analysis framework presented views a software system as a collection of components that interact with each other to accomplish a process leading to a particular goal. The framework is based on the concept of component-based interactions. A component interaction is described by function, communication, and behavior

characteristics (Wieringa, 1998) and can best be viewed graphically as depicted in Figure 2.2.

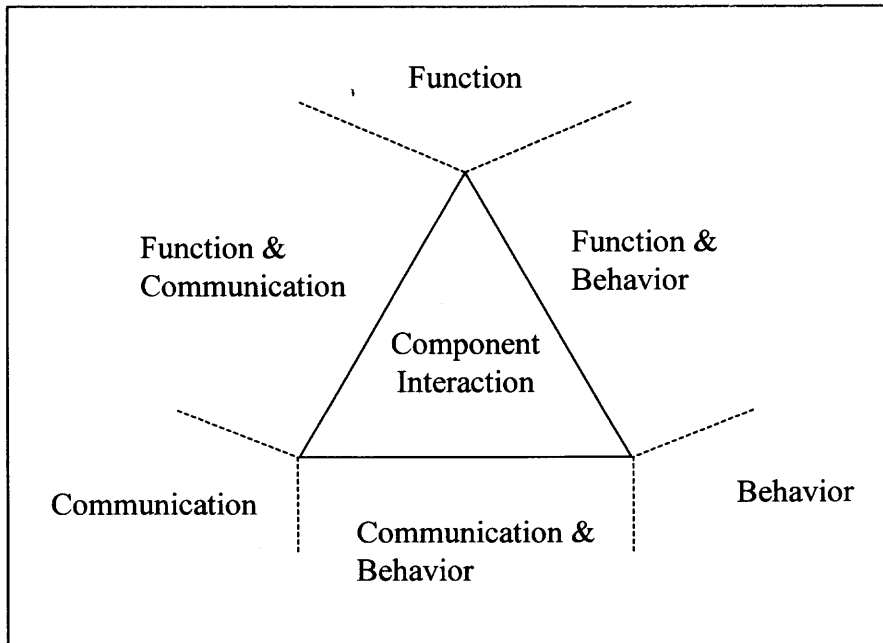
Figure 2.2 Component Interaction Characteristics



The function characteristic describes the actions or functionality of a component interaction. Communication characteristics describe how information is exchanged. Behavior, the third characteristic, describes how the component responds to an event. Function, communication, and behavior characteristics can act either independently or dependently with each other to describe a component interaction.

In my framework, I incorporate the dependent nature of the three characteristics to yield three additional characteristics depicted in Figure 2.3. These additional characteristics result from the intersection of the three primary characteristics.

Figure 2.3 Component Interaction Analysis Framework



A component interaction may consist of one or more functions that can communicate and share information with each other. The way functions communicate with each other helps to capture relationships among the functions. The fifth characteristic, function & behavior, represents the time-ordered behavior of a function. Lastly, the third combination, communication & behavior, describes the sharing of information with respect to time. These six characteristics comprise a component interaction and form the framework that will be used for comparison to describe each analysis technique. The literature does not explicitly categorize systems using all these characteristics in a framework. Wieringa's paper is an original approach that provides a framework to compare analysis techniques. I extend the framework and explicitly add three additional characteristics that describe the analysis technique in terms of its interdependence among the original three characteristics. These six characteristics are used to describe the static and dynamic features of an analysis technique. Therefore, the

varying analysis techniques will be described in terms of how well these characteristics are addressed.

Analysis techniques can be compared and contrasted using the characteristics of function, communication, or behavior either independently or in combination. For example, to determine the effectiveness of how an analysis technique identifies major software system functionality, the function characteristic can be utilized. This approach permits comparisons between techniques to be made. A second example involves the determination of how well the analysis technique identified the scope of work utilizing a re-usability approach. In this case, the analysis technique should focus on the function and communication characteristics of the framework since re-usability is most successful when major system functionality can be encapsulated into a component that communicates through an interface.

Both of these examples show how the framework could be used to compare and contrast concepts utilizing various analysis techniques. This framework allows the analysis techniques to be compared and contrasted in a variety of ways utilizing the function, communication, and behavior system properties in either an independent or dependent manner.

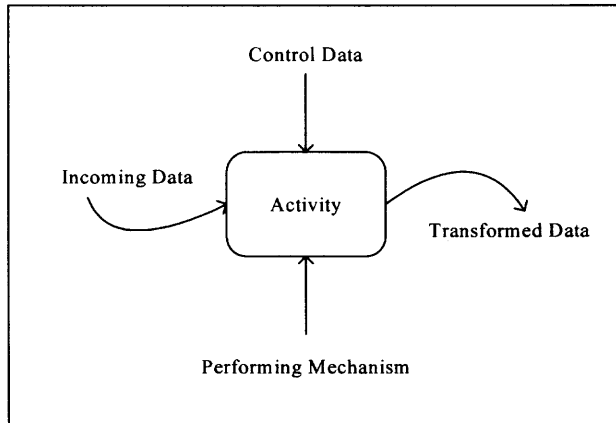
2.5 Structured Analysis

Structured analysis is a methodology that aids the practitioner during the analysis phase of the software system development life-cycle. Structured analysis is a process-oriented system definition approach to the description of the software system in a top-down fashion. The top-down approach decomposes the software system in a leveled manner

whereby each level provides more details until a primitive, atomic level is reached. This is a conceptual decomposition as opposed to a physical decomposition. A conceptual decomposition partitions the software system in terms of components that correspond to domain entities. In contrast, a physical decomposition is defined in terms of the actual software system components and is realized during the implementation phase. The conceptual decomposition is a way to make the demands of external functionality explicit without yet worrying about implementation decisions (Wieringa, 1998). Therefore, the conceptual decomposition is a top-down leveled approach that organizes views of the software system into a hierarchical structure. This structure defines the software system based on the system functionality and behavior, emphasizing both data and control flow. The structured analysis perspective is to generate a detailed, logical description of tasks and operations by focusing on the control flow and data processing of information.

Within the context of structured analysis, there are many prominent variants to analyze information flows. These variants suggest different ways to approach analysis in a structured manner, sharing the common goal of improving the understanding of the software system. Ross developed the “Structured Analysis and Design Technique” (SADT), which begins the process of analysis by determining the why and what of the software system components before progressing to the implementation or how phase. The SADT graphically depicts interactions of data and activities by utilizing activity diagrams whose constructs are depicted in the Figure 2.4.

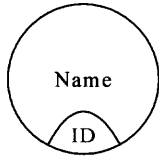

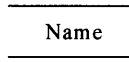
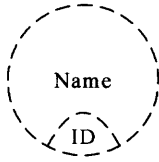


Figure 2.4 SADT Activity Diagram Constructs



There were other developments of structured analysis techniques around the same time period. For example, DeMarco's technique for analyzing information flow is based upon the process flow chart developed by Taylor and Gilbreth (Couger, J. 1973). These process flow charts graphically depict the movement of materials in a manufacturing or service-oriented capacity. The process flow chart is an abstraction and defines the key points and activities of the system processes. DeMarco and Yourdon extended the process flow chart concept to include the analysis component of the software engineering process. The constructs of their structured analysis technique are listed in Figure 2.5 (DeMarco & Yourdon, 1978). Gane and Sarson developed a method similar to DeMarco's process and data flow oriented-technique but it focuses more on the data view by emphasizing the identification of the data components of the software system (Gane & Sarson, 1979). To this end, their technique utilizes data access diagrams to describe the contents of the software system data stores. These data access diagrams depict the entities and links of a data store. This data centric technique builds from Chen's unified view of data concept (Chen, 1976). Chen identifies major data components of the system by utilizing the characteristics of entities, attributes, and relationships. The characteristics of

these components are captured in an entity-relationship-attribute (ERA) model. These entity relationship (E/R) diagrams have become the basic building blocks to database design techniques and model the characteristics of the database system to be designed.

Figure 2.5 DeMarco - Yourdon Constructs

Construct Name	Construct Symbol
Data Process	
Data Flow	
Data Store	
Control Process	
Control Flow	
External Entity	

Models and structural analysis techniques can be used to produce software system specifications that describe the software system. All versions of structured analysis

utilize a top-down decomposition approach to develop conceptual abstractions that lead to concrete software components. These structural analysis techniques can produce software system specifications by utilizing the constructs previously described. The specifications consist of various diagrams depicting the systems processes and data flow in both a static and dynamic nature. Traditional structured analysis of business-oriented software systems utilizes data flow diagrams, data dictionary, mini-specifications, and structured walkthrough components to identify the requirements (Svoboda, 1990).

2.5.1 Data Flow Diagrams

A data flow diagram (DFD) is a labeled directed graph in which the nodes represent functions and the edges data flows between functions (Wieringa, 1998). These diagrams show the way in which data is processed and describe the overall behavior of the software system. The DFD specifies the data and control processes of a software system utilizing the constructs described in the previous section. A data process describes the data as it progresses through a process, while a control process describes the behavior of the process. Both the data and control perspectives help to identify the requirements of the software system.

These diagrams are created to represent different levels of the software system. The highest and most abstract view of the system is called the data context diagram (DCD). This context level data flow diagram depicts the software system as a collection of external entities or sub-systems that represent major system functionality. The DCD helps to show the overall scope of the software system by representing each sub-system as an independent entity. Identifying the data process interaction in finer detail then

further decomposes each sub-system. These subsequent decompositions are performed in a top-down hierarchical leveled approach. Each intermediate level shows more detail than its predecessor until a primitive level is reached. The processes, data stores, and data flows depicted in the data flow diagrams are from a functional viewpoint and the communications perspective is also addressed by the input and output data contained within the DFD. Each process, entity, and data store of the decomposition depicts the input and output data to the construct. This permits a hierarchical trace ability of information flow through all the diagrams. Therefore, the DFD component of structured analysis addresses the function and communication aspects of the proposed comparative analysis framework. In addition, there are variants of the DFD that represent varying software system states of activity. These types of diagrams represent the dynamic state of the machine and model the behavior of the software system in response to internal or external events (Sommerville, 2001). Consequently, these state diagrams identify the software system dynamics. Thus, the various types of data flow diagrams depict both the functional aspects and system dynamics of the software system. These diagrams graphically depict both data and control flow utilizing pictorial constructs. To further enhance the meaning of these diagrams, structured analysis techniques produce a narrative textual description.

2.5.2 Data Dictionary

The data dictionary is a textual description of each data flow diagram and defines all the names, processes, data flows, and data stores. The data dictionary is a repository, manual or computer-based, containing information about the various data objects appearing on

each data flow diagram (Svoboda, 1990). It acts as a central location so that the textual description information is contained in a single location. Much of Hitchcock's research is based on the premise of processing information contained in a central location (Hitchcock, 1980). Hitchcock argues that data dictionaries form the basis of understanding the software system in its entirety. This description is comprehensive and each element identified on the data flow diagram should also have an entry in the data dictionary. These descriptions encompass all but the actual primitive processes.

2.5.3 Mini-specifications

The mini-specifications are also called the primitive process specifications and contain information concerning the actual primitive processes. A mini specification should describe the actual steps required to carry out the primitive process. These primitive processes represent a description that cannot be further decomposed into additional subcomponents. The four most common ways to realize a mini specification are: text-based narrative, decision tree, decision table, and procedure definition language (PDL) (Svoboda, 1990). Regardless of the method chosen, the focus of the description is on what steps the primitive process must perform. The ability to adequately describe what is to be performed aids in comprehending the functionality required by the software system. The creation of the mini-specifications is the last step of the structural analysis before the analysis is reviewed and critiqued in its entirety.

2.5.4 Structured Walkthrough

The inspection process or structured walkthrough review follows the completion of the structured analysis and its primary function is to verify the accuracy of the analysis. This review takes the form of a formal meeting in which the participants should consist of all the stack-holders of the proposed software system. This meeting allows all participants to verify that the analysis actually captured the problem that the proposed software system is to solve. Since one of the primary reasons why projects fail is because requirements analysis is not performed (Abdel-Hamid & Madnick, 1989), it is crucial that all participants reach consensus concerning the software system before design and implementation begins. Thus, the goal of the structured walkthrough review is to have all stack-holders understand and agree on the requirements of the software system.

2.5.5 Deficiencies

Although structured analysis aids in describing the requirements, it also has many shortcomings. These deficiencies encompass both the technical and human aspects concerning structural analysis. With respect to the human element, the process of structured analysis does not distinguish between the way people act and the way machines function and represents a functionalistic or system-structural approach (Astley & Van de Ven, 1983). Bansler describes how this highly structured technique treats workers as machines causing dissatisfaction to increase and motivation to decrease, resulting in the following deficiencies (Bansler & Bødker, 1993).

- Underrates the skills and ingenuity of the workers since problem-solving skills cannot be reduced to structured analysis rules.
- Ignores the significance of informal communication among workers to coordinate their tasks and to cooperate in group-problem solving.

- Underestimates the frequency and significance of errors and exceptions.
- Does not consider resistance from individuals or groups of users to the development and introduction of a new software system.
- Contains no concepts for modeling organizational units or resources.
- Offers no help in specifying relations of authority and responsibility.
- The role of the users in this process is passive.

In addition to structured analysis deficiencies concerning the human aspects, the following list outlines some of the technical deficiencies (Sommerville, 2001).

- They do not provide effective support for understanding or modeling non-functional system requirements.
- They do not usually include guidelines to help users decide whether or not a method is appropriate for a particular problem.
- They are not developed for the concept of re-use.
- They often produce too much documentation.
- The models are very detailed and users find them difficult to understand.

The aforementioned lists describe both the technical and human component deficiencies associated with structured analysis. Most of the literature dealing with structured analysis does not mention human aspects, only the technical aspects. However, the study of any information system topic should encompass both the technical and human component since they coexist and should not be considered mutually exclusive.

2.5.6 Summary

Structured analysis is a technically oriented technique that takes a functionalistic approach to problem solving. The approach utilizes constructs and rules to break down a problem in a highly structured top-down approach. In doing so, the goal of the analysis is to decompose the software system into its basic components utilizing a systematic method. Table 2.4 summarizes the structured analysis technique utilizing the characteristics of the analysis framework.

Table 2.4 Structured Analysis Framework Characteristics

Function Description	Explicitly realized through the use of functional decomposition.
Function & Communication Description	Depicted through data flow diagrams that show function and data flow.
Communication Description	Depicted through data flow diagrams that show data input and output.
Communication & Behavior Description	Depicted through state diagrams.
Behavior Description	Depicted through data and control flow.
Function & Behavior Description	Depicted through state transition diagrams.

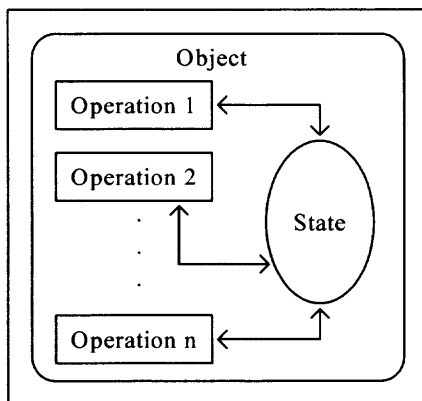
Structured analysis does address the six characteristics of the analysis framework discussed and the results are listed in Table 2.4. In particular, the system entities and relationships among them are identified by data flow diagrams. As the name implies the nature of the relationships among entities are data flow centric. These relationships are tightly coupled with corresponding functions thus making relationship identification dependent on function identification. This relationship identification is solely based upon the ability of the development team to identify system functions. However, many functions are not identified until the implementation phase causing many relationships to be missed during the analysis phase. This could lead to inadequate problem domain understanding and an incomplete analysis process.

2.6 Object-oriented Analysis

Object-oriented analysis (OOA) is a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain (Booch, 1994). With respect to software systems, OOA is a method that develops software engineering requirements and specifications utilizing an object model approach.

An object model represents the software system by providing a description of the major software components or objects comprising the system. An object is a real world concept or abstraction that represents a portion of the problem that is to be solved. An object is an entity that has a state and a set of operations that access the state and is depicted graphically in Figure 2.6.

Figure 2.6 Object Structure



The object is comprised of two sets of components: state information and operations. An object's state is defined by a set of attributes and the operations performed on that state are called methods. Consequently, the object model is a collection of interacting objects that maintain their own state and provide operations that permit access to this state information. These objects help to encapsulate an abstract concept into a self-contained unit. This unit or component-based approach provides object-oriented analysis powerful modeling techniques. Therefore, the principle behind object modeling is encapsulation and abstraction (Booch, 1996) (Rumbaugh, 1991). Booch defines a spectrum of abstraction for objects that closely model problem domain entities:

- Entity abstraction is an object that represents a useful model of a problem domain or solution-domain entity.

- Action abstraction provides a generalized set of operations, all of which perform the same kind of function.
- Virtual-Machine abstraction is an object that groups together control operations.
- Coincidental abstraction is an object that packages a set of operations that have no relation to each other.

These types of objects are the building blocks of the object-oriented paradigm, which incorporate the object-oriented strategy throughout the software development process. Sommerville breaks the object-oriented development process into three main components: analysis, design, and implementation (Sommerville, 2001).

- Object-oriented analysis develops an object-oriented model of the application domain.
- Object-oriented design develops an object-oriented model of a software system to implement the identified requirements.
- Object-oriented programming realizes a software design using an object-oriented programming language.

Each stage of the object-oriented development process uses the same notation, thereby eliminating transition gaps. These uniform principles apply throughout the software development process. Objects identified during the analysis phase map directly into the design and implementation phases. This similar notation dependency has both positive and negative aspects. Since objects encapsulate a portion of the problem to be solved, tracing requirements become easier since manipulation of object entities is a more natural approach to problem solving (Nerson, 1992). Conversely, if during the analysis phase the objects are incorrectly created, it negatively impacts the design and implementation phases. Ultimately this result in a final architecture that reflects the poor decisions made during the analysis phase.

The object-oriented analysis problem solving method differs from the structured analysis process-oriented method in two major respects (Bailin, 2000):

- The method in which a software system is portioned into subsystems and components.
- The way in which the interactions between these subsystems or components are described.

The object-oriented (OO) paradigm takes the data and procedure components, discussed in structured analysis, but de-emphasizes the procedures, stressing instead the encapsulation of data and procedural features together. A fundamental goal in defining objects is to group data items together with methods that read and write to these data items. This kind of grouping makes each object a cohesive set of methods and data thereby helping to encapsulate problem domain concepts into a collection of self-contained units.

Encapsulation and abstraction are the principles behind object-oriented data modeling encompassing the fundamental abstraction concepts of :

- Classification: Grouping entities that share common characteristics
- Generalization: Extracting from one or more objects the description of a more general object that captures the commonalities but suppresses the differences
- Aggregation: Treating a collection of objects as a single object
- Association: Considering set of member objects as an object
- Attribution: Identification of properties or attributes of an object

Therefore, encapsulation helps to decentralize object-oriented architectures resulting in software systems to be more understandable, reliable, and easier to maintain (Anderson, 1989) (Sun, 2002) (Booch, 1994) (Rumbaugh, 1991).

2.6.1 Characteristics

In addition to abstraction and encapsulation, there are several other characteristics common to all variants of object-oriented analysis methods. This section introduces the following terms:

- Class
- Inheritance
- Information hiding
- Polymorphism

A class is a way of organizing objects in terms of their similarities and differences (Bailin, 2000) (Booch, 1994) (Rumbaugh, 1991). An object class describes a group of objects that have the same attributes and behavior patterns. This grouping of objects supports the concept of abstraction and affords modeling the ability to generalize a real world concept, as a single class comprised of a collection of interacting objects. Operations that are shared by different objects can be written once for the class instead of once for each object. This fosters the concept of re-use. Software code re-use has been identified as the key to improving software development and productivity due to its trait of reducing product cycle time, which in turn reduces both cost and risk during the product development process (Arango, 1994).

Inheritance has become synonymous with code re-use within the object-oriented programming community (Rumbaugh, 1991). Inheritance defines a relationship among classes by grouping similar classes together to re-use common code. The predicate “is-a” is used to determine these relationships. Each member of a sub-class “is-a” member of the parent class and inherits characteristics from the parent class. These characteristics are methods and variables of the parent class, which are accessible to the sub-classes, thereby allowing these sub-classes to re-use code from the parent class. This “whole-part” or “part-of” relationship identification is aggregation. Aggregation is the “whole-part” or “part-of” relationship in which objects representing the individual components, when combined, represents the entire component.

Classes also have the ability to hide information from its sub-classes. This information could be some or all of the methods or attributes. However, the focus of the analysis phase is on what methods and attributes are needed to describe a problem domain concept and information hiding begins to enter the area of how the class is to be implemented. Polymorphism is also a concept that borders between the analysis and implementation phases. Polymorphism is the ability of classes related through inheritance to respond differently to the same method call due to late dynamic binding. This occurs at run-time and permits the same operation to take different forms in different classes. For the purpose of analysis considerations, it suffices to say that both information hiding and polymorphism aid to abstract and encapsulate a problem domain concept.

By understanding the concepts of abstraction, encapsulation, object, class, information hiding, inheritance, and polymorphism at the analysis phase, it helps to ensure their proper use and incorporation into the software system during the design and implementation phases. These concepts are used to develop the following four views in the description of the software system:

- Object Identification
- Object Communications
- Object Behavior
- Object Operations

2.6.2 Object Identification

One of the first steps in any object-oriented analysis is to identify all the objects of the problem domain. Object-oriented analysis identifies the types of objects that map into components of the application domain that is being modeled. To identify these

components, Booch adopted Abbott's method of object identification by the differentiation of noun phrases contained within the problem domain narrative description (Abbott, 1983). This is an intuitive process thereby making the process difficult to describe and document (Nerson, 1992) (Henderson-Sellers & Edwards, 1990) (Booch, 1994). Some guidelines exist to perform this noun decoupling and the goal is to define relationships between the software system components. This is a multi-step process whereby the problem domain is first decomposed into a collection of individual domains. These domains are self-contained and are comprised of a collection of sub-systems. These sub-systems are highly cohesive and loosely coupled and are two software quality characteristics that help to indicate a good analysis and design (Sommerville, 2001). The sub-systems are then further decomposed into a collection of interacting objects. The ability to describe a complex object as a structure of interacting simpler objects is a key benefit of the object-oriented approach (Bailin, 2000) and provides a static information model overview.

Shlaer and Mellor view this information model as a way to identify and capture objects, relationships, and attributes in an entity-relationship (E/R) diagram derived from Chen's view of data concept (Chen, 1976). Their extension improves on Chen's entity-relationship-attribute (ERA) diagram in two ways (Shlaer & Mellor, 1992):

- The information model helps to organize the entities, attributes, and relationships of the information model.
- The information model provides a rich set of graphical constructs to represent entities, attributes, and relationships. The graphical symbols provide a more efficient way to describe the information model by utilizing a unique set of symbols and identifiers.

The entity relationship diagram is a conceptual decomposition and represents the structure of the software system and is graphically depicted by class diagrams (Booch,

1994). These class diagrams depict the descriptive attributes and properties of an object. There are many variations to the class diagram concept, but all object-oriented analysis methods use the class diagram technique as part of their method. Some examples of the first generation class diagram variations are: object model (Rumbaugh, 1991) (Martin & Odell, 1995), information model (Shlaer & Mellor, 1992), object-relationship model (Embley et al., 1992), static object model (DeChampeaux et al., 1993), general semantic net (Firesmith, 1993), and object/class model (Henderson-Sellers & Edwards, 1994).

The various first generation class diagrams each provide an independent set of rules and graphical constructs to depict the entities, relationships, and attributes of the software system objects. These class diagrams depict the classification, generalization, aggregation, association, and attribution of the software system problem domain. Regardless of the graphical constructs used, the creation of the class diagram should follow a seven-step specification process (Booch, 1994) (Rumbaugh, 1991) (Martin & Odell, 1995) (Shlaer & Mellor, 1992) (Embley et al., 1992) (DeChampeaux et al., 1993) (Firesmith, 1993) (Henderson-Sellers & Edwards, 1994) (Coad & Yourdon, 1990) (Jacobson et al., 1992):

- Identification of key problem domain objects
- Distinguish between active and passive objects
- Establish data flows between active objects
- Decompose objects into sub-objects
- Check for new objects
- Group functions under new objects
- Assign new objects to appropriate domains

These steps are performed during the analysis phase to create the class diagram and represent the architecture's conceptual decomposition of the software system problem domain.

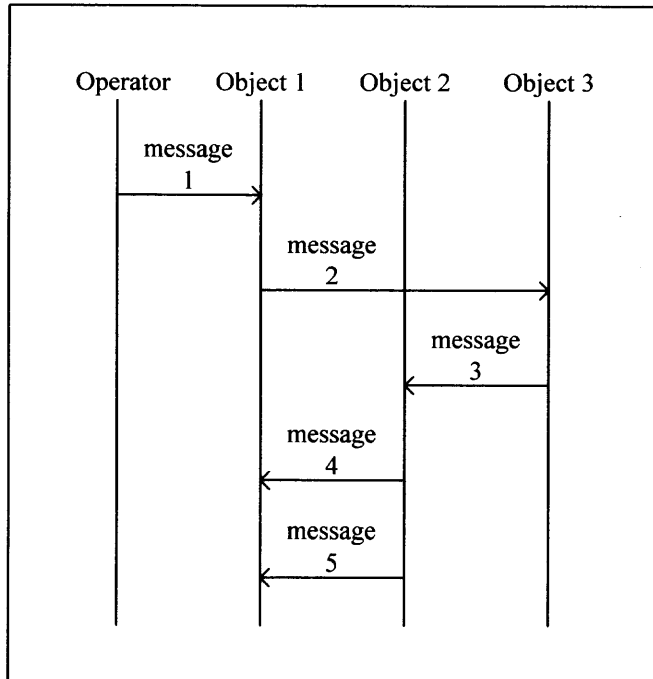
2.6.3 Object Communications

Others have extended the use of the class diagram to depict object communications (Coad & Yourdon, 1990) (Graham, 1994) (Selic et al., 1994). In addition to representing the software system's conceptual decomposition, their implementation of class diagrams also depicts the operations or services performed by the objects. However, the literature reviewed did not indicate that this was a widely accepted approach due to its tendency to clutter the class diagram. Instead the literature did indicate that the bulk of the object-oriented analysis practitioners prefer to utilize two kinds of diagrams to represent communications:

- Communication Sequence: Sequence and Collaboration Diagrams
- Communication Only: Object Communication and Interaction Diagrams

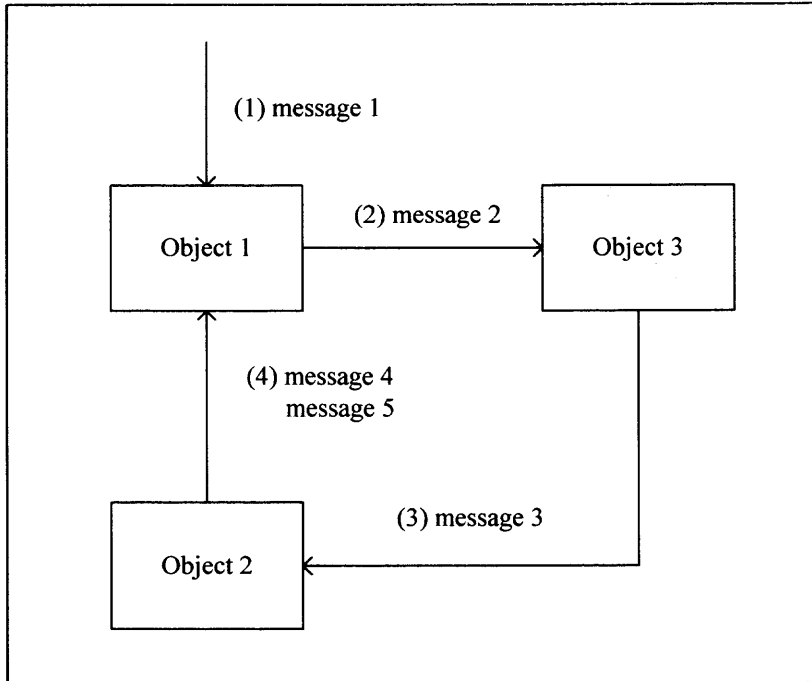
Most of the object-oriented analyses techniques prefer to describe object communications utilizing sequence and collaboration diagrams. A sequence diagram represents a particular sequence of messages exchanged between entities or objects and is used to show the flow of functionality or process flow. In the telecommunications community, sequence diagrams have been standardized as message sequence charts (ITU, 1994). The Objectory technique introduced sequence diagrams into object-oriented modeling (Jacobson et al., 1992). Vertical lines represent entities or objects and the horizontal arrows represent the messages (Booch, 1994) (Rumbaugh, 1991) (Jacobson et al., 1992) and are depicted in Figure 2.7.

Figure 2.7 Sequence Diagram



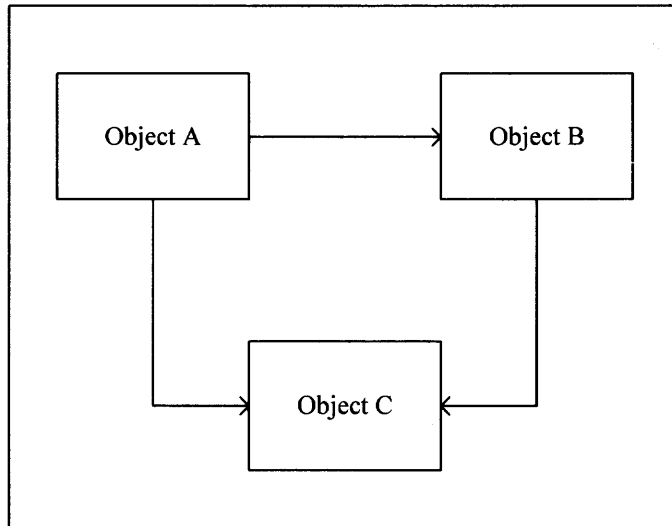
Sequence diagrams focus on events as opposed to operations, and help to define the boundaries of the software system. These diagrams also aid to view the passing of messages in order, thereby enhancing the dynamic description of the software system with respect to time. These timing diagrams describe a sequence of interactions over time (Firesmith, 1993).

Whereas sequence diagrams illustrate the objects and interactions over time, collaboration diagrams show the objects and interactions using a different set of graphical constructs. A collaboration diagram is a directed graph in which the nodes represent entities or objects and the edges represent communications. The edges are numbered to represent the ordering of communications in time (Wirfs-Brock, 1990). Therefore, collaboration diagrams show the same information as sequence diagrams, but utilize a directed graph form.

Figure 2.8 Collaboration Diagram

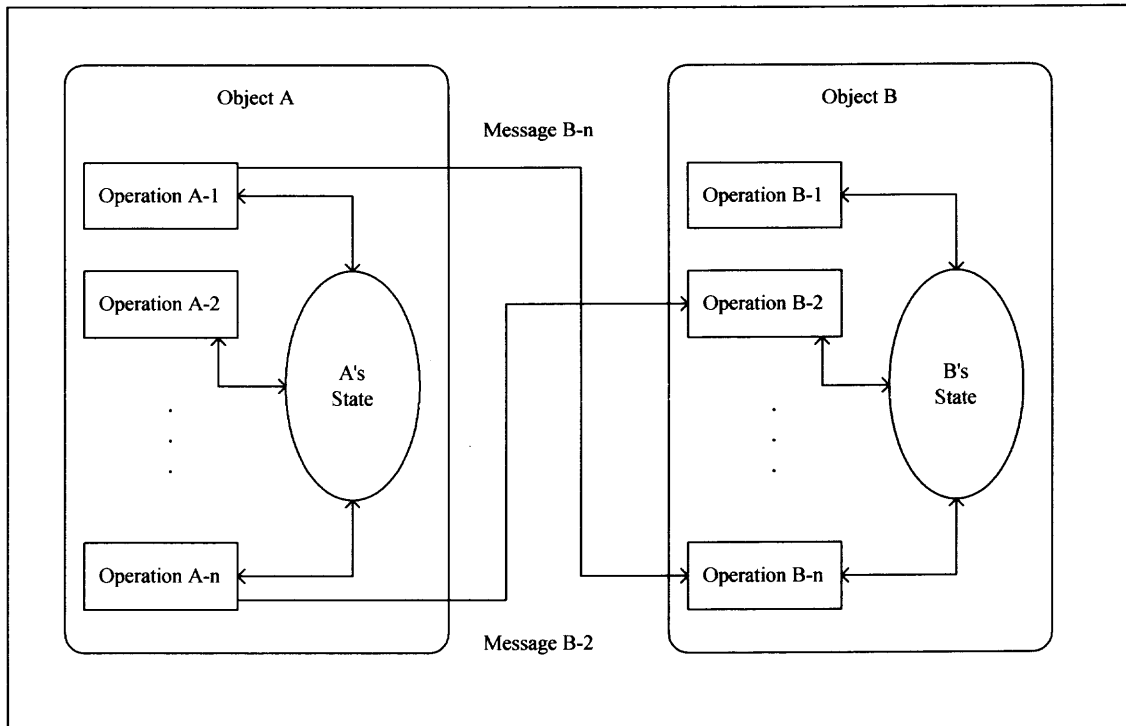
This directed graph form offers a different view of the software system when compared to the sequence diagram. This view helps to see the overall processing and sequencing between objects.

The alternate type of diagram shows communications only. These object communication and interaction diagrams show communications without indicating any sequence. These diagrams are directed graphs in which the nodes represent object classes and the edges represent object communications. These types of diagrams show only the communication and not the behavior of the software system and are depicted in Figure 2.9.

Figure 2.9 Object Interaction Diagram (Less Detail)

Different levels of detail can be displayed on these object interaction diagrams. An interaction diagram with the most detail will display the messages passed between objects by depicting the methods. Shlaer and Mellor provide the object communication model (OCM) and the object access model (OAM) to describe object communications in detail. The OCM shows the messages sent and received by the state machines of the objects and the OAM shows accesses to object data stores made by object processes (Shlaer & Mellor, 1992). Both of these models summarize all of the interaction among the objects in the software system and provide a system-wide overview of object interaction.

Figure 2.10 Object Interaction Diagram (More Detail)



The directed graph technique has been embraced by many others to represent communications between components in which the nodes represent activities or processes and the edges depict the communications (Martin & Odell, 1995) (Embley et al., 1992) (DeChampeaux et al., 1993) (Firesmith, 1993). However, object communications are usually coupled with its behavior and other diagrams exist to depict object behavior.

2.6.4 Object Behavior

Object behavior is how an object acts and reacts to its state changes and message passing. These aspects of the software system deal with time and are represented by a dynamic model. The major dynamic modeling concepts are events, which represent external stimuli; and states, which represent values of objects (Rumbaugh, 1991). The dynamic model is comprised of multiple state diagrams and shows the pattern of activity for the

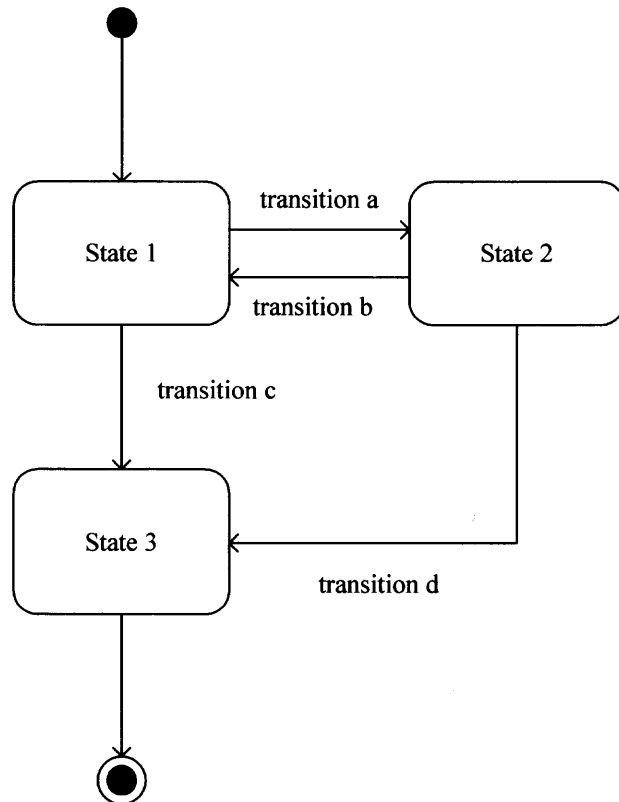
entire software system. A state diagram shows the life-cycle of an object, from the time it is created until it is destroyed and depicts its pattern of events, states, and state transitions. This collection of software system state diagrams depicts the dynamic behavior of the objects comprising the software system. Objects are typically modeled as finite state-machines, which means that there are only a finite number of states in which the object can exist (Bailin, 2000). A state diagram or state machine can be specified in either a graphical or tabular form. The literature did not indicate that the tabular form, represented by a transition matrix, was utilized in object-oriented analysis techniques. Instead practitioners of object-oriented analysis prefer to use state-transition diagrams using state-chart notation to model the dynamic behavior of the software system (Booch, 1994) (Rumbaugh, 1991) (Shlaer & Mellor, 1992) (Embley et al., 1992) (DeChampeaux et al., 1993) (Firesmith, 1993) (Henderson-Sellers & Edwards, 1994) (Jacobson et al., 1992) (Graham, 1994) (Selic, 1994).

There is a great deal of similarity between the different notations of state-transition diagrams but all notations use a directed graph form. The nodes of the directed graph represent states and the edges represent transitions. For example, the Mealy state machine distinguishes input events from output actions. These input events are associated with a transition, meaning that an input event occurrence triggers the transition (Wieringa, 1998). This is depicted in the state-transition diagram by separating the input events from the output events of a transition by a solid horizontal line or by a slash. In contrast, the Moore state machine shows that outputs are associated with states. This means that actions are performed upon entry of a state and all transitions entering a state

will generate the same output. Both the Mealy and Moore state machine models represent the same information, but utilize a different approach using similar notation.

State-charts incorporate the information provided by the Mealy and Moore state machine into its directed graph as well as additional type of information. A state-chart uses the same notation as the other state machines in which the nodes represent states and the edges represent state transitions. In addition to these features, actions can be specified along transitions, upon entry and exits of states. The state-chart specifies the state sequence caused by an event sequence.

Figure 2.11 State-chart Diagram



State-chart diagrams have become the standard way to model the dynamic behavior of the software system and have been integrated into the Unified Process developed by Booch, Rumbaugh, and Jacobson.

2.6.5 Object Operations

The events depicted in the state diagrams trigger object operations. These operations are identified to respond to the data and control flow of the software system. An object operation performs a service and are actions associated with a state. Each action is specified by means of executable code associated with the object's state (Bailin, 2000). The executable code or function is executed when the software system is in some state and may leave the software system in a different state. The effect of the function can be specified textually in two ways, declaratively or imperatively (Wieringa, 2000). Declarative specifications describe pre and post-condition states. A pre-condition is a condition on the input and system state at the start of executing the function and a post-condition is a condition on the output and the system state after the execution of the function. Shlaer summarizes the process by stating that an action may read or update the state of any object in the subsystem, it may create an object, and it may send an event to any object or to an entity outside the subsystem (Shlaer & Mellor, 1992).

Some object-oriented analysis techniques represent these actions by an action dataflow model (Rumbaugh, 1991) (Shlaer & Mellor, 1992) (DeChampeaux et al., 1993) (Coleman et al., 1992). The purpose of the action dataflow model is to specify the data processing performed by an action and is comprised of the data store and data processing components. An action dataflow diagram is a directed graph in which the nodes represent objects and the edges represent attributes that are read or written by an operation. In addition, the diagram depicts the data-flows between the operations. These flows may cross object boundaries and helps to improve the definition of the relationship between the functional model and the object model (Wieringa, 1998). Although the

techniques used differ slightly, the goal is to define a functional decomposition for each action. This approach attempts to connect the static model to the dynamic model by providing dynamic action decomposition.

In contrast, other object-oriented analysis techniques specify object operations informally by means of text-based object specifications (Booch, 1994) (Jacobson et al., 1992) (Henderson-Sellers & Edwards, 1994). The elements contained in the specifications may be written in a given implementation language or in narrative form. The literature did not indicate that there was a standardized way to specify the operations in text format. The literature does indicate, however, that the Unified Process does not support the action dataflow diagrams to depict object operations but instead incorporates operations on the class diagram (Booch et al., 1998) (Boggs & Boggs, 2002).

2.6.6 Summary

Object-oriented analysis views the software system as a collection of interacting objects. These objects are part of the object-oriented model that features an approach based on abstraction, encapsulation, classification, and inheritance. This approach identifies objects and encapsulates data and operations together. In addition, similar objects are grouped together to form classes. This type of decomposition of a problem into objects and classes depends on judgment and the nature of the problem. There is no one correct representation. All of the variants of object-oriented analysis methods discussed were developed to represent or view the software system in terms of object identification, communication, behavior, and operations. These views help to outline the static architectural structure of the software system as well as the system's dynamic behavior.

Booch, Rumbaugh, and Jacobson have unified their object-oriented analysis and design techniques and provide a method to describe the development of a software system's static and dynamic architecture in detail. The systems development life-cycle is titled the Rational Unified Process (RUP) and uses an iterative method development process as opposed to the traditional sequential development process offered by the Waterfall model (Boehm, 1988) (Sommerville, 2001) (Booch et al., 1998). The iterative development process treats the project as a series of small Waterfalls. Each one is designed to encompass a subset of the entire project. Each subset or project piece is large enough to mark the completion of an integral component of the project, but small enough to minimize the need for backtracking. The RUP process provides specific process steps, guidelines, and workflows that can be used during the development process. These steps have helped support the iterative approach to the development of a system using object-oriented techniques. The RUP life-cycle approach helps to break a problem into smaller more manageable pieces, which in turn makes these components more re-usable, maintainable, and extensible.

Table 2.5 summarizes the object-oriented analysis technique utilizing the characteristics of the analysis framework.

Table 2.5 Object-oriented Analysis Framework Characteristics

Function Description	Explicitly realized through the use of object identification.
Function & Communication Description	Depicted through class diagrams that depict attributes and properties of objects.
Communication Description	Depicted through sequence, collaboration and interaction diagrams.
Communication & Behavior Description	Depicted through state-transition diagrams.
Behavior Description	Depicted through state-transition diagrams.
Function and Behavior Description	Depicted through state-transition diagrams.

Object-oriented analysis does address the six characteristics of the analysis framework discussed and the results are listed in Table 2.5. In particular, the system objects or entities are identified through the use of object identification techniques. The concepts of classification, generalization, aggregation, association, and attribution are key principles behind object-oriented modeling. Object identification is an intuitive process in which entities and relationships are determined by examining the noun phrases contained within the problem domain narrative description. Once objects are identified, they are grouped together to form a class, which is a collection of interacting objects. The interaction among the objects represents the relationship structure. As with object identification, relationships among the objects comprising a class as well as the relationships among different classes are also determined by examining the narrative description of the problem domain. Determining the entity and relationship structure of a problem domain using object-oriented analysis is an implicit process. As with structured analysis, an implicit process can cause many relationships to be missed during the analysis phase. This could lead to an inadequate problem domain understanding and an incomplete analysis process. However, the object-oriented paradigm graphically represents entities and relationships. These visual aids help to provide communication tools among members of the development team.

2.7 Use-case Analysis

Use-case analysis is a scenario-based technique for requirements elicitation and was first introduced in the Objectory object-oriented analysis methodology (Jacobson et al., 1992).

Use-case analysis is the process of capturing requirements from the user's point of view

and helps describe what functionality is contained within the system. This perspective is not implementation-oriented but stresses instead what the user expects from the system.

Use-cases and actors define the scope of the system being designed. A use-case illustrates how someone might use the system and represents a piece of system functionality. An actor is anyone or anything that interacts with the system being designed. A primary actor is one having a goal requiring the assistance of the system and a secondary actor is one from which the system needs assistance to satisfy that goal (Cockburn, 1997). There are three types of actors (Boggs & Boggs, 2002).

- System users
- Other systems that interact with the system being built, an external application
- Time triggered events that interact with the system at a particular interval

Each kind of actor uses the system in different ways and each way of using the system is called a use-case. A use-case represents a portion of functionality that the system will provide and are a way of specifying system functionality in a manageable way. Boggs provides four questions to determine the functionality each actor expects from the system (Boggs & Boggs, 2002):

- What will the actor need to do with the system?
- Will the actor need to maintain any information (create, read, update, delete)?
- Does the actor need to inform the system about any external events?
- Does the system need to notify the actor about certain changes or events?

In general, end-user needs identify the issues and features associated with the problem that is to be solved and help to define the problem domain. Use-case analysis extends the problem description by mapping how end-users interact with the system to realize the various features.

2.7.1 Identification

Each use-case identifies the actors involved in an interaction and names the type of interaction (Sommerville, 2001). In addition, use-cases describe the sequences of system interactions and actors in response to some event. Therefore, the theoretical sum or collection of all use-cases represents all of the possible interactions that will be represented in the system requirements for a particular interaction. However, it is not likely, nor necessary to capture all possible interactions comprising a functional component. Use-cases are designed to capture requirements from the user's point of view and helps describe what functionality is contained within the system. To identify the use-cases, one should review all documentation of the proposed project and list the scenarios that are fundamental to the system's operation. The collection of scenarios describes the system functions of the application. Each scenario is then analyzed using storyboarding techniques similar to practices in the television and movie industry (Zahniseer, 1990). This technique involves identifying the objects that participate in the scenario. The first pass outlines the use-cases primary behavior. Subsequent passes consider exceptional conditions and secondary system behaviors. Rumbaugh suggests capturing use-cases at the beginning of the analysis using the following approach (Rumbaugh, 1994):

- Identify the boundary of the proposed application.
- Identify the objects just outside the boundary that interact directly with the system objects.
- Classify the outside objects by the roles that they play in the application. Each role defines an actor.
- Make a list of actors. State the purpose of each actor in using the system.
- For each actor, think of the fundamentally different ways in which the actor uses the system. Each way of using the system is a use-case.
- Group different scenarios into the same use-case if they appear to be variations of the same theme.

- Determine the interaction sequences. For each use-case, identify the event from the actor that initiates the use-case. Determine if there are preconditions that must be true before the use-case can begin. Determine the logical conclusion of the transaction.
- Write a prose description of the use-case. Identify the sequence of interactions that occur in a normal transaction together with the system operations that are invoked.
- Consider all the exceptions that can occur in handling a transaction and specify how they affect the use-case.
- Look for common fragments among different use-cases and factor them out into base cases and additions. Determine if the additions are optional or mandatory, and specify where they go in the main sequence.

Kentworthy has refined the process to eight steps (Kentworthy, 1997):

Table 2.6 Use-case Recipe

Step 1	Identify who is going to directly use the system. These are the actors.
Step 2	Pick one of these actors.
Step 3	Define what the actor wants to do with the system. Each of these things that the actor wants to do with the system becomes a use-case.
Step 4	For each of these use-cases, decide on the most usual course when that actor is using the system. What normally happens is considered the basic course.
Step 5	Describe that basic course in the description for the use-case.
Step 6	Once the basic course is described, consider the alternatives and add those as extending use-cases.
Step 7	Review each use-case description against the descriptions of the other use-cases. Notice any glaring commonality? Extract those out as your common course use-cases.
Step 8	Repeat steps 2 – 7 for each actor.

The following questions help to determine if all the use-cases have been identified

(Jacobson et al., 1992) (Boggs & Boggs, 2002):

- Is each functional requirement in at least one use-case?
- Have you considered how each stakeholder will be using the system?
- What information will each stakeholder be providing for the system?
- What information will each stakeholder be receiving from the system?
- Have you considered maintenance issues?
- Have you identified all of the external systems with which the system will need to interact?
- What information will each external system be providing to the system?
- What information will each external system be receiving from the system?

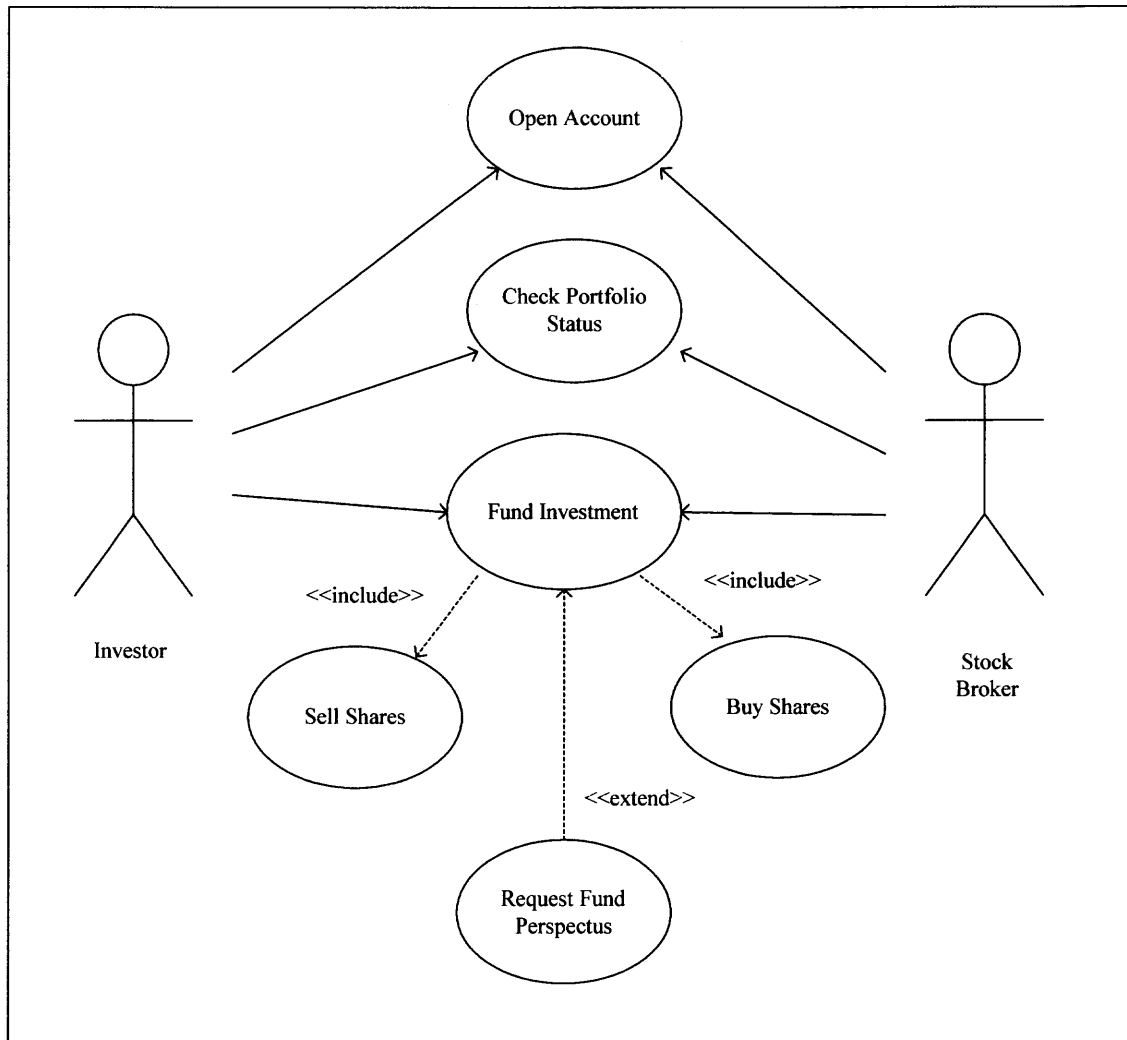
The use-case analysis approach focuses on what the system should do, not how it is realized. Use-cases are written as natural language text descriptions that express what happens from the user's point of view. Use-case diagrams supplement textual descriptions by providing graphical representations of the use-cases. The details of how the system works internally are not contained in either the textual or graphical representation of the use-case. Instead, the details are addressed during the implementation phase. Therefore, use-case analysis represents an end user perspective of the system to be designed. This approach helps the customer understand the functionality that will be provided by the system. In addition, use-case analysis provides a means of describing the scope of the system to be designed at the start of the development life-cycle process.

2.7.2 Structuring a Use-case

Use-case analysis has been incorporated into the Unified Modeling Language (UML), which provides association, generalization, include, and extend relationships to structure use-cases (Booch et al., 1998). An association relationship depicts the relationship between an actor and a use-case. A generalized relationship shows the commonality between actors or use-cases. The include relationship is similar to the concept of inheritance, whereby one use-case can re-use the functionality provided by another use-case. In contrast, an extend relationship captures a variant of the base use-case and extends its functionality to capture specialized behavior. Figure 2.12 depicts a use-case utilizing the UML notation. Solid lines with arrow indicate an association relationship between an actor and a use-case. A dotted line with arrow represents either an includes

or extends relationship and is specified by the phrases `<<include>>` or `<<exclude>>` respectfully.

Figure 2.12 Use-case (UML Notation)



Although use-cases are an integral part of the UML, there is no template for writing a use-case textual description. Coleman proposes the following use-case template (Coleman, 1998):

Table 2.7 Coleman's Use-case Template

Use-case	Each use-case should have a unique name suggesting its purpose. The name should express what happens when the use-case is performed. It is convenient to include a reference number to indicate how it relates to other use-cases. The name field should also contain the creation and modification history of the use-case preceded by the keyword history.
Description	Each use-case should describe the main goals. The description should list the sources for the requirements, preceded by the keyword sources.
Actors	List the actors involved in the use-case.
Assumptions	Each assumption should be stated as in a declarative manner, as a statement that evaluates to true or false. If an assumption is false, then it is unspecified what the use-case will do. The fewer assumptions that a use-case has, then the more robust it is. Use-case extensions can be used to specify behavior when an assumption is false.
Steps	The interactions between the system and actors are structured into one or more steps, which are expressed in natural language. A step has the form <sequence number><interaction> Conditional statements can be used to express alternate paths through the use-case.
Variations	Further detail about a step may be given by listing any variations on the manner or mode in which an event occurs.
Non-Functional	The non-functional requirements are listed in the form <keyword>:<requirement> Non-functional keywords include; performance, reliability, fault tolerance, frequency, and priority.
Issues	List of issues that remain to be resolved.

Both the graphical and textual ways to represent use-cases are designed to identify the possible interactions between the system components and the actors to realize a particular goal. Although goals are a normal part of software engineering, there is no formal methodology to capture goals (Cockburn, 2000). Use-case analysis attempts to address the void by its approach to goal-oriented analysis.

2.7.3 Deficiencies

One of the difficulties with use-cases is how to capture goals at the appropriate level of abstraction. Inexperienced teams may generate numerous use-cases to represent all

aspects of system functionality. However, it can be argued that use-cases were not designed to capture all types of system descriptions. Instead they were designed to describe what functionality is contained within the system from the user's point of view (Booch et al., 1998). Use-cases represent a high-level view of what the system will do, without worrying about the details of how it is realized.

Another area of confusion is about whether or not a use-case is a scenario on its own or, as suggested by Fowler, a use-case encapsulates a set of scenarios where each scenario is a single thread through the use-case (Fowler & Scott, 1997). In this case, there would be a scenario for the normal interaction plus scenarios for each possible exception. Cockburn describes a scenario as a sequence of interactions happening under certain conditions, to achieve the primary actor's goal, and having a particular result to that goal (Cockburn, 1997). The literature does support the definition of a use-case as a collection of possible scenarios between the system and the actors to achieve some goal. Although the scenarios may exist under different conditions, they are grouped together since they share a common goal.

A confusing part of use-case analysis is how to write the textual narrative since there is not a formal structure. Coleman, Jacobson, and Rumbaugh provide similar semi-formal templates in many of their writings, but by design the process has remained informal. This affords people the opportunity to use their own prose and lets them communicate the way they think best for the desired project (Cockburn, 1997). This permits use-case analysis to remain flexible in its approach to describe what functionality is contained within the system from the user's point of view.

2.7.4 Summary

Use-case analysis helps to capture the functional requirements of the product being developed from the user's point of view. Use-cases and actors help to define the scope of the system to be built by identifying the sequences of system interactions. The collection of possible sequences is what constitutes system behavior. Use-cases describe the goals to be achieved by identifying the possible interactions between the actors and system components from a user-centric perspective. Use-cases are implementation independent stressing instead goal-oriented functionality. The goal-oriented nature of use-case analysis helps to define the problem domain by describing the scope of the system. This approach to problem description helps to track the project by goals. In addition, use-case analysis increases the chances that the system being developed meets user needs and expectations thereby increasing user satisfaction and acceptance.

Table 2.8 summarizes the use-case analysis technique utilizing the characteristics of the analysis framework.

Table 2.8 Use-case Analysis Framework Characteristics

Function Description	Determined from the user's point of view using use-case scenarios and actors.
Function & Communication Description	Depicted on use-case diagrams using association, generalization, includes and extends notation.
Communication Description	Described in a prose description of each use-case.
Communication & Behavior Description	Described in a prose description of each use-case.
Behavior Description	Described in a prose description of each use-case.
Function & Behavior Description	Described in a prose description of each use-case.

Use-case analysis does address the six characteristics of the analysis framework discussed and the results are listed in Table 2.8. In particular, the system entities are determined by capturing functionality from the user's perspective. In addition, the use-

case diagrams capture the relationship structure among the entities utilizing generalization, association, include and extend concepts. The generation of use-case diagrams is an explicit process using well-defined process steps and desired user-determined system functionality. Use-case diagrams provide excellent communication tools among members of the development team and end-users. In addition, use-case diagrams help to provide a deeper understanding of the problem domain as seen by the end-users by explicitly identifying its relationship structure. However, use-case analysis falls short when describing the dynamic nature of the system. Much of the system's dynamic behavior is described in written format. In contrast, both structured and object-oriented analysis techniques provide various types of state transition diagrams to explicitly capture the dynamic nature of the problem domain. Since both structured and object-oriented analysis do not address the problem domain from an end-user perspective, use-case analysis can be incorporated at the front end of these software development life-cycle paradigms. This is exactly the approach taken by Rational in their creation of the Rational Unified Process (RUP). The RUP uses Jacobson's use-case analysis as its first step to identify desired user functionality expected from the proposed system. Subsequent steps of the software development process are realized using the object-oriented paradigm. This approach has been very successful and incorporated into the development process by many software engineering companies.

2.8 Domain Analysis

Domain analysis is the process by which information used in developing systems in a domain is identified, captured, and organized with the purpose of making it re-usable

when creating new systems (Prieto-Diaz, 1991). The ability to develop software within a particular application domain relies on a comprehensive understanding of that domain. To understand the domain, an analyst gathers all relevant information from various sources and synthesizes it into a domain model. Problem solving skills are essential to understand what is to be solved. Problem solving is best realized utilizing a highly structured strategic method (Svoboda, 1990). This is accomplished through eliciting, verifying, and formalizing software requirements and specifications (Iscoe, 1993). A successful usable domain analysis is dependent upon how well the process is performed and should answer the fundamental questions:

- Who is the customer?
- Who are the stakeholders in the domain?
- What are the software assets comprising the total software solution for that domain?
- What is the domain boundary?
- How can existing system assets be leveraged to the current domain?

Two examples of domain analysis methods to answer the above questions are Feature Oriented Domain Analysis (FODA) and Organizational Domain Modeling (ODM) techniques. The focus of both techniques is to:

- Gather domain information to define the scope and boundary of the domain
- Describe the data and variables needed to support the system
- Identify the system's relationships
- Determine if existing system component assets can be re-used
- Develop re-usable assets

Both techniques are designed to improve maintainability, understandability, and re-usability, thereby improving the software development life-cycle. These quality measures allow improvements utilizing domain analysis to be measured.

2.8.1 Feature Oriented Domain Analysis

Feature Oriented Domain Analysis (FODA) was one of the first domain analysis methods and is based upon identifying the features of a class of systems. FODA is based upon abstraction and refinement to develop re-usable software assets by abstracting away variables that differentiate related applications. Consequently, applications comprising the domain exist at a high level within the overall system. Specific applications are refined from these generalized assets to develop the specialization needed for the specific application domain.

FODA defines three basic activities: context analysis, domain modeling, and architectural modeling (Kang, 1990). Context analysis defines the domain scope by determining the relationships that exist within the domain. Secondly, the domain-modeling phase creates a model by analyzing both similarities and differences of the applications comprising the domain. Lastly, the architectural modeling phase provides the software solution for the applications contained within the domain. It defines the process for allocating the features, functions, and data objects defined in the domain models to the processes and modules (Cohen, 1992). Simply stated, this feature oriented analysis technique facilitates the re-use of a software asset. Its process steps are geared to this end. A re-usable software asset helps to reduce product cycle time, cost, and risk during the product development process.

2.8.2 Organization Domain Modeling

Organizational domain modeling (ODM) integrates organizational and strategic aspects of domain planning, domain modeling, architectural engineering, and asset base

engineering (Simos, 1995). Simos developed a configurable domain analysis process model, useful for diverse organizations and domains, and amendable to integration with a variety of implementation technologies. The ODM process model is comprised of a descriptive and prescriptive phase. Descriptive modeling uses knowledge and past experience from existing models and extends it to the new system. This is realized by documenting the similarities in system structure and functionality of those systems. The prescriptive phase binds the decisions and commitments concerning both the system functionality and implementation. The most common problem with the ODM approach deals with its two-phase approach and its potential inability to limit overlap of responsibility and scope between these phases. The primary benefit of utilizing the ODM process is to gain a better understanding of the problem domain. A complete understanding of problem domain scope aids in the development of the software solution.

2.8.3 Standardizing Product Re-use

Another feature of domain analysis is that it also facilitates the concept of re-use by examining existing solutions and leveraging these known solutions to new problems (Lung & Urban, 1995). The ability to transfer knowledge from an existing solution to a new problem helps construct re-usable solutions. Consequently, domain analysis emphasizes both re-usability and the study of pertinent information to the solution of a new problem. One advantage of re-usability is that less time is needed to develop a component. Consequently, needed delivery time can be reduced. Re-usability is an established principle in engineering and applies previous successful solutions to new problems. In addition to reducing product cycle time, the strategy is aimed at reducing

both cost and risk during the product development process (Arango, 1994). Thus, re-use has been identified as the key to improving software development and productivity.

Since the 1980s, software engineering has attempted to create a software component industry based on a repository model of components that could be accessed by many different kinds of applications (Favaro, 1997). Favaro notes that components are best produced in the context of a domain such as a banking or telecommunications system instead of a fit-all generic component development approach. These more specific domain architectures facilitate the development of more re-usable domain components. However, not all applications benefit from the re-usable component-oriented approach to software production. Table 2.9 shows an analysis of typical domains according to criterion (Favaro, 1997).

Table 2.9 Domain and Re-useability Criterion

Application Domain	Rate of Introduction of New Products	Degree of Standardization in the Domain	Software Criticality (safety, reliability)	Rate of Technical Evolution
Financial Services	High	Low	Medium	Medium/High
Telecommunications	High	High	High	High
Medical	Medium	Medium	High	High
Multimedia	High	Low	Low/Medium	High
Command and Control	Low	Medium/High	High	Medium
Computer-Aided Design	Medium	High	Low/Medium	Medium
Entertainment Electronics	High	Low	Low	Medium
Geographic Information Systems	Medium	Low	Low/Medium	Medium
Robotics and Industrial Automation	Medium	High	High	Medium
Office Automation	High	Low/Medium	Low	Medium

The values of Table 2.9 describe the degree of the column's attribute in these domains. An application domain with high marks in the degree of standardization category indicates that this type of application domain may benefit through domain analysis more so than others with low marks. Favaro suggests that high marks generally indicate that the technology is well established. In contrast, the application domain that would benefit the least through domain analysis is multimedia. The multimedia application domain has not yet reached a sufficient level of maturity. The high rate of technical evolution has impeded the degree of standardization. To help reach a sufficient level of maturity, the application domain needs to be understood from various perspectives and levels. The Unified Modeling Language (UML) (Booch et al., 1998) is a tool that aids in the development of domain models utilizing process diagrams that outline the domain from different perspectives. UML's widespread acceptance into the software business community has helped foster the acceptance of the domain analysis component approach.

2.8.4 Summary

Domain analysis stresses the creation of a software system utilizing a re-usable component-based approach. Components are referred to as assets, which domain analysis attempts to leverage from existing systems to be incorporated into future software systems. Incorporating re-usable software assets into a new system reduces product cycle time, cost, and risk during the product development process. Table 2.10 summarizes the domain analysis technique utilizing the characteristics of the analysis framework.

Table 2.10 Domain Analysis Framework Characteristics

Function Description	Determined through domain modeling, which describes system functionality through explicit feature description.
Function & Communication Description	Context analysis determines the relationships that exist within the problem domain.
Communication Description	Architectural modeling describes data communications among system processes.
Communication & Behavior Description	Addressed through the architectural model.
Behavior Description	Addressed through the architectural model.
Function and Behavior Description	Addressed through the architectural model.

Domain analysis does address the six characteristics of the analysis framework discussed and the results are listed in Table 2.10. In particular, the system entities are determined by capturing system functionality. In addition, the relationship structure is also determined by examining the desired functionality of the system.

Similar to use-case analysis, domain analysis does consider the end-users an integral component of the process and elicits their input at the onset of the system development process. In addition, proponents of domain analysis argue that the ability to develop software within a particular application domain relies on a comprehensive understanding of that domain. Incorporating end-user requirements into the process at the start of a project does help in the successful acceptance of the delivered system. Therefore, domain analysis endeavors to provide a comprehensive application domain analysis prior to subsequent phases of the system development life-cycle.

2.9 Requirements Analysis

Requirements analysis, as it relates to software projects, is the process of studying, determining and documenting user needs and expectations of the software system to be

designed that solves a particular problem. The process is referred to as requirements engineering and entails feasibility studies, elicitation, specification, and validation process steps. The process generates software requirement documents that capture what is to be implemented by fully describing the software system's functionality, performance, design constraints, and quality attributes. Precisely documenting what to build helps to reduce uncertainty and equivocality (Daft & Lengel, 1986). Determining and documenting the requirements of an information system, is arguably the key to developing successful information systems (Vessey & Conger, 1994). Not getting the correct final software system requirements at the project onset is largely responsible for the cost and schedule overruns plaguing the information system development process (Boehm, 1981) (Abdel-Hamid & Madnick, 1991) (Vessey & Conger, 1994). Table 2.11 shows that the earlier in the development process an error occurs and the later the error is detected, the more expensive it is to correct (Faulk, 2000).

Table 2.11 Relative Cost to Repair a Software Error in Different Stages

Stage	Relative Repair Cost
Requirements	1-2
Design	5
Coding	10
Unit Test	20
System Test	50
Maintenance	200

Thus, performing software requirements analysis at the project onset helps to identify and correct problems early. Consequently, the relative repair cost is low and reduces the chances of project cost overruns.

Much of the literature describes the requirements analysis process as three sub-processes and compares it to an engineering methodology (Thayer & Dorfman, 2000)

(Loucopoulos & Karakostas, 1995) (Macaulay, 1996) (Vessey & Conger, 1994) (Sommerville, 2001):

- Elicitation
- Specification
- Validation & Verification

Each sub-process addresses the problem definition aspects from different angles during the requirements creation process to collectively describe the software system to be built. The software specification documents generated, as a result of the analysis, captures the user needs and describes the operation of the proposed software system. The software system description is generally comprised of three types of documents (Leffingwell, 1999) (Thayer & Dorfman, 2000) (Sommerville, 2001).

- Functional Requirements
- Non-functional Requirements
- Design Constraints

Functional requirements describe what the software system should provide and how it should behave. In contrast, non-functional requirements are not concerned with specific functionality delivered by the system. Instead non-functional requirements relate to system properties such as reliability, response time, memory space, portability, maintainability, ease-of-use, robustness, security, and re-usability (Mylopoulos et al., 1999) (Sommerville, 2001). Design constraints describe the requirements that are specific to characteristics of the problem domain that do not easily categorize into the other two types of documents.

The hardest single part of building a software system is determining and documenting precisely what to build (Brooks, 1987). The difficulties of documenting and specifying software requirements are primarily due to human problem-solving

limitations (Davis, 1982). Davis also points out that these limitations are because human beings have a limited ability to process information. Much of the time humans leave or filter out information to prevent information overload (Davis & Olson, 1985). To help include all the available information, methodologies have been developed to provide a systematic repeatable approach to the description and development of software requirements and systems (Demarco, 1978) (Gane & Sarson, 1979) (Jackson, 1983) (Rumbaugh,1991) (Booch, 1994). Methodologies help to structure the problem and solution domain into a collection of smaller sub-problems. These sub-problems are then individually described and eventually implemented. The aggregate of all the components, describe the entire problem domain. This approach helps to divide large complex problems into smaller, more manageable components. In addition, complexity is reduced since the amount of information that a sub-component must consider is also reduced. Therefore, the documentation of sub-problems help achieve a goal of requirements analysis, namely to understand and capture what is to be solved in a component-oriented manner using all available information.

2.9.1 Eliciting Requirements

Requirements elicitation is the process of identifying the application domain by determining the desired software system functionality. This activity should involve many different kinds of people that have a stake in the system being built. These stakeholders work together to define and scope out the application domain. Each participant is a stakeholder and represents a different interest in the project. These interests are dependent upon the role the individual performs. Therefore, the elicitation process

should include all people that are either directly involved with the project or indirectly affected. Once the stakeholders are identified, the process enters the problem domain description phase. The description is realized either in an independent or a team-oriented collaborative manner. Gause points out that many organizations reinforce a negative image of cooperative work, encouraging instead competition among employees by such devices as individual achievement awards (Gause & Weinberg, 1989). However, software development projects should not be realized alone and need the diversity and ability that a collaborative team approach can provide. Much of the literature supports the concept that groups generate more and better solutions to identifying, describing, and solving a problem (Baroudi et al., 1986) (DeSanctis & Gallupe, 1987) (Gallupe et al., 1988) (Connolly et al., 1990). The general consensus is that problem definition is likely to be more complete when realized by participation in a collaborative team environment.

The literature identifies many different techniques that are possible to elicit requirements of a computer or information system in both a group team and single person team approach (Goguen & Linde, 1993) (Rumbaugh, 1994) (Kotonya & Sommerville, 1996) (Sommerville, 2001). Some examples of a non-group approach are:

- Introspection
- Questionnaires
- Interviews
- Protocol Analysis
- Ethnography

The introspection technique attempts to elicit requirements of the desired computer or information system by having the development team members individually imagine the system they want. Thus, many perspectives and interpretations will result from introspection. Many viewpoints help to identify all aspects of the problem domain,

but these do not necessarily reflect the needs of the end-users of the system. The literature does not consider this technique a practical way to elicit requirements due to its apparent lack of end-user involvement.

Similar to introspection, questionnaires and interviews attempt to elicit requirements by asking questions in a non-group oriented fashion. Questions are presented to individuals in either a written or verbal format, and the answers recorded. Although this is a systematic process Suchman argues that these approaches lead to multiple interpretations in both the questions and the answers (Suchman & Jordon, 1990). To reduce misinterpretations, the interview technique can be extended to permit dialog between the interviewer and interviewee. However, the literature indicates that the interview process usually involves assumptions concerning the interaction among participants. Goguen strongly argues that assumptions and misinterpretations that can result from questionnaires and interviews make this technique impractical to elicit computer and information systems requirements (Goguen & Linde, 1993).

Protocol analysis is a process in which a person performing a task does so aloud while his or her thought processes are observed and recorded. This represents direct verbalization of specific cognitive processes (Simon, 1984). This technique helps to understand an individual's approach to problem solving. Therein lies the problem; it is not a team-oriented approach. The project team consists of many different kinds of people operating in different roles. Some of these individuals have knowledge about the business and organizational needs, while others have technical knowledge. The process of eliciting requirements from these different types of people possessing different types of knowledge is a social endeavor requiring group communications. Protocol analysis is

an individual process, not a social interaction method. Although protocol analysis has greatly influenced cognitive psychology it is inappropriate for the requirements analysis and elicitation process (Goguen & Linde, 1983).

Ethnography is an observational technique to develop an understanding of work processes through observing as opposed to interviews in which people act differently than they say. These observations help to understand the social and organizational requirements. This is a time consuming process but contains much depth by helping to identify implicit system requirements. Implicit requirements are more easily identified by a third party observer because workers often perform tasks out of habit and rarely consider these tasks as part of their work process. The literature describes many ethnographic studies that showed a worker's actual work practices were much more detailed and complex than these individuals were able to describe (Suchman, 1983) (Simonsen & Kensing, 1997) (Myers, 1999). Thus, ethnography has been shown to be very effective at discovering the way in which people actually work rather than the way in which process definitions say they should work (Sommerville, 2001). In software system project development, ethnography may be best suited to determine how to modify an existing system to make it more effective as opposed to at the onset of a completely new project.

The aforementioned techniques are non-collaborative and tend to be inaccurate, inflexible, costly, time consuming, and do not represent natural interactions among people. The literature does indicate that more effective techniques to elicit requirements of a computer or information system are collaborative by design. Three such techniques are:

- Rapid Application Development (RAD) Focus Groups
- Viewpoint-oriented Requirements Definition (VORD)
- Use-case Scenarios

Rapid application development focus groups are a type of group interview that permits interactions among people to discuss requirements of the desired system. These interactions are both formal and informal depending on the organization performing the RAD and can reduce the cycle-time by 50% in the definition and development of the system (Engler, 1996). RAD is an iterative process that employs interactive sessions of developer, customer and end-user to identify and define the requirements of the desired system. This collaborative effort affords the project team the ability to openly discuss the system that is to be built. The successes of RAD have been attributed to the inclusion of the end-users during the system definition process (Gonzales & Wolf, 1996).

In addition to end-users, viewpoint-oriented elicitation also includes other stakeholders in the viewpoint-oriented requirements definition (VORD) approach to identify requirements. A viewpoint represents a certain perspective of the system. The VORD process recognizes the different viewpoints provided by the different stakeholders and incorporates their perspectives into the requirements specification process (Sommerville, 2001). The VORD method includes four steps (Kotonya & Sommerville, 1996) (Sommerville, 2001):

- Viewpoint Identification
- Viewpoint Structuring
- Viewpoint Determination
- Viewpoint-System Mapping

These four steps utilize a highly structured text-based method to document a viewpoint's attributes, events, and scenarios during brainstorming sessions. These viewpoints are documented using viewpoint and service templates that help to identify

both the functional and non-functional requirements. Nuseibeh uses a similar approach but augments the process by allowing incomplete scenarios to exist during elicitation (Nuseibeh et al., 1994). As the process of determining the desired system to be built continues, the incomplete scenarios are more rigorously defined, specified, and resolved by final specification.

Use-case analysis is a scenario-based technique for requirements elicitation. Use-cases capture requirements from the user's point of view and helps describe what functionality is contained within the system. The literature indicates that use-case analysis is the most widely accepted method to eliciting requirements for software systems. Section 2.7 of this paper describes use-case analysis in detail.

These three techniques are more successful than other techniques because they adopt a team-oriented user-inclusive strategy. It is important to include the eventual end-users of the system since they make or break the product. Therefore, the team should consist of the customers, developers, and end-users. However, it should be noted that the inclusion of customers, developers and end-user stakeholders during the elicitation process does have six primary difficulties (Sommerville, 2001).

- Stakeholders often do not know what they want from the computer system except in the most general terms, finding it difficult to articulate what they want from the system.
- Stakeholders make unrealistic demands because they are unaware of the cost of their requests.
- Stakeholders in a system naturally express requirements in their own terms and with implicit knowledge of their own work. Others must try to understand these terms and relate them to the application domain.
- Different stakeholders have different requirements and may express them in different ways. Requirement engineers have to discover all potential sources of requirements and discover commonalities and conflict.
- Political factors may influence the requirements of the system. These may come from managers who demand specific system requirements because these allow them to increase their influence in the organization.

- The economic and business environment is dynamic, which can lead to inevitable changes during the process. The importance of particular requirements may change. New requirements may emerge from new stakeholders who were not originally consulted.

Regardless of these difficulties, the literature suggests that the most effective way to elicit requirements is utilizing a team-oriented user-inclusive strategy.

2.9.2 Documenting Requirements

During the software system elicitation process, software requirements must be captured in a written format to help stakeholders clarify the operational needs of the software system. A document, known as the concept of operations document (ConOps), provides a well-defined operational concept definition of system goals, missions, functions, and components. Thayer lists the essential elements to be included in a ConOps document as (Thayer & Dorfman, 2000):

- Description of current system or situation
- Description of needs that motivate development of a new system or the modification of an existing system
- Modes of operation
- User classes and characteristics
- Operational features
- Priorities among operational features
- Operational scenarios for each operational mode and class of user
- Limitations
- Impact analysis

The ConOps document represents a bridge between the description of the user needs and the technical specifications of the software system specification process (Thayer & Dorfman, 2000). Both the U.S. Department of Defense and the IEEE support the creation of the ConOps document when developing or modifying a software system.

The document serves as a framework to guide the analysis and provides the foundation document for all subsequent system development activities.

The literature indicates that the ConOps document can be developed anytime during the system life-cycle but is most beneficial at the beginning of the software development process (IEEE, 1998) (Faulk, 2000) (Thayer & Dorfman, 2000). Developing the document at the onset of the development process affords all parties involved the opportunity to repeatedly review and revise the document until all stakeholders agree on the content. This iterative process helps bring to the surface many viewpoints, needs, wants, and scenarios that might otherwise be overlooked. In addition, the creation of formal specifications forces a detailed system analysis that could reveal errors and inconsistencies. This error detection is perhaps the most powerful argument for developing a formal system specification (Hall, 1990).

Developers utilize the ConOps document to create a Software Requirements Specification (SRS). The SRS describes exactly what is to be built by capturing the software solution. This represents a transition from the problem analysis to the technical analysis. Faulk outlines the roles of the SRS document (Faulk, 2000).

- Customers: Provide a contractual basis for the software project
- Managers: Provide a basis for scheduling and measuring progress
- Designers: Provide a basis for what to design to
- Coders: Provide tangible outputs that must be produced
- Quality Assurance: Provide a basis for test planning and verification
- Marketing: Provide a basis for marketing plans and advertising

A properly written SRS satisfies both the semantic and packaging properties (Faulk, 2000). An SRS satisfies the semantic properties if it is complete, implementation independent, unambiguous, precise, and verifiable. To satisfy the packaging properties, the SRS must be readable, modifiable, and organized for reference and review. There is

not a specific industry-wide system specification standard. The literature indicates that the IEEE Std. 830-1998 is widely accepted at documenting the SRS (IEEE, 1998) (Faulk, 2000) (Thayer & Dorfman, 2000) (Sommerville, 2001). The IEEE Std 830-1998 format is as follows:

1. Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, Acronyms, and Abbreviations
 - 1.4 References
 - 1.5 Overview
2. Overall Description
 - 2.1 Product Perspective
 - 2.2 Product Functions
 - 2.3 Constraints
 - 2.4 Assumptions and Dependencies
3. Specific Requirements
 - 3.1 External Interface Requirements
 - 3.1.1 User Interfaces
 - 3.1.2 Hardware Interfaces
 - 3.1.3 Software Interfaces
 - 3.1.4 Communications Interfaces
 - 3.2 Functional Requirements
 - ... Description Organized by Functional Requirements ...

OR

 - 3.2 Classes / Objects
 - ... Description Organized by Objects ...

OR

 - 3.2 System Features
 - ... Description Organized by Features ...
 - 3.3 Performance Requirements
 - 3.4 Design Constraints
 - 3.5 Software System Attributes
 - 3.6 Other Requirements

Regardless of the format chosen for the SRS, the information provided by the aforementioned IEEE Std 830–1998 should be contained within the requirements document describing the system to be built.

2.9.3 Verifying and Validating Requirements

Verification and validation (V&V) is the process that determines if the software conforms to the specifications and meets the needs of the customer. Verification involves checking that the software conforms to the specifications. Validation checks that the software meets the expectations of the users (Sommerville, 2001). Boehm expresses this subtle difference as follows (Boehm, 1981):

- Verification: Are we building the product right?
- Validation: Are we building the right product?

The V&V process of system checking is realized using two techniques, namely software documentation inspection and software system testing. The major software V&V activities are outlined in Table 2.12 (Wallace & Ippolito, 1996).

Table 2.12 Major Software Verification and Validation Activities

Activity	Tasks
Software V&V Management	Planning Monitoring Evaluating Results Reporting
Software Requirements V&V	Review of Concept Documentation Traceability Analysis Software Requirements Evaluation Interface Analysis Initial Planning for Software Integration Test Reporting
Software Design V&V	Traceability Analysis Software Design Evaluation Interface Analysis Initial Planning for Unit Test Initial Planning for Software Integration Test Reporting
Code V&V	Traceability Analysis Code Evaluation Interface Analysis Completion of Unit Test Preparation Reporting

Table 2.12 Major Software Verification and Validation Activities (Continued)

Activity	Tasks
Unit Test	Unit Test Execution Reporting
Software Integration Test	Completion of Software Integration Test Preparation Execution of Software Integration Tests Reporting
Software System Test	Completion of Software System Test Preparation Execution of Software System Tests Reporting
Software Installation Test	Installation Configuration Audit Reporting
Software Operation and Maintenance V&V	Impact of Change Analysis Repeat Management V&V Repeat Technical V&V Activities

Static, dynamic, and formal verification techniques can be performed to fulfill the requirements of the V&V process activities (Gause & Weinberg, 1989) (Wallace & Ippolito, 2000) (Sommerville, 2001).

- Static analysis reviews the structure of the software product prior to its execution. The analysis is performed on the requirements documents, design documents, and source code utilizing review and inspection oriented techniques.
- Dynamic analysis detects errors by executing the software and testing the actual outputs against the expected outputs as outlined in the SRS documents.
- Formal analysis is the use of rigorous mathematical techniques to analyze the algorithms of a software solution (Sommerville, 2001).

The software V&V process should begin at the onset of the project and continue throughout the entire software development life-cycle process. Prior to software construction, the V&V process should examine preliminary documents such as the ConOps document to help determine if the system to be built is feasible. Subsequently, the V&V process examines the SRS to ensure that the requirements are complete, consistent, accurate, readable, and testable. The approach helps to find errors in the software requirements before the software implementation phase. Also, the software

requirements analysis conducted is necessary in order to develop relevant test plans. This early error detection helps to reduce cost overruns, late deliveries, poor reliability, and user dissatisfaction (Abdel-Hamid & Madnick, 1991). However, the predominant V&V technique is software testing. Software testing involves executing the software product and examining its operational behavior. Testing is used to find discrepancies between the software program and the corresponding specifications and is referred to as defect testing. In contrast, statistical testing determines the software's performance and reliability by noting the number of system failures (Musa & Ackerman, 1989) (Butler & Finelli, 1991).

Test management organizes the testing process and utilizes test plans to determine the objectives for unit, integration, and system testing. Table 2.13 outlines the structure of a software test plan (Frewin & Hatton, 1986) (Sommerville, 2001):

Table 2.13 Software Test Plan Structure

The Testing Process	A description of the major phases of the testing process.
Requirements Traceability	Testing should be planned so that all end-users requirements and expectations are individually tested.
Tested Items	The products of the software process to be tested should be specified.
Testing Schedule	The overall testing schedule should indicate resources needed to perform this task in the time frame allotted.
Test Recording Procedures	The test results must be systematically recorded and repeatable.
Hardware and Software Requirements	This section should outline the software tools required and hardware requirements.
Constraints	A description of constraints affecting the testing process.

The test plan document is dynamic and may change to support the dynamic nature of the software life-cycle process. As changes are made to the software requirements, management of all documents, including the test plan document, becomes crucial. The

V&V process must determine how software requirement changes affect the overall testing plans, which may also affect the deliverables schedule. Thus, managing the documents is crucial to incorporating change into the overall software development life-cycle process.

2.9.4 Requirements Management

Requirements management is a life-cycle process that attempts to understand and control change to software system requirements. Change to requirements is constant due to the inability to fully define the problem upfront, therefore creating incomplete software requirements and specifications. As the software development process progresses the understanding of the problem also changes. These changes cause modifications to the original desired software system. These changes must be incorporated into the requirements and specifications in a systematic and traceable manner. The requirements change management process coordinates change requests and must ensure that the modifications are performed at all levels. Changes affect requirements, specifications, design documentation, implementation, verification & validation test plans, and operational procedures.

The change management process assesses the cost of changes and consists of three stages (Hooks, 2001) (Sommerville, 2001).

- **Problem Analysis and Change Specification:** The change proposal is analyzed to determine if the request is necessary, attainable, and verifiable.
- **Change Analysis and Costing:** The cost of making the change is estimated in terms of modifications to the requirements documents, specification documents, design documents, test plans and implementation.
- **Change Implementation:** Modifications to the requirements documents, specification documents, design documents, test plans, and implementation are scheduled and performed.

The ability to effectively trace changes in the baseline documents help to link requirements to stakeholders resulting in decision-making accountability (Ng & Yeh, 1990). Traceability provides an audit trail as to what changes were requested by whom and for what purpose. The traceability process helps to ensure accountability, which generally results in modification requests that are necessary, attainable, and verifiable.

The current state of the practice of traceability management is realized using Computer Aided Software Engineering (CASE) tools that support a change and version control system. Cadre TeamWork for Real-Time Structured Analysis (CADRE), Requirements Traceability Manager (RTM), SLATE, DOORS, Requirements Driven Design (RDD), Foresight, Software Requirements Methodology (SREM), Problem Statement Language / Problem Statement Analyzer (PSL/PSA), and Requirement Networks (R-Nets) are some of the more widely used software industry CASE tools (Nallon, 1994) (Rundley & Miller, 1994) (Vertal, 1994) (Palmer, 2000). These automated tools help to maintain a history of all the changes by capturing the following requirement attributes:

- Document Name
- Version Number
- Creation Date
- Modification Date
- Author
- Responsible Manager
- Approval
- Sign-Off
- Change Description
- Priority

These attributes help requirements management establish version control, change control, and traceability process procedures. These activities help to create a well-defined requirement definition process. These processes should be incorporated into the

software life-cycle to help ensure that the documents describing the software product contain all features and operational behaviors of the released software product.

Version and release management are the processes of identifying and tracking different versions and releases of the software system (Sommerville, 2001). CASE tools provide version numbering management and ensure that each version is uniquely identifiable. This type of configuration control generally uses some type of check-in/check-out procedures to help aid both developers and coordinators manage the process. Whereas version management identifies each component version, release management handles all the steps that are necessary to package the software solution to the customer. This packaging includes executable code, configuration files, data files, installation program, and documentation. Configuration management (CM) is the term used to identify configuration control processes. CM is separate from developmental procedures and helps to coordinate the release process. Thus, version control, change control, and traceability are intertwined and are essential components of the requirements management process.

2.9.5 Summary

Requirements analysis is the process of studying, determining and documenting user needs and expectations of the software system to be designed that solves a particular problem. The process generates software requirements documents that capture what is to be implemented by fully describing the software system's functionality, performance, design constraints, and quality attributes. The software system attributes are fully described in the functional requirements, non-functional requirements, and design

constraints documents. In addition to defining the static and dynamic features of the system, these documents describe end-user system needs. Table 2.14 summarizes the requirements analysis technique utilizing the characteristics of the analysis framework.

Table 2.14 Requirements Analysis Framework Characteristics

Function Description	Determined from both the user's point of view and development team during the elicitation process using various techniques.
Function & Communication Description	Determined from both the user's point of view and development team during the elicitation process using various techniques.
Communication Description	Described in the Software Requirements Specification using both textual and graphical descriptions.
Communication & Behavior Description	Described in the Software Requirements Specification using both textual and graphical descriptions.
Behavior Description	Described in the Software Requirements Specification using both textual and graphical descriptions.
Function & Behavior Description	Described in the Software Requirements Specification using both textual and graphical descriptions.

Requirements analysis does address the six characteristics of the analysis framework discussed as listed in Table 2.14. Requirements analysis is flexible in its approach of how to capture and properly document both end-user needs and system functionality. Precisely documenting the system to be built helps to ensure that final delivery of the system meets end-user expectations and functionality needs. Studying, determining, and documenting desired system functionality helps provide a fuller, richer application domain analysis.

2.10 Relationship Analysis

Relationship Analysis (RA) is an elicitation technique that identifies entities and their inter-relationships of a problem domain (Yoo, 2000). RA is an extension of Bieber's

original research work on hypermedia (Bieber, 1998) (Bieber & Yoo, 1999). The concept of hypermedia structures information as a collection of nodes and interrelating links. These interrelationships permit information to be accessed directly, as opposed to an indirect sequential structure. Bieber argues that hypermedia is a theoretical and practical means to facilitate information access both efficiently and effectively. Yoo's RA concept helps to determine the interrelationships of a problem domain using an informal technique performed during the analysis stage of the development cycle.

The RA technique explicitly identifies a domain's relationship structure, and in doing so, increases the understanding of that domain. A complete understanding of a domain relies on knowing how all the entities are interconnected (Yoo, 2000). In his dissertation, Yoo developed a thorough general relationship taxonomy listed in Table 2.15.

Table 2.15 Relationship Analysis Generic Relationships

Generic Relationships	Generalization/Specialization	
	Self	Characteristic Descriptive Occurrence
	Whole-part /Composition	Configuration/Aggregation Membership/Grouping
	Classification/Instantiation	
	Comparison	Equivalence Similar/Dissimilar
	Association /Dependency	Ordering Activity Influence Intentional Socio-organizational Temporal Spatial

These relationship categories were developed based on a very extensive literature and strenuous trial-and-adjustment prototyping review (Yoo, 2000). Yoo compares RA's

taxonomy with ten domain-specific taxonomies in detail and over twenty additional comparisons. RA's categories encompass all of these taxonomies' relationships. This includes object-oriented analysis, which provides RA's generalization/specialization, whole-part, classification/instantiation and association relationship classifications (Martin & Odell, 1995). Generalization/specialization relationships concern the relationships between objects in a taxonomy (Borgida et al., 1984) (Brachman, 1983) (Smith & Smith, 1977). Self-relationships include characteristic, descriptive, and occurrence relationships. Whole-part/composition relationships include configuration/aggregation relationships based on configuration aspects of the whole-part relationships, and membership/grouping relationships (Brodie, 1981) (Motschnig-Pitrik & Storey, 1995) based on membership aspect of the whole-part relationships (Henderson-Sellers, 1997) (Odell, 1994). Classification relationships connect an item of interest and its class or its instance. Comparison relationships break down into similar/dissimilar and equivalence relationships, involving such relationships as in thesaurus or information retrieval (Belkin & Croft, 1987) (Neelameghan & Maitra, 1978). Association/dependency relationships break down into ordering, activity, influence, intentional, socio-organizational, spatial and temporal relationships. The term association and dependency could be used interchangeably, because every association involves some concept of dependency (Henderson-Sellers, 1998). Because association is defined as a relationship that is defined by users, there could be no fixed taxonomy for it. The association relationship taxonomy is fluid compared with other relationships. The current association relationship taxonomy is based on observations, analyses, ontologies (Mylopoulos, 1998), and the existing classifications (Henderson-Sellers, 1998). Ordering relationships involve some kind of

sequence among items. Activity relationships are created by combining SADT activity diagrams (Mylopoulos, 1998) and case relationships (Fillmore, 1968). Activity relationships deal with relationships associated with activities or actions abstractly. This relationship could cover any activities that involve input or output, and deal with agents and objects involved in the activities. Influence relationships exist when one item has some power over the other items. Intentional and Socio-organizational relationships could be identified in intentional and social ontologies respectively. Temporal (Allen, 1983) (Frank, 1998) and spatial (Cobb & Petry, 1998) (Egenhofer & Herring, 1990) (Rodriguez et al., 1999) relationships deal with temporal and spatial perspectives, respectively. Each relationship category can be further broken down into lower levels of detail and Yoo's dissertation describes each of these and the literature from which each is derived (Yoo 2000).

Each of the taxonomy's sixteen categories has a series of brainstorming questions to help elicit a set of relationships that exist for a desired problem domain. Table 2.16 depicts a series of generic brainstorming questions that can be used to elicit domain information (Yoo, 2000).

Table 2.16 Sample Brainstorming Questions

Generalization/ Specialization	Is there a broader term for this item of interest? Is there a narrower term for this item of interest?
Characteristic	What attributes and parameters does this item of interest have?
Descriptive	Does an item of interest have a description, definition, explanation, or a set of instructions or illustrations available within or external to the system?
Occurrence	Where else does this item of interest appear in the application domain? What are all uses of this item of interest?
Configuration/ Aggregation	Which components consist of this item? What materials are used to make this item? What is it a part of? What phases are in this whole activity?

Table 2.16 Sample Brainstorming Questions (Continued)

Membership/ Grouping	Is this item a segment of the whole item? Is this item a member of a collection? Are these items dependent on each other in a group?
Classification/ Instantiation	Is this item of interest an example of a certain class? If a class, which instances exist for this element's class?
Equivalence	What is this item of interest equal or equivalent to in this domain?
Similar/ Dissimilar	Which other items are similar to this item of interest? Which others are opposite to it? What serves the same purposes as this item of interest?
Ordering	What prerequisites or preconditions exist for this item? What logically follows this item for a given user's purpose?
Activity	What are this item's inputs and outputs? What resources and mechanisms are required to execute this item?
Influence	What items (e.g., people) cause this item to be created, changed, or deleted? What items have control over this item?
Intentional	Which goals, issues, arguments involve this item of interest? What are the positions and statements on it? What are the comments and opinions on this item? What is the rationale for this decision?
Socio- organizational	What kinds of alliances are formed and associated with this item of interest? Who is committed to it in the organizational structure? Who communicates with it or about it, under what authority and in which role?
Temporal	Does this item of interest occur before other items? Does this item occur while other items occur?
Spatial	Which items is this item of interest close to? Is this item of interest nearer to the destination than other items? Does this item overlap with other items?

These questions are highly generic and should be tailored to the desired problem domain to be more meaningful. However, the literature does not indicate how to realize a more specific question set.

Yoo has demonstrated initial success to the proposed two-step RA process. The two steps are:

- Perform a stakeholder and item of interest analysis
- For each item of interest ask questions to identify relationships

Both steps are an informal process and once the initial items of interests are identified, the analyst isolates each item of interest and asks questions. The questions are

formulated to help explicitly determine relationships relevant to a particular item of interest. The process continues for each item of interest and the collection of explicit relationships comprises the relationship structure of the problem domain.

In regards to the analysis framework, RA directly addresses the identification of entities and relationships as illustrated in Table 2.17.

Table 2.17 Relationship Analysis Framework Characteristics

Function Description	Identified through stakeholder and item of interest analysis.
Function & Communication Description	Described using the brainstorming questions of the 16 Relationship Analysis categories.
Communication Description	Does not apply.
Communication & Behavior Description	Does not apply.
Behavior Description	Does not apply.
Function & Behavior Description	Does not apply.

RA provides a deeper understanding of the problem domain by explicitly determining its relationship structure. The resulting relationship structure helps analysts gain a deeper understanding of the problem domain during the analysis phase of the software development process. In turn, subsequent software development activities should also benefit from the initial, more complete problem analysis. In addition, documentation and specification should also be more complete and meaningful. Thus, RA enhances the system analyst's effectiveness, which should result in the development of higher quality software applications.

2.11 Analysis Quality

The processes used in the development of a software product influences the quality of that software product. The software development process involves intermediate steps leading up to the final software product. To help measure the quality of the end software product, it is possible to measure the quality of the intermediate steps in the development process. The development process begins with requirements gathering and analysis. The focus of this section will be on analysis quality.

The analysis process results in the creation of requirements specifications and in particular the Software Requirements Specification (SRS). Determining and documenting the requirements for a software system is, arguably, the key to developing successful information systems (Vessey & Conger, 1994). Reports show that up to 56% of all errors made on a software development effort can be traced to errors in the SRS (Boehm, 1975) (DeMarco & Yourdon, 1978). The implications, if SRS quality is ignored, are (Gause & Weinberg, 1989) (Davis, 1993) (Sommerville, 2001):

- The resulting software may not satisfy user needs
- Multiple interpretations may cause disagreements
- It may be impossible to thoroughly test
- The wrong system may be built

Therefore, developing a high quality SRS should help to contribute to a successful, cost-effective software system that solves end-user needs (Davis, 1993).

2.11.1 Quality Attributes of Natural Languages

The literature indicates many different software evaluation criteria (Boehm, 1976) (Gause, 1989) (Rombach, 1990) (Davis, 1993) (Sommerville, 2001). Boehm, for example, identified over sixty quality factors that affect total software quality. From this

study, software quality metrics (SQM) was developed (Murine, 1984). Software Quality Metrics (SQM) deals with how to measure the quality of the entire software development process or the intermediate steps comprising the software solution. SQM measures attributes of the software development process and evaluates software from multiple viewpoints.

The quality of an SRS is correlated to the successful implementation of the desired system. Quality can be an elusive and abstract concept to quantify. The ability to identify and evaluate those factors influencing and defining quality helps to mitigate the risk of generating an ineffective SRS. The use of natural language to prescribe complex, dynamic systems has at least three severe problems: ambiguity, inaccuracy and inconsistency (Wilson et al., 1997). To identify the quality of a written specification, a list of characteristics must be created. No industry-sanctioned set of quality attributes exists. The ACM, IEEE, ANSI and other organizations have yet to create a prescribed list of official characteristics that should be used to evaluate the quality of an SRS. Any organization implementing a software development project must determine a set of quality factors they deem appropriate for the software effort. Furthermore, on any project, requirements writers need to agree as to which quality attributes are most important, and strive for those attributes (Davis, 1993). Davis proposes the twenty-four attributes that help to determine a quality SRS listed in Table 2.18 (Davis, 1993).

Table 2.18 Quality Attributes

Quality Attribute	Description
Unambiguous	A statement that specifies a requirement is unambiguous if it can only be interpreted one way.
Complete	A complete requirements specification must precisely define all the real-world situations that will be encountered and the corresponding response.
Correct	For a requirements specification to be correct, it must accurately and precisely identify the individual conditions and limitations of all situations that the desired capability will encounter and it must also define the capability's proper response to those situations.
Understandable	An SRS is understandable if all classes of SRS readers can easily comprehend the meaning of all requirements with a minimum of explanation.
Verifiable	In order for a specification to be testable it must be stated in such a manner that pass/fail or quantitative assessment criteria can be derived from the specification itself and/or referenced information.
Internally Consistent	A consistent specification is one where there is no conflict among individual requirements statements that define behavior of essential capabilities.
Externally Consistent	
Achievable	An SRS is achievable if and only if there could exist at least one system design and implementation that correctly implements all the requirements stated in the SRS.
Concise	An SRS is concise if it is as short as possible without adversely affecting any other quality of the SRS.
Design Independent	An SRS is design independent if and only if there exist's more than one system design and implementation that correctly implements all requirements stated in the SRS.
Traceable	Each requirement statement must be uniquely identified to achieve traceability.

Table 2.18 Quality Attributes (Continued)

Quality Attribute	Description
Modifiable	In order for requirements specifications to be modifiable, related concerns must be grouped together and unrelated concerns must be separated.
Electronically Stored	An SRS is electronically stored if and only if the entire SRS is in a word processor, generated from a requirements database, or synthesized from some other form.
Executable/Interpretable	An SRS is executable, interpretable, or prototypable if and only if a software tool exists to input the SRS and providing a dynamic behavioral model.
Annotated by Relative Importance	Ranking specification statements according to stability and/or importance is established in the requirements document's organization and structure.
Annotated by Relative Stability	An SRS is annotated by relative stability if a reader can easily determine which requirements are most likely to change, which are next most likely, etc.
Annotated by Version	An SRS is annotated by version if a reader can easily determine which requirements will be satisfied in which version of the product.
Not Redundant	An SRS is redundant if the same requirement is stated more than once.
At Right Level of Detail	The right level of detail is a function of how the SRS is being used. Generally, the SRS should be specific enough that any system built that satisfies the requirements in the SRS satisfies all user needs, and abstract enough that every system that satisfies all user needs also satisfies all requirements.
Precise	An SRS is precise if and only if numeric quantities are used whenever possible and the appropriate levels of precision are used for all numeric quantities.
Re-usable	An SRS is re-usable if and only if its sentences, paragraphs and sections can be easily adapted for use in a subsequent SRS.
Traced	An SRS is traced if and only if the origin of its requirements are clear.
Organized	An SRS is organized if and only if its contents are arranged so that readers can easily locate information.

Table 2.18 Quality Attributes (Continued)

Quality Attribute	Description
Cross-referenced	An SRS is cross-referenced if and only if cross-references are used in the SRS to relate sections containing requirements to other sections.
Annotated by Relative Stability	An SRS is annotated by relative stability if a reader can easily determine which requirements are most likely to change, which are next most likely, etc.
Annotated by Version	An SRS is annotated by version if a reader can easily determine which requirements will be satisfied in which version of the product.
Not Redundant	An SRS is redundant if the same requirement is stated more than once.
At Right Level of Detail	The right level of detail is a function of how the SRS is being used. Generally, the SRS should be specific enough that any system built that satisfies the requirements in the SRS satisfies all user needs, and abstract enough that every system that satisfies all user needs also satisfies all requirements.
Precise	An SRS is precise if and only if numeric quantities are used whenever possible and the appropriate levels of precision are used for all numeric quantities.
Re-usable	An SRS is re-usable if and only if its sentences, paragraphs and sections can be easily adapted for use in a subsequent SRS.
Traced	An SRS is traced if and only if the origin of its requirements are clear.
Organized	An SRS is organized if and only if its contents are arranged so that readers can easily locate information.
Cross-referenced	An SRS is cross-referenced if and only if cross-references are used in the SRS to relate sections containing requirements to other sections.

This list of quality attributes considers several things: customer requirements, internal policies, problem domain, and personal preferences. The customer may specify that the requirements must be written using a given standard such as IEEE 830. The internal policies of an organization may require the use of specific tools to facilitate

modifiability, indexing and versioning. The problem domain may introduce specific quality requirements. For example, a system that interfaces to other systems should include traceability to other interface specifications. It is important to note that there is an art to writing requirements, which entails a certain degree of personal preference.

Table 2.19 depicts examples of selected quality characteristics and shows how two different organizations selected different subsets of Table 2.18 (Davis, 1993) (Wilson et al., 1997). The first column is a list of attributes collected from a formal IEEE research paper addressing the measurement of quality. The second column is a list of attributes collected from a series of papers written by NASA describing their Software Assurance Technology Center's (SATC) automated system for evaluating the quality of NASA technical documents.

Table 2.19 Examples of Selected Quality Characteristics

Quality Attribute	IEEE Metrics Symposium	NASA SATC Project
Unambiguous	*	*
Complete	*	*
Correct	*	*
Understandable	*	
Valid		*
Verifiable	*	*
Internally Consistent	*	
Consistent		*
Achievable	*	
Design-Independent	*	
Traceable	*	*
Modifiable	*	*
Electronically Stored	*	
Executable/Interpretable/Prototypal	*	
Testable		*
Annotated by Importance	*	
Ranked		*
Annotated by Relative Stability	*	
Annotated by Version	*	
At Right Level of Abstraction/Detail	*	

Table 2.19 reinforces the earlier findings that no list of universally accepted characteristics exists. Instead, organizations determine their own set of quality factors they deem appropriate for the software effort.

2.11.2 Generating a Quality Specification

The generation of an SRS containing desirable quality attributes is a three-tier process:

- Identification
- Generation
- Measurement

The value an organization assigns to an SRS can be evaluated by how well the tiers are realized. The methodologies for generating a quality SRS exist at the second tier. At this tier, an organization tailors its requirements generation approach to improve their quality. Three primary factors influence quality:

- Structure
- Process
- Tools

When Ericsson Eurolab evaluated their SRS documentation, they were dissatisfied with quality and resolved to improve their methodology. Ericsson found that their process to define and organize requirements was ineffective. They also determined that a new tool, a changed process, or another policy would not have solved the problem. A massive effort was required to effect a change of culture and behavior (Jacobs, 1999). The specification and generation of requirements documentation is not simply a process in the software life-cycle, but as Jacobs implies, it is part of the culture of an organization. Quality documentation is expensive and time consuming. In a cost and schedule focused industry, it can be difficult to justify resolving problems before they

exist. Identifying the structure of an SRS provides guidance to the author of the document. It should provide a partitioned list of subjects and definitions as to the content that each partition should contain. Several standardized structures exist. IEEE 830 is a well-accepted format that is widely used. The department of defense (DOD) uses Mil Std-498 and Mil Std-2167A, which both contain data item descriptions (DIDs) for software requirements specifications. Organizations that provide software for the military are typically required to use these documents. The VOLERE requirements process template is another popular format described by The Atlantic Systems Guild Inc. that when completed, represents a requirements specification. A requirements template is used as a guide to discovering requirements and building the specification (Nusibeh & Robertson, 1997).

A process of specifying requirements may be used to improve the quality attributes of the SRS. One such process is the Gib Style, which implements the following steps: structuring information, quantification, and sourcing (Jacobs, 1999). Three types of information should appear in an SRS: assumptions, requirements and a glossary. The SRS is based on the premise that all of the assumptions are correct. The Gib style then divides requirements into functional requirements, quality requirements, constraints and cost requirements. The glossary is used to insure that the meaning of words and concepts are clear and unambiguous. To quantify requirements, the approach identifies the following forms of measurement: gist, scale, meter, past/record, and must/plan/wish. The gist of a requirement is a rough approximate of what will be used to measure the requirement. The scale is used to define the units used to measure the requirement. A meter is used to measure the requirement against the scale. The past record is used to

compare the requirement to similar requirements in previous or similar projects, and what the user will expect from that requirement. The must/plan/wish defines the success criteria for that requirement. Gib also suggests appending the following information to every requirement: author, and the author's role. While these are all well accepted engineering practices, Ericsson found that when all stakeholders agree to a common, well-established process, the quality of the product improved.

The use of tools to generate requirements has exploded in the past several years. The advent of CASE tools for object-oriented analysis and design has benefited all methods of design. The literature advocates the use of use-cases as a functional requirement-triggering tool. Literally dozens of documentation management, configuration management, collaborative, multi-user, requirement specification, etc., tools exist and with each passing month more are created. While a tool will certainly not insure a quality SRS, it can provide a skilled engineer a valuable set of services. These services, when used appropriately, will contribute to the overall quality of the SRS.

2.11.3 Quantifying the Specification Quality

Many approaches can be used to quantify the quality of an SRS. The ability to recognize and measure quality attributes in an SRS helps to detect errors in the SRS, which in turn helps to develop a successful software system. The structure and attributes of the document, natural language, and specific requirements can be evaluated. In NASA's SATC (Huffman & Rosenberg, 1998) software system emphasis is placed on the attributes of the document, and natural language. The result is a set of metrics used to describe the SRS. These can be used to infer the quality of the SRS. The software

searches each line of text for specific words and phrases that are indicated by previous SATC's studies to be an indicator of the document's requirements specification quality. NASA's SATC uses counts of the items identified in Table 2.20 to help quantify specification quality (Wilson et al., 1997).

Table 2.20 Physical SRS Quality Attributes

Lines of Text	Physical lines of text as a measure of document size.
Imperatives	Words and phrases that command something must be done or provided. (shall, must, will, should, is required to, are applicable, and responsible for)
Continuances	Phrases that follow an imperative and introduce the requirements specification at a lower level for supplemental requirements count. (as follows, following, listed, in particular, and support)
Directives	References provided to figures, tables or notes. (figure, table, for example, and note)
Weak Phrases	Clauses that cause uncertainty and leave room for multiple interpretations or a measure of ambiguity. (adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not limited to, and if practical)
Incomplete	Statements within the document that have to-be-determined (TBD) or to-be-specified (TBS).
Options	Words that seem to give the developer latitude to satisfy the specifications that can be ambiguous. (can, may, and optionally)

Table 2.20 describes the physical SRS quality attributes to quantify the quality of the documentation. The number of lines of text in a document is a blunt measure of size. It provides no direct indication of quality. The number of imperatives provides an indication of the number of explicit requirements in a specification. NASA found that SRS documents with the majority of their imperatives at high levels of functional abstraction were the most explicit. The use of continuances provides an indication of the traceability of the document. A significant number of continuances suggested a document that emphasized traceability. However, an excess of continuances indicates overly complex and detailed requirements. The greater the level of traceability, the more

difficult it is to maintain the document. Each change will require that references to that section be verified. NASA found that a high ratio of directives compared to total lines of text is an indication of how precisely requirements are specified. They identified that the number of options in an SRS is proportional to cost and schedule overruns (Wilson et al., 1997). An option may indicate that the scope has not clearly been identified and may be uncontrolled. Weak phrases are associated with ambiguity within an SRS. A large number of weak phrases suggest an SRS that may be misinterpreted.

Another technique for evaluating the quality of a SRS is to explore the document's structure. Functional requirements documents are typically divided into paragraphs. Each paragraph in turn contains sub-paragraphs. High-level requirement documents rarely had numbered statements below a structural depth of four (Wilson et al., 1997). It is simple to evaluate the level of depth in a specification. While, it is not necessarily inappropriate to use more than four levels of depth in a high-level requirements specification, NASA has found that their engineers typically do not. This measurement can be used as an indicator to potential problems within the SRS. The text structure of documents, well organized and having a consistent level of detail, were found to have a pyramidal shape.

Davis provides mathematical equations to extrapolate metrics and evaluate the inherent quality of the SRS requirements and a subset of those are defined in Table 2.21 (Davis, 1993) and admits that this is just a beginning and hopes that others will be able to expand the list of quality attributes and provide other means of measurement.

Table 2.21 Quality Attribute Metrics

Quality Attribute	Metric	Description
Unambiguous	$Q = N_{ui}/N_r$	N_{ui} = number of requirements for which all reviewers presented identical interpretations N_r = number of requirements
Complete	$Q = N_u/(N_i \times N_s)$	N_u = number of actual unique functions N_i = number of inputs N_s = number of states
Correct	$Q = N_c/(N_c + N_i)$	N_c = number of correct requirements N_i = number of incorrect requirements
Understandable	$Q = N_{ur}/N_r$	N_{ur} = number of requirements reviewers think they understand N_r = number of requirements
Verifiable	$Q = N_r/(N_r + \sum C(R_i) + \sum T(R_i))$	N_r = number of requirements $C(R_i)$ = cost to verify requirement $T(R_i)$ = time to verify requirement
Concise	$Q = 1/Size + 1$	Size = number of pages
Design independent	$Q = D(Re \cup Ri)/D(Re)$	$D(Re \cup Ri)$ = number of actual solution designs that satisfy all requirements $D(Re)$ = number of actual solution designs that satisfy external behavior requirements
Not redundant	$Q = N_f/N_u$	N_f = number of functions specified N_u = number of unique functions

This section has described techniques used to quantify requirements specifications quality ranging from evaluation of words, phrases, and document structure to actual mathematical equations. The primary goal of the various evaluations is to help generate comprehensive documents that fully describe the software system to be designed and subsequently implemented.

2.12 Summary and Conclusion

The software development process is the progression from the identification of some application-specific domain need to the creation and delivery of a software product to fulfill that need. To understand the need, one must first understand the application domain. Analysis helps to identify and define application domain details. Software engineers have a wide assortment of analysis techniques that can be used to help analyze and design software systems. The underlying commonality or objective of different analysis techniques is to gain a better understanding of the problem domain by identifying entities and relationships.

The analysis component of the software development process determines the scope of the software system by completely describing what is to be created. The literature indicates that the current trend is to approach analysis and design of a software system in a component-oriented re-usable fashion. Components represent fully encapsulated entities of software system functionality that can interact with each other to provide a service. The aggregate of all components comprise the software system. The component-based approach represents a shift from the traditional process and data-oriented way of thinking to the object-oriented software development paradigm. Structured analysis and design uses the top-down approach to decompose the software system and represents a functionalistic approach to problem solving. In contrast, object-oriented analysis and design concepts utilize an object model approach to problem solving by encapsulating a related set of data and actions to be performed on that data into an object. The object-oriented paradigm uses common terminology throughout all phases of the software development life-cycle. This terminology uses objects as its basic

building blocks and helps to encapsulate an abstract concept into a self-contained unit. The object-oriented techniques are used throughout all phases of the software development process, from project definition to implementation phases. That is, objects identified during project specification are grouped and mapped directly into classes that can be implemented and coded. The resulting objects represent a self-contained, reusable software component.

The component-based approach provides object-oriented analysis and design powerful modeling techniques. The Unified Modeling Language (UML) is an industry-standard and accepted modeling language that is used to specify and document the data and processes of software system development in an object-oriented manner. It is unified because Grady Booch, James Rumbaugh, and Ivar Jacobson merged their individual object-oriented modeling techniques into a single modeling technique known as the UML, which is part of their overall Rational Unified Process (RUP) software development methodology. Modeling can improve all the activities that lead up to the deployment of a good software system. A successful software organization can consistently deploy a good quality software system that meets user needs and is usable. Therefore, the best way to deploy a software system that meets user needs and expectations is to properly capture the desired user requirements. In addition, to develop a quality software system it must have a solid architecture that can handle change. This implies that the development process used must be systematic and adaptable to changing needs by users, business, and technology. Modeling and in particular the UML helps organizations achieve and realize this goal. The UML provides eight process diagrams that graphically represent both the static and dynamic views of the software system.

These views help to describe the software system from multiple stakeholder perspectives. Some diagrams depict the system functionality requirements as seen by the users, while other diagrams outline the component attributes, relationships, and behavior from a more technical point of view. These diagrams aid analysis of the requirements and permit a common point of reference and language to be used among all stakeholders when discussing the software development project.

Requirements Engineering (RE) and Requirements Analysis is the process of studying, determining, and documenting user requirements and expectations of the software system to be designed and implemented. The process generates software requirement documents that capture what is to be implemented by fully describing the software system's functionality, performance, design constraints, and quality attributes. RE gets the users involved early on and throughout the entire software development life-cycle process. RE and in particular Requirements Analysis are supported in the object-oriented way of thinking and has further helped to evolve the object-oriented paradigm of software development to encompass a goal-oriented mindset. The focus of goal-oriented analysis is on the description and evaluation of alternatives and their relationship to the organizational objectives (Mylopoulos et al., 1999). Organizational objectives involve doing what is best for the company, which ultimately means developing a software system that both the customer and end-user expect. The goal-oriented way of thinking is to get the users involved by documenting and understanding their point of view or perspective. User participation increases the chances of their acceptance of the software system.

Relationship Analysis (RA), a relatively new elicitation technique, can help stakeholders gain a better understanding of the problem domain by its approach in identifying entities and their relationships. Although in its infancy, RA promises to be a technique that can be utilized at the project onset during the analysis phase. Yoo developed the preliminary RA elicitation process but it still needs to be refined into a well-defined process. An issue concerning RA is that its relationship taxonomy is not based on a theoretical model. To address this limitation RA should develop a new relationship taxonomy grounded in theory. Also, the current taxonomy's categories are not distinct enough and identical relationships are often discovered in multiple categories. A more thorough understanding of relationship identification spillover is needed. In addition, RA questions are highly generic and should be tailored to the desired problem domain. Therefore, it is necessary to study and formulate a way to describe how to realize a more specific question set. A facilitator performs the RA process and asks questions to a domain expert and records the results. However, there is no prescribed way to record the results. The RA process needs to be expanded to describe how the identified relationships are to be recorded. As previously mentioned a single facilitator performs the RA process with a domain expert, another area of study is to perform RA in a collaborative effort. Software development is a team effort from the problem definition through to its implementation phases. Since the intention is to make RA part of software development process, namely during the analysis phase, perhaps it best to perform RA utilizing a collaborative team effort.

CHAPTER 3

RELATIONSHIP ANALYSIS THEORY

Techniques exist to identify system entities and attributes, but provide a weak representation of relationships. There are two approaches to analyzing the relationship structure of a domain: the first is by using existing methodologies in practice, and the second is by using existing theories. These approaches are used within systems analysis and knowledge elicitation techniques to acquire information. The focus of this review is to describe the multitude of ways a domain's relationships are classified and identified.

The chapter begins by describing creativity in software engineering because analysis and relationship identification inherently is a creative process. Next, conceptual modeling in systems analysis is discussed. Following this, the chapter reviews existing methodologies and existing theories. This leads into the proposed semantic model to classify relationships and provides the theoretical background to the Relationship Analysis Model (RAM). The last section is a summary. Throughout the various sections, rationale is provided as to why the RAM is more appropriate than existing methodologies and theories.

3.1 Creativity in Software Engineering

Systems analysis consists of collecting, organizing, and evaluating facts about a system and the environment in which it operates. The objective of systems analysis is to examine all aspects of a system to establish a basis for designing and implementing the

system (Couger, 1973). Computer-oriented system analysis techniques have attempted to build more structure to the process. For example, Time Automated Grid (TAG) system, Problem Statement Language (PSL), and Problem Statement Analyzer (PSA) are early examples of semi-automated techniques to facilitate the process of mapping inputs, outputs, and data flow. These mappings provided an overview of the system and provided ways to show the relationships of all data in the system. Reports were generated that documented input, analyzed data requirements, data flow definitions, time-grid analysis, and database requirements. However, the process is still manually driven (Couger, 1973) and the individual or group is still responsible for defining the overall framework and structure of the problem. How does one go about defining, analyzing, and designing the structure of a software system's abstract concepts? The literature indicates that software analysis and design is just as much a creative process as it is an engineering process (Gomes et al., 2001) (Gero, 1994) (Partridge & Rowe, 1994).

Creativity involves a combination of originality and usefulness in producing outcomes. The choices made during a creative process are not necessarily the most obvious and often surprise observers due the originality of the innovative solution (Couger, 1994). Creativity in software engineering is a cognitive process that analysts use to generate products to satisfy certain kinds of properties. Software product creation involves identifying what is needed to solve a problem. The ability to creatively generate software products consists of three primary components, namely creative person, creative process, and creative product (Couger, 1990). Brown adds a fourth component, creative situation and argues that without a situation, there is no need for creativity (Brown, 1989). The creative process is used in both new idea generation and the transfer of ideas

or product re-use. The skills needed for both new and re-use product development is regarded as a type of reasoning ability associated with creative thinking in software analysis and design (Sycara & Navinchandra, 1991). Although component re-use is highly desirable, much of the time software engineers cannot use components without modification. Adapting existing components to newer systems requires creative reasoning. Re-use of ideas and components from other domains provides analysts the opportunity to find creative solutions.

Gero identified the following five main reasoning processes involved in creative software designs (Gero, 1994), which are inherently incorporated into analysis techniques including relationship analysis (RA).

- Mutation
- Combination
- Analogy
- Reasoning from Principles
- Emergence

Mutation is the modification of an existing design in order to generate a new design. The ability to transfer knowledge from an existing solution to a new problem helps construct re-usable solutions. One advantage of re-usability is that less time is needed to develop a component. Consequently, needed delivery time can be reduced. Re-usability is an established principle in engineering and applies previous successful solutions to new problems. In addition to reducing product cycle time, the strategy is aimed at reducing both cost and risk during the product development process (Arango, 1994). Thus, re-use has been identified as the key to improving software development and productivity. Another way for generation of new solutions is the combination of multiple pieces from different software product designs. Analyst's creativity is used to

determine whether and how multiple components can be leveraged into new software products.

Analogy is regarded as one of the more important processes in the creative design process because intellectual reasoning abilities are needed to determine whether a previous solution can be leveraged and re-used, to some degree, and incorporated into a new system (Gero, 1994). It comprises mapping between a source and target design (Gomes et al., 2001). Analogy reasoning is a mechanism for transfer of ideas across different domains and goes beyond component re-use by extending the scope into a different domain. Reasoning from principles makes use of domain models in order to generate new designs. Analysts use knowledge and past experience from existing models and extend it to the new system. This is realized by documenting the similarities in system structure and functionality of those systems.

Emergence is a process in which additional attributes are identified besides the intentional attributes. This reasoning mechanism represents the ability to view things in new ways, which is a characteristic of creative reasoning (Partridge & Rowe, 1994). Guilford categorizes emergence types of reasoning abilities as fluency and flexibility of thinking (Guilford, 1967) (Guilford, 1950). Guilford's Structure of Intellect theory captures creative reasoning through emergence utilizing divergent properties. These properties represent the ability to produce new ideas to the given information.

During the course of the System Development Life Cycle (SDLC), developers tend to focus on a relatively small aspect of a larger problem. Consequently, developers develop tunnel vision and lose sight of the larger issues being addressed. Sometimes, a systematic analysis process helps generate a breakthrough, often referred to as creative

inspiration (Nagasundaram & Bostrom, 1995). There are techniques that help stimulate creativity, and fundamental to these creativity techniques are the concepts of divergence and convergence (Couger, 1993). Divergence implies expanding the field of possibilities, and convergence implies reducing a field to one or a few for further consideration. The most widely known divergent technique is brainstorming. Others include excursion and wishful thinking. The more popular techniques for convergence are check listing, highlighting, clustering, and criteria grid. The use of these creative techniques in the information system development process can best be applied at four different points, namely near the conclusion of requirements definition, logical design, physical design, and program design (Couger, 1990). RA is a brainstorming technique and this dissertation applies RA at the logical design or analysis phase to systematically embrace the creative process in relationship discovery.

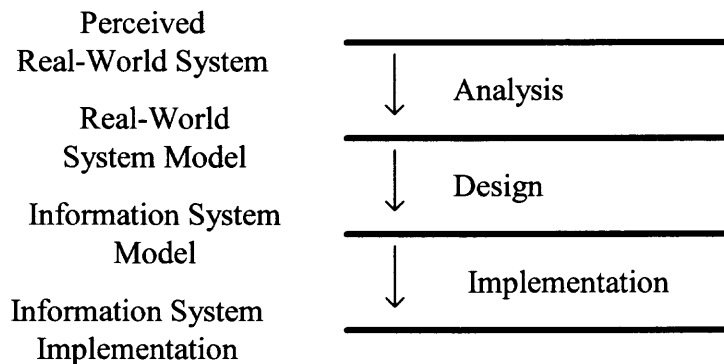
Developing and applying technology is both a creative and engineering process. The newer agile software methodologies described in Chapter 2, view collaborative approaches as an essential component to the creative process. Assembling a dynamic project team of individuals with complementary backgrounds and skills is a powerful ingredient to innovation and collaborative thinking (Capps, 2002), which fosters creative software development. Therefore, one aspect of the RA approach to relationship identification is to perform RA in a team environment.

3.2 Conceptual Modeling

A mental model is a conceptual representation of an abstract concept. A conceptual model provides an accurate and complete representation of a target system. Conceptual

models are used as tools for understanding the system. Therefore, conceptual models can aid human beings in developing an accurate mental model for an information system. As such, conceptual modeling or semantic modeling in systems analysis and development is designed to capture the meaning of an application domain. Human beings conceive of things in terms of models of things and form mental models utilizing their own unique style of information processing (Wu et al., 1998). The way people perceive and process information forms the uniqueness of their own cognitive learning style (Kolb, 1984). The value of conceptual models is their abilities to identify and capture the relevant knowledge about a domain. The literature refers to the conceptual model as a way to represent certain aspects of human perceptions of the real world so that these aspects can be incorporated into an information system (Wand, 1999) and is depicted in Figure 3.1.

Figure 3.1 The Role of a Conceptual Model in Systems Development



Conceptual models provide four roles (Chung & Solvberg, 1986):

- A way for developers and users to communicate
- Increase analysts' understanding
- Serve as a basis for design
- Serve as a means of documentation

Conceptual models are concerned with things or entities and relationships among these entities (Brodie, 1984). "Entity" is the term used in structured analysis (SA) and "object" is the term used in object-oriented analysis (OOA). Both SA and OOA are

established methodologies used in software engineering. While entities and relationships are fundamental concepts to conceptual modeling, of the two, relationships are more difficult to identify (Prietula & March, 1991) (Batra et al., 1990) (Goldstein & Storey, 1990). This is due to the fact that the meaning of a relationship is unclear. Modeling techniques provide three basic relationship categories, namely generalization/specialization, aggregation, and association (Booch, 1994) (Rumbaugh, 1991) (Martin-Odell, 1995) (Shlaer & Mellor, 1992) (Coad & Yourdon, 1990) (Jacobson, 1992) (Embley, 1992) (DeChampeaux, 1993) (Firesmith, 1993) (Henderson-Sellers, 1994). Whereas researchers agree that the meaning of generalization/specialization and aggregation is clear, what is the meaning of an association? Any dependency between two or more entities or objects is an association. An association only denotes a semantic dependency and does not exactly state the way in which one class relates to another. The type of relationship is not explicitly identified, instead it is implied by naming the role each class plays in the relationship (Booch, 1994). Teorey concurs and argues that relationships have some type of semantic meaning (Teorey, 1986).

Therefore, present day conceptual models do not adequately convey the relationship structure of the problem domain. This dissertation presents a semantic data model, based on theory, that does identify and document the relationship structure of the problem domain. However, prior to the description of the model, this chapter reviews existing practical and theoretical approaches to relationship classification.

3.3 Existing Methodologies

The analysis phase of the Software Development Life-Cycle strives to precisely and comprehensively isolate and understand the problem domain, and document what is to be built. Software engineering has several established methodologies to support the activities during the analysis phase. These methodologies are functional decomposition, structured analysis (SA), and object-oriented analysis (OOA). This section describes the techniques used by the methodologies and concludes by describing how RA could support (supplement) the analysis phase of the two most common life-cycle methodologies – Structured Analysis and Object-oriented Analysis.

Relationships within information systems are organized using modeling techniques. These techniques identify system components and properties to help represent the static and dynamic views of the problem domain. In particular, popular modeling techniques used in today's software engineering product development are:

- Entity-Relationship Modeling
- Object-oriented Modeling
- Unified Modeling

The remainder of this section will focus on describing these three popular modeling techniques in its method of classifying relationships. In addition, other semantic models: TAXIS, SDM, FDM, TM, SAM, Event Model, and SHM are also discussed and compared.

Entity-Relationship (E/R) modeling is one of the best known semantic data modeling approaches and is often used to represent the conceptual schema of the problem domain by identifying its entities, properties, and relationships. Conceptual modeling is an important phase to analyze and design database applications (Elmasri & Navathe,

2000). The concepts in a data model are usually represented in a diagrammatic form. A conceptual schema diagram must be powerful enough in its semantic expressiveness and easily comprehensible, as it serves as a communication medium between professional designers and users who interact with it during the stage of requirements analysis and modeling (Shoval & Frumermann, 1994) (Topi & Ramesh, 2002). E/R modeling is used extensively in database design. Prior to the creation of database tables, a domain's entities, properties, and relationships are graphically depicted using a diagramming technique known as an E/R diagram (Chen, 1976). An E/R diagram represents the logical structure of a database and provides a means of depicting the salient features of the design of the database. Each entity is shown as a rectangle containing the name of the entity type. Properties are shown as ellipses containing the name of the property and attached to the relevant entity or relationship via a solid line. Each relationship is shown as a diamond containing its name. Subsequent steps map the E/R diagram into a specific database management system (DBMS), depending on the data model and DBMS used for implementation (Shoval & Shiran, 1997). Within the DBMS, the concepts of entity and properties are represented via tables and attributes, while relationships are represented using primary and foreign key constraints. The main problem, however, is to create a good conceptual schema that is semantically correct, easy to use, and comprehensible. The quality of the database schema thus depends critically on the quality of the original conceptual schema.

The E/R model helps provide a high quality schema, by classifying relationships among entities as binary, n-ary, or recursive.

- Binary: Relationship between two entities
- N-ary: Relationship between more than two entities

- Recursive: Relationship between one entity and itself

The relationship classification is further extended to include cardinality facts among entities. Cardinality describes how many entities can be associated with one occurrence of the other entity in a relationship. Entities may have a one-to-one, one-to-many, or many-to-many cardinality relationship and are represented on the E/R diagram using simple notation. E/R modeling provides a very good method to graphically depict entities and their attributes and maps very well to DBMS implementation. Although relationships are also depicted on the E/R diagram, the amount of information these relationships convey is rather limited and depicts primarily the cardinality among entities. Also, each relationship is depicted by a single word, which helps to avoid a cluttered diagram depicting too much detail, but only provides minimal information describing the relationship.

Other researchers have extended and refined the E/R model to capture both entity and relationship representation by a clustering technique (Jaeschke et al., 1993) (Feldman & Miller, 1986) (Teorey et al., 1989) (Rauh & Stickel, 1992). The technique produces different levels of abstractions that group entities and relationships into a cluster. Relationships are identified in an iterative fashion and a new E/R diagram is generated that reflects these newer relationships. At the end of the process the top-most abstraction layer details the set of relationships existing in the data model. However, the major weakness is that the data model becomes more complex with each iterative refinement and the highest-level cluster, which represents the complete conceptual schema, is very complex.

Gandhi et al. developed a Leveled Entity Relationship Model (LER) that associates two entities, similar to an E/R entity, and also associates sub-entities (Gandhi et al., 1994). A sub-entity depicts a lower level of main entity abstraction. Moody enhanced the LER model by dividing the data model into subject areas (Moody, 1996). Each subject area is a portion of the E/R diagram. Each portion depicts sub-entities and their corresponding relationships. A top-level diagram, context data model (CDM), is used to depict all the subject-areas and the main relationships among them. The main limitations of this technique are that cardinality and n-ary relationships are not supported. In addition, sub-entity diagrams differ significantly from traditional E/R diagrams.

Overall, the various forms of E/R modeling provide a strong representation of entities and attributes, but provide a weak representation of relationships. Main relationships are depicted on the high level model in limited textual form, but important relationships among low-level constructs are not depicted. In contrast, RA identifies and documents relationships at all levels.

The Transaction And eXception handling Interactive database System (TAXIS) is a modeling technique that places emphasis on generalization/specialization abstraction hierarchies (Borgida et al., 1984) (Mylopoulos et al., 1980) (Nixon, et al., 1987) (O'Brien, 1983). The model combines ideas from programming language and database theory to support a semantic data model that utilizes the class concept. In TAXIS, a class is comprised of data and operations. The collection of classes represents conceptual abstractions depicted by an "is-a" hierarchy. This approach is extended to the system's dynamic features by creating hierarchies of operation invocation and exception occurrences. Such a modeling approach depicts relationships among classes by the

operations contained therein. Therefore, semantic relationships are determined by the operations each class can perform and are depicted in a hierarchical format. Whereas TAXIS exclusively models database systems utilizing the class concept, RA extends beyond database systems. In addition, RA is not bound by the class concept nor is the technique methodology dependent and can be used with other modeling techniques that do not use classes, such as structured analysis.

The Semantic Data Model (SDM) (Hammer & McLeod, 1981) also incorporates semantic modeling constructs into a collection of class abstractions. However, in contrast to TAXIS, the focus of an SDM model is upon the definition of the class itself and not its links or relationships to other classes via operations. SDM supports generalization, aggregation, and association relationships by defining members, attributes, interclass connections, and derivations of each class abstraction. Interclass connections are defined by sub-type and group constructs using generalization and association. SDM employs class abstractions to conceptually model entities. Relationships are embodied in the interclass connections that are specified as part of the class definition. A distinguishing feature of the SDM technique is the focus of the specification of the class without the development of hierarchies to depict relationships among classes (Peckham & Maryanski, 1988). SDM is similar to TAXIS in that it uses the class concept, RA does not have this restriction. In addition, SDM does not focus on relationship identification instead relationships are implicit in the class description. In contrast, RA explicitly identifies relationships among items.

The Functional Data Model (FDM) (Shipman, 1981) was constructed with the DAPLEX data definition language. The FDM model provides constructs to depict

entities and functions, but does not classify relationships nor represent generalization. Functions are used to define aggregation of attributes used to form the entity abstractions. Functions implicitly depict the relationships among entities (Buneman & Nikhil, 1984) and are used to show that an entity is comprised of an aggregate of disparate components. In contrast, RA is independent of the functions identified in the system. Instead, RA classifies relationships that exist for the entities and not just the aggregation relationships. Once the relationships are established, it could be used to help identify the functions of the system, in this case the aggregation relationships.

The Tasmanian Model (TM) is an extension to Codd's Relational Model and Temporal data (RM/T) (Codd, 1979) and captures the semantics of relationships through integrity rules. Relationships among database tables are dynamically formed based on the data in the tables. For example, if the same data exists in different tables, a relationship is established. Each entity type is defined by a single E-relation and one or more P-relations, which define the properties or attributes of the entity type. P-relations are directly associated with E-relations and represent a means of enhancing the semantic expressiveness of the model. RM/T provides integrity rules for the various entity types (Codd, 1979) (Davis, 1983) and explicit support for type hierarchies and is viewed as an enhanced E/R model. Although the TM is an improved E/R model, relationships are only made between data elements that exist in two or more tables through integrity rules. This works well for database tables and RA can also identify these types of relationships through its comparison focus. However, if data exists uniquely in a single table and nowhere else, no database relationships for that table will be identified. RA identifies relationships of all the data elements.

The Semantic Association Model (SAM) is a semantic model designed originally for scientific-statistical databases (Su, 1983). The technique provides a well-structured and semantically consistent approach to entity definition by classifying seven types of relationships among entities. The seven relationship types are: membership, aggregation, interaction, generalization, composition, cross product, and summarization. Su based these seven modeling constructs on evaluation of the requirements of the conceptual modeling needs of the CAD/CAM environment (Krishnamurthy et al., 1987). Whereas SAM's relationships are based on the modeling needs of CAD/CAM, RA is not restricted by a specific modeling environment. Instead it is based on theory to identify the complete set of relationships of a domain.

The event model provides support for generalization via functions and aggregation through attributes (King & McLeod, 1984). This model addresses both the static and dynamic properties of an application. A subtype relationship is used to organize the static schema into a set of hierarchies. Membership in a subtype is defined using predicates evaluated on attributes. Thus, the semantics of an attribute depend on the constraints imposed by the designer in the definition of the attribute type. The dynamic properties are determined through a sequence of design phases that create diagrams, similar to state diagrams, to depict dynamic behavior. This process models dynamic behavior better than other modeling techniques previously discussed due to the explicit identification of dynamic events. This differs from the RA approach in that the event model determines relationships during the design phase of the development process. In contrast, RA identifies the relationships during the analysis phase prior to

design. Therefore, RA improves the analysis phase and facilitates the design process in a more systematic manner.

Similar to the event model, the Semantic Hierarchy Model (SHM) (Brodie, 1984) models both the static and dynamic properties of an application. Object and behavior schemes are used to capture system properties. Structural relationships of data objects are captured via aggregation and generalization properties. These structural relationships model the static structure of the application. Connecting main entities to operations explicitly depicts behavior or dynamic relationship representation or association. The graphical representation of these control abstractions is identical to that used to represent the structural abstractions of aggregation and generalization. The similarity of constructs provides commonality to modeling semantic relationships (Peckham & Maryanski, 1988). Thus, SHM provides a unified modeling technique for both static and dynamic objects. This is a limited semantic model in which only generalization and aggregation relationships are identified. In contrast, RA provides a more complete set of relationships that model the domain more accurately and provides analysts a deeper understanding of the domain.

Within the object-oriented methodology, conceptual models of a problem domain are represented as a collection of interacting objects. These objects help to encapsulate an abstract concept into a self-contained unit. This unit or component-based approach is the foundation of object-oriented modeling (Booch, 1986) (Rumbaugh, 1991). Objects are organized by their similarities into classes. An object class describes a group of objects that have the same attributes and behavior patterns. This grouping of objects supports the concept of abstraction and affords modeling the ability to generalize a real-world

concept, as a single class comprised of a collection of interacting objects. Classes do not exist in isolation, rather for a particular problem domain, key abstractions are usually related in a variety of ways (Booch, 1994). Therefore, within the object-oriented methodology, relationship classification caters towards class representation. As a result, relationships of object-oriented conceptual models depict class relationships. There is no prescribed way to determine classes nor relationships, however techniques have emerged that offer recommended practices and rules of thumb for identifying classes and objects germane to a problem domain.

For example, Shlaer and Mellor suggest that candidate classes and objects usually come from one of the following sources (Shlaer & Mellor, 1992):

- Tangible Things: houses, cars, trees
- Roles: professor, manager, dean
- Events: request, response, eating
- Interactions: meeting, intersection

Ross offers a similar list (Ross, 1986):

- People: Humans who perform some function
- Places: Areas for people or things
- Things: Tangible physical objects
- Organizations: Formally organized collections of people, resources, facilities, and capabilities having a defined mission
- Concepts: Ideas that are not tangible
- Events: Things that occur at a given time or in a particular ordered sequence

Coad and Yourdon suggest yet another set of sources of potential objects (Coad & Yourdon, 1990):

- Structure: “Is-a” and “Part-of” relationships
- Other Systems: External systems in which the system interacts
- Devices: Devices with which the application interacts
- Events Remembered: An event that is recorded
- Roles Played: The different roles users play in interacting with an application
- Locations: Physical places, offices, and sites accessed by the application
- Organizational Units: Groups comprised of users

From these various ways of object classification, links and associations are used to depict relationships among classes and objects. Links are the conceptual connections between object instances. Associations are groups of links with common semantics and describe a set of links in the same manner that a class describes a collection of objects.

Object-oriented modeling techniques provide three basic relationship categories: (Booch, 1994) (Rumbaugh, 1991) (Martin-Odell, 1995) (Shlaer & Mellor, 1992) (Coad & Yourdon, 1990) (Jacobson, 1992) (Embley, 1992) (DeChampeaux, 1993) (Firesmith, 1993) (Henderson-Sellers, 1994).

- Generalization/Specialization/Inheritance: Denotes an “is-a” relationship
- Whole-Part/Aggregation: Denotes a “part-of” relationship
- Association: Denotes some semantic dependency among otherwise unrelated classes

Inheritance describes class relationships in the context of their similarities while preserving their differences. Inheritance defines class relationships, whereby a particular class shares the attributes or behavior in one or more other classes and represents a hierarchy of abstractions, in which a sub-class inherits from one or more super-classes. Sub-classes also contain their own unique features or differences not found in their super-classes. Semantically, inheritance denotes an “is-a” relationship since a sub-class “is-a” type of its super-classes. Without inheritance, every class would be a separate unit developed from the ground up. Inheritance makes it possible to define new software, by comparing it with something that is already familiar.

In contrast, aggregation denotes a “part-of” relationship among class objects and permits the grouping of logically related components. For example, a sentence is “part-of” a paragraph. Aggregation is a type of association but specializes relationships using the “part-of” connotation.

Any dependency between two or more classes is an association. An association only denotes a semantic dependency and does not exactly state the way in which one class relates to another. The type of relationship is not explicitly stated, instead it is implied by naming the role each class plays in the relationship (Booch, 1994).

Similar to E/R modeling, object-oriented modeling provides a strong representation of entities represented as classes and their attributes. Whereas the E/R model depicts the data model of the system so as to map the E/R diagram directly into a specific database management system (DBMS), an object-oriented model portions the system into a collection of sub-systems and classes. Inheritance and aggregation types of relationships are strongly defined among classes. However, all other types of relationships are strongly defined among classes. However, all other types of relationships that exist in the problem domain are lumped into the association category and depicted by a name connecting the classes. These names only indicate that a dependency exists but does not explicitly indicate how. Thus, association relationships are identified more implicitly than explicitly. In addition, the processes of relationship discovery are not defined and low-level relationships that exist among class objects are not identified.

The Unified Modeling Language (UML) approach to relationship classification is similar to object-oriented relationship categories. Both support generalization, association, and aggregation relationships categories. However, the UML more explicitly supports dependency and realizes relationship categories and specifies five types of relationships between classes (Kobryn, 2000) (Booch, et al., 1998):

- Associations
- Dependencies
- Aggregations
- Realizes

- Generalizations

Association, aggregation, and generalization relationship types are defined exactly the same as in object-oriented modeling. This is to be expected since the UML creators used object-oriented concepts from Booch, Rumbaugh, and Jacobson. Dependencies connect two classes but only in one direction and depict that one class can send a message to another class. A realize relationship is used to show the relationship between a class, package, or component, and its interface. The relationship connects a publicly visible interface such as an interface class or use-case to the detailed implementation and helps separate an interface from its implementation.

Similar to the object-oriented analysis, the UML provides syntax to depict generalization and aggregation relationship types. As with object-oriented relationship classification, generalization and aggregation types of relationships are strongly defined among classes. However, association, dependency, and realize relationships are identified by looking at the sequence and collaboration diagrams. The literature does not address the fact that in order to depict relationships on the class diagram static view, it is necessary to extract information from the dynamic view using sequence and collaboration diagrams. Also, only a label is used to indicate that an association, dependency, or realize relationship exists but does not explicitly indicate how. Thus, these relationships are identified more implicitly than explicitly. In addition, the processes of relationship discovery are not defined and low-level relationships that exist among class objects are not identified.

To summarize, semantic classification within information systems strongly categorizes main system entities or components but poorly classifies how they are

related. Table 3.1 summarizes the semantic modeling techniques discussed. These techniques model both the static and dynamic view of a desired system. The different views identify the problem and solution domain from different perspectives and map out the functional requirements. To this end, modeling techniques focus on identifying main system components but loosely identify how components are related and interrelated. These semantic models offer the modeler a small set of the fundamental abstractions needed to identify the relationships structure of the application domain.

Table 3.1 Semantic Models

Model Type	Relationship Representation	Relationship Categories
E/R	Tables	Aggregation
TAXIS	Classes	Generalization, Aggregation
SDM	Classes	Generalization, Aggregation, Association
Functional	Functions	Aggregation, Association
RM/T	Independent Entity	Generalization, Aggregation, Association
SAM	Independent Entity	Membership, Aggregation, Interaction, Generalization, Composition, Cross Product, Summarization
Event	Attributes	Generalization, Aggregation
SHM	Attributes and Entities	Generalization, Aggregation, Association
Object-oriented	Classes	Generalization, Aggregation, Association
UML	Classes	Associations, Dependencies, Aggregation, Realizes, Generalization

SA is the most popular approach to problem analysis and although SA is process and data oriented, its primary focus is to determine what data needs to be transformed by the system while maintaining a degree of separation between process and data. SA uses functional decomposition to map from problem domains to functions and sub-functions. Because of the emphasis on data, SA extensively makes use of analysis tools such as data

flow diagrams, which do not capture relationships, and entity-relationship diagrams, which capture merely a subset of relationships. SA techniques at best provide general guidelines for discovering relationships as opposed to providing a systematic approach.

The E/R diagram deals primarily with entities, their attributes and relationships (Chen, 1976). An E/R diagram maps from the real world into entities, attributes and relationships. It provides a way to express problem domain understanding by direct mapping. E/R diagrams for the most part allow for only a single relationship to connect two entities. Also, the analysis techniques for developing E/R diagrams provide, at most, *ad hoc* approaches for determining the relationships. Often it is assumed that the relationships are obvious between any two entities, and that an analyst will see them intuitively.

While SA is still widely used, especially in the United States, OOA rapidly is gaining popularity around the world. The development of OOA was realized by combining the concepts of semantic data modeling and object-oriented programming languages. OOA methodologies focus on objects and recommend the modeling of object classes including their attributes and behaviors as well as their relationships through the mechanism of message passing.

OOA uses popular tools such as use-cases and class diagrams extensively to document the processes and objects and make it easier to move from the analysis stage to design and then onto development. They provide, however, at most an *ad hoc* approach to documenting relationships, the focus being more on objects and their interactions via messages. The systematic nature of RA makes it an accessible approach to analysts of varying experience levels.

None of the existing methodologies explicitly helps the analyst in determining the detailed relationship structure of the application domain, and therefore they are not as comprehensive as analysts treat them. For any analysis methodology truly to be effective, it needs to be systematic, controlled and comprehensive. RA is a systematic, controlled technique that can supplement and “complete” the existing approaches.

3.4 Existing Theories

Conceptual models or semantic data models in systems analysis and development are designed to capture the meaning of an application domain. The value of conceptual models is their abilities to identify and capture the relevant knowledge about a domain. In order to categorically say that the model is complete, it should be based on a theoretical model. Wand promotes the idea that theories related to human knowledge can be used as foundations for conceptual modeling in systems analysis and development in general (Wand et al., 1995). This section describes three theories that have been used to develop conceptual models in the context of systems analysis, namely ontological theory, classification theory, and speech act theory. In addition, The Structure of Intellect (SI) theory is described in the context of classifying the complete range of intellectual ability. Guilford designed SI with a focus on measuring creativity (Guilford, 1950), which is an integral aspect of systems analysis and brainstorming activities in general. The latter theory has been adopted and applied to the Relationship Analysis Model (RAM) and arguments are made throughout this section as to why this is the best approach.

3.4.1 Ontological Theory

There is a need to share meaning of terms in a given domain. Achieving a shared understanding is accomplished by agreeing on an appropriate way to conceptualize the domain, and then to make it explicit in some language. The result, an ontology, can be applied in a wide variety of contexts for various purposes and may take a variety of forms, but necessarily it will include a vocabulary of terms, and some specification of their meaning (Uschold, 1998).

Ontology is a branch of metaphysics concerned with the nature and relations of being or existing. Since an information system represents a perceived real-world system, relationships can be viewed as constructs that model certain kinds of real-world phenomena (Wand et al., 1999). Therefore, it may be possible to derive the meaning of relationships via ontological theories. Fundamentally, ontologies are used to improve communication between either humans or computers (Jasper & Uschold, 1999) and can broadly be grouped into communication, inter-operability, and systems engineering benefits. The focus of the latter is to improve the process and/or quality of engineering information systems to encompass the following six areas (Uschold, 1998):

- **Re-usability:** The ontology is the basis for a formal encoding of the important entities, attributes, processes and their relationships in the domain of interest. This formal representation may be a particular component of a software system that meets specific requirements and certain criteria of a proposed software component of another software system. Therefore the existing component can be re-used in another software system.
- **Search:** An ontology may be used as meta-data serving as an index into a repository of information.
- **Reliability:** A formal representation makes it possible to systematically reproduce results, thereby providing more reliable software.
- **Specification:** The ontology can assist the process of identifying requirements and defining a specification for an IT system by providing a vocabulary of terms, and some explanation of their meaning.

- **Maintenance:** The use of ontologies in systems development, or as part of an end application, can render maintenance easier in two primary ways. The first is systems built using explicit ontologies serve to improve documentation of the software by providing a vocabulary of terms, and some specification of their meaning. This documentation can be used by those responsible to maintain the software to help gain a faster comprehension of the software, which helps to reduce maintenance costs. The second is if an ontology is used with multiple target languages, it only has to be maintained in one place because of its extensibility properties.
- **Knowledge Acquisition:** Using an existing ontology as the starting point may increase the speed in which knowledge is acquired since the existing ontology already provides a vocabulary of terms and their meaning.

In general, the accepted industrial meaning of “ontology” makes it synonymous with “conceptual model”. There is a slight differentiation between the terms. A conceptual model is an actual implementation of an ontology. Taxonomies are a central part of most conceptual models. Properly structured taxonomies help bring substantial order to elements of a model, are particularly useful in presenting limited views of a model for human interpretation, and play a critical role in reuse and integration tasks (Goldstein & Storey, 1999).

Given the diverse applications of ontologies from the literature, and the various dimensions by which they can be classified, four main categories emerge, namely neutral authoring, ontology as specification, common access to information, and ontology-based search (Jasper & Uschold, 1999). The commonality among the categories is the need for sharing the meaning of terms in a given domain, which is the central role of ontologies. Of these four, ontology as specification has been used to model application domains in terms of systems analysis. Ontology as specification and has been further subdivided into four ontologies concerned with conceptual modeling in the context of relationships (Mylopoulos, 1998), namely dynamic ontology, intentional ontology, social ontology, and static ontology. Dynamic ontology includes occurrence (state transition), temporal,

and influence relationships. Intentional ontology includes intentional relationships. Social ontology includes socio-organizational relationships. Static ontology includes all other types of relationships.

Bunge applies ontology to systems analysis and outlines ten ontological constructs to analyze the meaning of a relationship (Bunge, 1979), (Wand et al., 1999) (Weber & Zhange, 1996). These ten ontological constructs are:

- Thing: A thing is anything perceived as a specific object of the system, whether it exists in physical reality or in an analyst's mind.
- Property: Properties are attached to things either intrinsically or mutually. An intrinsic property is dependent only on one thing. A mutual property depends on two or more things.
- Attribute: Attributes are the properties of things and are characteristics assigned to things according to human perceptions.
- Class: A set of things possessing common properties.
- Kind: A thing defined by a set of properties.
- Functional Schema: A finite sequence of attributes defined on a certain domain.
- State: A state is a description of what a thing may change into.
- Law: Laws are restrictions of how a thing may change.
- Interaction: Things can interact, which may cause other things to change.
- Composition of Things: Fundamental ontological concept which addresses the notion that a thing is made of other things.

Table 3.2 shows how ontological constructs map to conceptual modeling constructs.

Table 3.2 Mapping of Ontological Constructs to Conceptual Model Constructs

Ontological Construct	Conceptual Model Construct
Thing	Entity, Object
Property	Relationship
Attribute	Attribute
Class	Entity Type, Object Class
Kind	Entity Type, Object Class
Functional Schema	Entity Type
State	No direct representation
Law	No direct representation
Interaction	Message connection, Relationship
Composition	Aggregate Entity, Object

These ten constructs encompass static and dynamic ontology. Three types of models related to systems analysis have been developed based on Bunge's ontology (Wand & Weber, 1995), namely the representation model, state tracking model, and system model. A representation model deals with the mapping between ontological constructs and information systems constructs. The state tracking model views an information system as an artifact that changes state to reflect the changes of state of the represented real world system. The system model analyzes the structure and behavior of a system as a whole in terms of the states and laws of its components.

Of the three model types, the literature indicates that only the representation model has been used in systems analysis practice. For example, the entity-relationship (E/R) model (Chen, 1976) maps an entity to a thing, an entity type maps to a functional schema, and a relationship maps to a property or interaction. Thus, the application of ontology to the E/R model results in rules for the use of entities, relationships and attributes (Wand et al., 1995). Bunge's ontology has also been used in object-oriented analysis (OOA) to propose a model of objects as representation constructs. The resulting model was used as the basis for an object-oriented (OO) conceptual model approach (Takagaki & Wand, 1991) and serves to propose guidelines for OO modeling (Parsons & Wand, 1991).

However, there are two main problems with an ontological approach as the basis to systems analysis. First, there is no generally accepted ontology (Wand et al., 1995). Bunge's ten ontological constructs is the only ontology relevant to systems analysis. A different ontology may utilize different constructs, thereby possibly leading to different outcomes. Therefore, how does one know what constructs should be employed? Second,

although Bunge's ontology is the only ontology specific to systems analysis, it does not deal with organizational and behavioral aspects of information systems. In contrast, the proposed relationship analysis model (RAM) encompasses the static, dynamic, intentional, and social ontologies in its classification and analysis of relationships.

3.4.2 Classification Theory

The terms classification, taxonomy, ontology, and morphology are often confused and used interchangeably. Classification is a systematic arrangement of information in groups or categories according to established criteria. Taxonomy is the orderly classification of plants and animals according to their natural relationships. Ontology is a branch of metaphysics concerned with the nature and relations of being or existing. Morphology is the study of structure or form. Simply stated, these are all ways of organizing information (things or animals) into categories. For example, the Linnaean system of classification used in the biological sciences to describe and categorize all living things in terms of genus and species is a classical taxonomy we are all familiar with today. Similarly, both the Dewey Decimal System and Colon Classification System describe the way libraries categorize and catalog information (Daniels & Martin, 2000) (Ranganathan, 1965). Information placed in categories based on common characteristics helps to break down information into smaller more manageable pieces. The aggregate of these organized components comprise the totality of what is being classified. Classification theory extends beyond the traditional biological and library sciences. Environments such as the World Wide Web and digital libraries face classification challenges. These challenges encompass effective information presentation, retrieval, and use (Giles et al.,

1998). Classification of information is as much an art form as it is a science. Successful information organization is to a large extent a function of the mental abilities that the performer brings to the task (Bloomberg & Weber, 1976). There seems to be a universal level at which humans name things. This level at the broader term is the genus level and for the narrower term, species level (Kay et al., 1991). The RA model also has broader and narrower levels. The broader term describes the focus or aspects of the relationship being classified, while narrower levels describe relationship types.

Concept theory is a type of classification theory that has been applied to systems analysis and involves the notion of a class as its fundamental concept. Classification theory defines a class as a well-defined set of properties that determines membership in a class. Therefore, a class is the way classification theory categorizes information. In essence, a class is the implementation of groupings. In concept theory, a class structure is a set of properties satisfying four conditions (Wand et al, 1995):

- Each class must be able to have instances
- Each class must contain every property common to all instances
- Every known property of an object must be included in the definition of at least one class in the class structure
- No class in a class structure is defined as the union of the properties of any other classes

A class organizes information in terms of cognitive economy and inference. Cognitive economy translates to instances of a class as being the same and provides a way to represent all the instances by a single class. Classes are unique when its instances contain meaningful differences (Rosch, 1978) with instances of other classes. Wand argues that meaningfulness can only be determined with respect to some use of knowledge (Wand et al., 1995) and meaningfulness differs among people. Therefore, the use of concept theory cannot always generate a set of classes to uniquely model a

domain. Instead, the generated classes are dependent upon the human being performing the task. Inference is the second way a class organizes information and it is the ability to derive conclusions on unobserved properties of class instances by classifying them based on other observed properties (Wand et al., 1995).

A concern in using classification theory in developing a conceptual model in systems analysis is which theory to use. Only the concept theory of classification theory has been described in potentially being useful in systems analysis (Wand et al., 1995). Other classification theories exist, but have not been applied to conceptual modeling in systems analysis. Finally, similar to ontological theory, classification theory does not consider beliefs, goals, organizational, and behavioral aspects of information systems in its classification schema. In contrast, the proposed relationship analysis model (RAM) encompasses these aspects in its classification and analysis of relationships. More research is needed in the area of applying conceptual modeling in systems analysis before it is possible to say that classification theory is or is not a worthwhile approach.

3.4.3 Speech Act Theory

Another interesting theory that has been applied to conceptual modeling in information systems is Speech Act Theory (SAT). SAT can be used to analyze the activities in a modeled domain and is widely accepted in linguistics and philosophy in the study of how language understanding and communication work (Fornara & Colonbetti, 2003) (Tosca, 2000). Speech acts (SACTS) are symbolic deeds that result in linguistic expressions having a meaning and always involve at least two agents, speaker and hearer. SACTS form conversations or discourses, which exhibit systematic regularities that can be

studied and analyzed. A speech act is the basic unit of communication and the premise of SAT is that every speech act can be analyzed as consisting of the following four distinct actions (Austin, 1962) (Searle & Vanderveken, 1985) (Auramäki et al., 1988):

- Utterance Act: Act that a speaker performs by uttering an expression.
- Illocutionary Act: A basic unit of meaningful human communication. It is always performed when one utters certain expressions with an intention.
- Propositional Act: Act of denotation and predication.
- Perlocutionary Act: Act involved in uttering that produces effects on the feelings, attitudes, and subsequent behaviors of the hearers.

Although there is no generally accepted description of the theory, the application of SAT deals with the classification of these four communication acts according to their intentions and possible effects. Sequences of SACTS form an ordered sequence or logical pattern. These patterns can be grouped into larger discourse segments. Segments share a common topic and have a goal that is relevant to achieving the purpose of the discourse type (Fornara & Colonbetti, 2003). SACTS can be analyzed to determine system relationships in the context of human-to-human relationships, human to system component relationships, and system component to system component relationships.

SAT has been used by different researchers to model different aspects of interactions (Wand et al., 1995). One of the most popular applications is Speech Act based office Modeling aPprOach (SAMPO) (Auramaki et al., 1988), which models office functions. The SAMPO systems analysis technique provides several tabular tools to describe a discourse in office information systems. The table is constructed by defining the discourse type on the basis of prerequisites and possibilities, thereby specifying the semantics of each unit of information as it passes among people and/or processes. These characterizations reveal features of the discourse. The literature indicates that SAMPO provides insights into observing and understanding information flows by specifying the

flow semantics in terms of social and nonsocial communication. However, the results were difficult to interpret and often ambiguous.

Current applications of SAT have drawbacks ranging from a lack of an overall picture of how actions relate among each other to ambiguous classification (Wand et al., 1995). It can be argued that the primary reason for problems associated with the application of SAT is its lack of an accepted description of the theory. Wand suggests that other techniques and theories should be used jointly with the application of SAT. In contrast, RA is a stand-alone technique, which provides unambiguous relationship classification categories. In addition, RA is based on theory to identify the complete relationship structure of a domain thereby improving the overall picture of the system and its relationships.

3.4.4 Structure of Intellect Theory (SI)

The Structure of Intellect (SI) theory is a general theory of human intelligence, thus forming a basis for comparing and classifying the complete range of intellectual ability. Guilford designed SI with a focus on measuring creativity (Guilford, 1950), which is an integral aspect of systems analysis and brainstorming activities in general. The SI model classifies intellectual abilities into a cross-classification independent three-plane system comprised of contents, products, and operations (Guilford 1956).

Figure 3.2 Guilford's Structure of Intellect Model

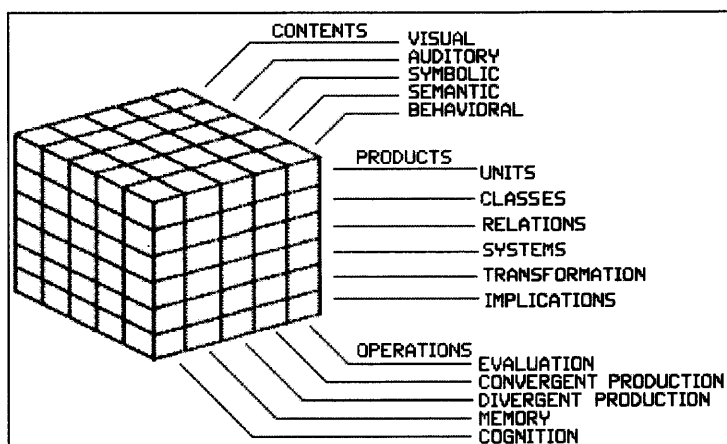


Figure 3.2 shows SI includes five kinds of contents, six kinds of products, and five kinds of operations. Due to the three independent planes, there are theoretically 150 different components of intelligence. The three dimensions of the model specify first, the operation, second the content, and third, the product of a given kind of intellectual act. Every intellectual ability in the structure is characterized in terms of the type of operation employed, the content involved, and the sort of resulting product. The convention (Operations, Contents, Products) is used to specify each factor. For example, (Cognition, SeMantic, Unit) or (CMU) represents cognition of a semantic unit. In this way the SI theory represents the major kinds of intellectual activities or processes as an interrelated three-dimensional model.

Turoff et al. apply SI to the computer application domain (Turoff et al., 1991) and argue that not all of the SI components are necessary for classifying computer application domains, they reduce it to two dimensions by classifying all SI types of content as one, namely semantic. The four SI contents; visual, auditory, symbolic, and behavioral are useful in classifying tests of intellect, but are not necessary for classifying application

domains. In addition, the SI operations, evaluation and memory are also not necessary for classifying application domains (Turoff et al., 1991).

Extending from these aforementioned models, the Relationship Analysis Model (RAM) approach in classifying relationships of computer application domains is to develop a semantic classification model. Therefore, the resulting model is a two-dimensional model, products vs. operations.

A product represents the organization that information takes in the analyst's processing of it (Guilford, 1967) (Meeker, 1969).

- Units: Most basic item. Things to which nouns are normally applied. Described units of information.
- Classes: Sets of items of information grouped by virtue of their common properties.
- Relations: Connections between items of information based on variables or points of contact that apply to them.
- Systems: Organized or structured aggregates of items of information.
- Transformations: Changes, redefinition, shifts, or modifications of existing information or in its function.
- Implications: Extrapolations of information. Emphasizes expectancies, anticipations, and predictions.

Operations represent major kinds of intellectual activities or processes that analysts perform with information (Guilford, 1967) (Meeker, 1969).

- Cognition: Discovery, awareness, or recognition of information by comprehension or understanding. Guilford views the cognition process as the classification of an object. Turoff et al. extend this concept to hypertext whereby cognition is represented by a node that classifies all the linked objects as related to a common concept or characteristic. Hypertext, at its core, concerns nodes (elements-of-interest) and links (relationships). These links or relationships among nodes are classified under convergent and divergent production properties. The RAM differentiates itself from the HMM in its application of cognition. The HMM represents cognition by a node and in hypertext terms: a node is an endpoint, and relationships exist among nodes or endpoints. In contrast, the relationships of each element-of-interest in the RAM represent by six cognitive focus perspectives.
- Convergent Production: Generation of information from the given information, where the emphasis is on achieving unique best outcomes. The given information

fully determines the response. Guilford views convergent production as when the input information is sufficient to determine a unique answer. Turoff et al. extend this concept and a convergent link is a relationship that follows a major train of thought. This is referred to as a convergent relationship in the RAM.

- Divergent Production: Generation of information from the given information, where the emphasis is on variety and quality of output from the given information. Guilford views divergent production as fluency of thinking and flexibility of thinking. Turoff et al. extend this concept and a divergent link is a relationship that starts a new train of thought. This is referred to as a divergent relationship in the RAM.

The RAM uses Guilford's categories from SI, condenses them in the same manner as Turoff et al., and re-labels several to reflect the goal of relationship discovery and documentation. The differences between the HMM and RAM are semantically metaphoric. The HMM, interprets the "products" as nodes or endpoints, while the RAM interprets "products" to represent the six possible *cognitive foci* of the current artifact or "element of interest" being analyzed. (Guilford views the cognition process as object classification.) The "operations" now represent relationships that either conceptually *converge* or *diverge* within this focus. (Guilford views convergent production as when the input information is sufficient to determine a unique answer. Turoff et al. extend this concept to a convergent link that follows a major train of thought. Guilford views divergent production as fluency and flexibility of thinking. Turoff et al. extend this concept to a divergent link that starts a new train of thought.) The following section applies the aforementioned concepts and describes the RAM in detail.

3.5 Relationship Analysis Model (RAM)

The Relationship Analysis Model (RAM) applies these three operations to the six products defined in the previous section to categorize relationships. Similar to Turoff's

Hypertext Morphology Model (HMM), each cognitive product becomes a focus point that classifies all the linked relationships pertaining to the particular cognitive focus. Thus, relationships of an element of interest are described by six cognitive focal points. Relationships of each focal point are classified under convergent and divergent operation properties. Therefore it is possible to classify the relationships of an element of interest in terms of six products each of which has convergent and divergent relationships. Table 3.3 depicts the cells of the model using SI nomenclature.

Table 3.3 Relationship Analysis Model Using SI Nomenclature

Cognition	Convergent Production	Divergent Production
Cognition, Semantic, Unit	Convergent, Semantic, Unit	Divergent, Semantic, Unit
Cognition, Semantic, Class	Convergent, Semantic, Class	Divergent, Semantic, Class
Cognition, Semantic, Relation	Convergent, Semantic, Relation	Divergent, Semantic, Relation
Cognition, Semantic, System	Convergent, Semantic, System	Divergent, Semantic, System
Cognition, Semantic, Transformation	Convergent, Semantic, Transformation	Divergent, Semantic, Transformation
Cognition, Semantic, Implication	Convergent, Semantic, Implication	Divergent, Semantic, Implication

During the analysis process, documents and dialogue provide analysts descriptions of desired system functionality. From these documents it is possible to extract elements of interest. Our goal is to fully describe the relationships of desired elements of interest using the meanings of the cells outlined in Table 3.3. As such, the following sub-sections describe the six focal aspects of the classification of relationships based on the SI theory and is depicted in Table 3.4.

Table 3.4 Relationship Analysis Model (RAM)

Cognition Focus	Convergent Relationship	Divergent Relationship
Unit	Specification	Elaboration
Collection	Membership	Aggregation
Comparison	Generalization/Specialization	Similar/Dissimilar
System	Structure	Occurrence
Transformation	Modify	Transpose
Implication	Influence	Extrapolate

3.5.1 Unit Focus

Guilford views a unit as relatively described items of information (Guilford, 1967). One thinks of items of information as units or definitions, first, before they form collections or groupings. Guilford views the cognition process as the classification of an object (Guilford, 1967) and as such, cognition of a unit takes the form as defining the object. These descriptions or definitions can be explicit or implicit. Explicit descriptions yield specific relationship types. In contrast, implicit relationships are uncovered as descriptions are further elaborated. Descriptions provide characteristics of items of information, which are attributes, also known as metadata. Thus, metadata relationships are identified within unit focus.

Guilford views convergence as when the input information is sufficient to determine a unique answer (Guilford, 1967). Therefore, unit or definition convergent relationships are explicitly specified in the description of the element of interest. The following list is an example of questions to determine specific convergent relationships (Yoo, 2000).

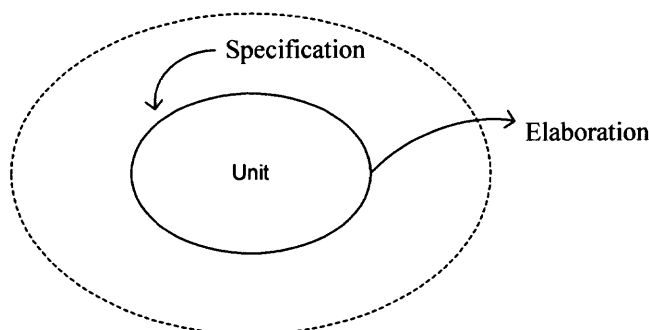
- Does the item have a description?
- Does the item have a definition?
- Does the item have an explanation?
- Does the item have a set of instructions?
- Does the item have an illustration?

In contrast, divergent relationships are determined as descriptions are further elaborated. Guilford views divergence as flexibility of thinking (Guilford, 1967). These types of relationships are generally found just below the surface of the description and not within the description of the element of interest. The following list is an example of questions to determine elaborated divergent relationships.

- Does the description fully describe the item?
- Does the definition fully encompass the item?
- Does the explanation make assumptions?
- Are the set of instructions complete?
- How can this item be expanded or broadened?

Both structured and object-oriented analysis utilizes functional definitions to help perform the analysis (Martin & Odell, 1995) (Borgida et al., 1984) (Brachman, 1983) (Smith & Smith, 1977). Jacobson's use-case analysis technique has made the process more explicit by generating descriptions of the use-cases (Booch et al., 1998). Use-case descriptions are narratives that describe a functional aspect of the desired system. From these narratives it is possible to extract both explicit and implicit relationships. Unit or definition focus is depicted in Figure 3.3.

Figure 3.3 Unit Focus



For example consider the following brief description. The software operates on both an Intel x86 and Motorola 68x000 architecture base computer and the results of the software are displayed on a monitor. Does the base computer have a description? Yes, a

specification relationship would be that between the computer and its Intel processor type, namely an Intel x86 and Motorola 68x000. However, what type of monitor? An elaboration relationship is determined by inquiring about the type of monitor desired. Does the description fully describe the monitor? No, the original description does not contain monitor specific information or a description, but can be determined by elaboration. In this case lets assume a flat-panel display.

The next example identifies a specification and elaboration relationship from the task problem statement of Appendix F. Does a course have a definition? A course is defined by three items: the professor teaching the course, the department offering the course, and the prerequisites required to enroll in the course. Does the description fully describe the item? No, additional information is needed to more fully describe a course. Some of this information includes the days of the week the course is offered, the time of day the course is held, the location of the course, the number of course credits, the course number, the course section number. These types of questions asked during the analysis phase helps to more fully describe the problem and aids to document the relationships of the system components. The result is a more comprehensive understanding of the domain.

3.5.2 Collection Focus

Collections are recognized sets of information grouped by virtue of their common properties (Guilford, 1967). Collections are derived from the previously determined definitions or units. It is valid to suppose that before one can make collections around a unit, one would have to perceive the unit already defined (Meeker, 1969). At the time of

Guilford's writing of the SI theory, analysis, as it pertains to software systems development, did not exist. The term class has a different connotation in present day software engineering methodologies. As in Guilford's definition, a class is a grouping of information or a collection of information. To prevent confusion with the term class in software engineering methodologies, the term collection is used in place of class. Therefore, collection (class) focus emphasizes group or collection relationships of units of information.

Guilford views convergent production of semantic collection as the ability to produce meaningful collections or groups under specific conditions and restrictions (Guilford, 1967). Therefore, collection convergent relationships represent groupings or membership properties. Membership relationships of collections are based on aspects of the whole-part properties (Henderson-Sellers, 1997) (Odell, 1994). Its intent is to represent an element of interest as a member of a collection. Membership connects a member of a collection to other members or to a whole collection or class. The following questions determine membership relationships (Yoo, 2000).

- Is this item a segment of a whole item?
- Is this item a member of a collection?
- What is this item a part-of?
- What components consist of this item?
- What phrases are in this whole activity?

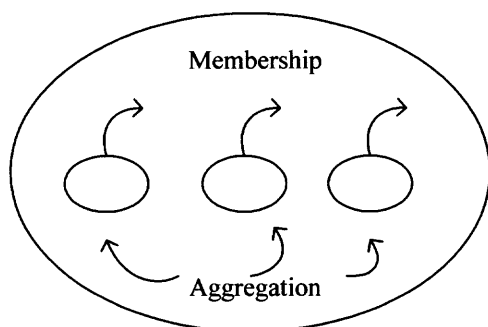
In contrast, Guilford views divergent production of semantic collection as the ability to produce meaningful sub-categories of ideas appropriate to a given collection (Guilford, 1967) (Meeker, 1969). Therefore, collection divergent relationships represent the components or aggregates of collection members. Aggregation relationships are determined for the collection members or whole-part composition (Boggs & Boggs,

2002) (Booch et al., 1998) (Brodie, 1981) (Motschnig-Pitrik & Storey, 1995). Its intent is to represent an element's members as part-of the whole. The following list is an example of questions to help determine aggregation relationships (Yoo, 2000).

- Which components comprise this item?
- What materials are used to make this item?
- What is part-of this item?

The collection focus is depicted in Figure 3.4 and represents membership relationships as looking outside the element and aggregation relationships as looking inside a collection. The premise is that membership converges to whole collection and aggregation diverges into the disparate components.

Figure 3.4 Collection Focus



Extending the computer brand example, one can describe a membership relationship as that among processors. What is the Motorola 68x000 architecture base computer a part-of? The Motorola 68x000 and Intel x86 processors are part of computer systems that can both run the same software package. Aggregation relationships determine components of the computer system. What is part-of the computer system? The computer is comprised of the aggregate of base unit, monitor, keyboard, mouse, speakers, and CD/DVD R/W.

The next example identifies a membership and aggregation relationship from the task problem statement of Appendix F. What are a professor's publications a part-of?

Publications are part of a list of a professor's projects and research interests. What is part-of a professor's publication list? Publications are comprised of publication type, title, date of publication, and co-authors.

3.5.3 Comparison Focus

The comparison focus, which is equivalent to Guilford's term relation, is defined as recognized connections between items of information based upon variables or points of contact that apply to them (Guilford, 1967). To prevent confusion with the term relationship in RA, the term comparison is used in place of relation.

Guilford views convergent production of semantic relation (comparison) as the ability to produce an idea that conforms to specific relationship requirements (Guilford, 1967) (Meeker, 1969). The ability to specify from a general meaning to a more specific or specialized meaning represents a way to represent commonality among concepts (Boggs & Boggs, 2002) (Booch et al., 1998). In terms of analysis, generalization/specialization are the terms used to describe commonality among components and the phrases "is-a" or "a-kind-of" are used to relate objects (Booch, 1994) (Rumbaugh, 1991). The following questions help determine generalization/specialization relationships (Yoo, 2000).

- Is the item a kind of parent item?
- Does the item completely include or encompass other items?
- Is there a broader term for this item?
- Is there a narrower term for this item?

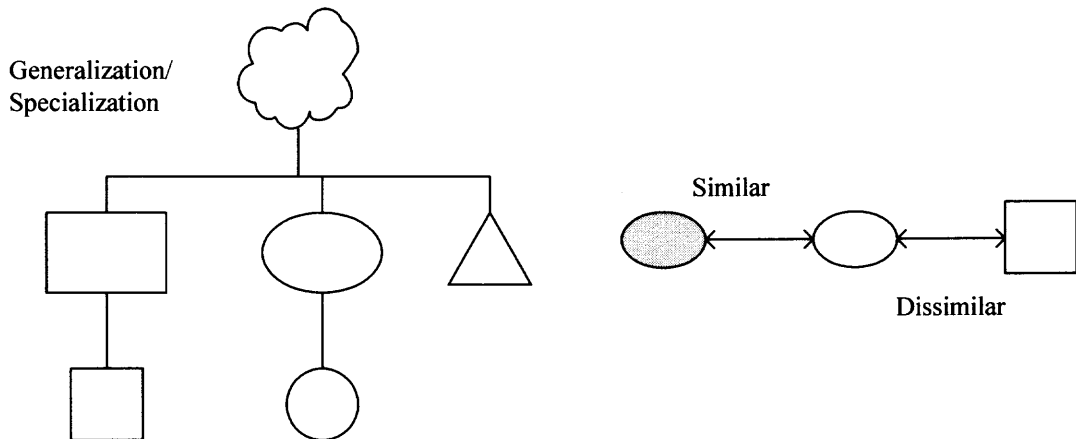
In contrast, Guilford views divergent production of semantic relation (comparison) as the ability to produce many relationships appropriate in meaning to a given idea (Guilford, 1967). The identification of appropriate meaning among

information represents similarity characteristics between information components. In addition, dissimilar characters are also determined as a natural result of components not being similar. Therefore, comparison divergent relationships represent both similarity and dissimilarity among elements of interest. Characteristics or attributes become criteria to determine the degree of similarity present with other elements (Booch et al., 1998) (Belkin & Croft, 1987) (Neelameghan & Maitra, 1978). The following questions help determine similar and dissimilar relationships (Yoo, 2000).

- Which other items are similar to this item?
- What serves the same purposes as this item?
- Which others items are opposite to this item?

Comparison focus is depicted in Figure 3.5.

Figure 3.5 Comparison Focus



An example of generalization/specialization relationships exists for shapes. A rectangle, ellipse, and triangle are all types of shapes. A square is a type of rectangle and a circle is a type of ellipse.

An example of a generalization/specialization relationship from the task problem statement of Appendix F exists for publication types. Is a journal a kind of publication? Yes. Is a book a kind of publication? Yes.

An example of similar/dissimilar relationships from the task problem statement of Appendix F exists for students and professors. Which other items are similar to freshman students? Other students of different rankings such as sophomore, junior and senior are similar to freshman students since they are register and take courses. Which others items are opposite to this freshman students? Professors are opposite to freshman students since professors teach courses and students take courses.

3.5.4 System Focus

Guilford defines a system as organized or structured items of information, a complex of interrelated parts (Guilford, 1967). Cognition of a semantic system shows comprehension of meaning derived from a system of components.

Guilford views convergent production of semantic system as the ability to order or structure information into a meaningful sequence (Guilford, 1967) (Meeker, 1969). Structure identifies how an item fits into the framework of a system and includes spatial perspective concepts of before, after (Cobb & Petry, 1998) (Egenhofer & Herring, 1990) (Rodriquez et al., 1999), above and below. The following list is an example of questions to help determine structure relationships (Yoo, 2000).

- What prerequisites or preconditions exist for this item?
- What follows this item for a given purpose?
- What precedes this item for a given purpose?
- Which items are close to this item?

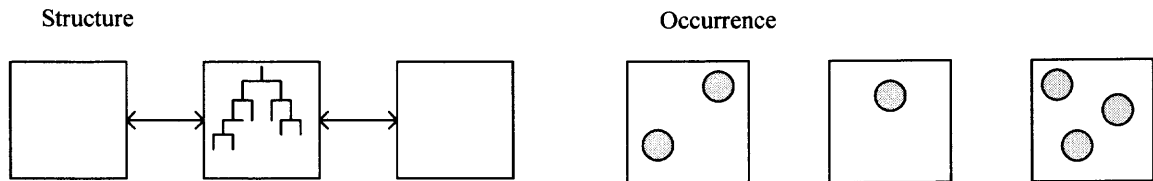
Guilford views divergent production of semantic system as the ability to organize information in various complex ideas (Guilford, 1967) (Meeker, 1969). Its intent is to represent an item within the context of its appearances and uses at different places and can be viewed as occurrence relationships based on the temporal attributes of before,

during, and after (Allen, 1983) (Frank, 1998) (Cobb & Petry, 1998) (Egenhofer & Herring, 1990) (Rodriguez et al., 1999). The following questions are examples to help determine occurrence relationships (Yoo, 2000).

- Where else does this item appear in the domain?
- Where else is this item used in this system and in other systems?
- What are all uses of this item?
- Where was this item used before?
- Where else is the item used now?
- Where will this item be used later?

The system focus is depicted in Figure 3.6.

Figure 3.6 System Focus



An example of structure and system relationships from the task problem statement of Appendix F exists for course registration. What follows the course registration for a given purpose? Following the registration process, outstanding course assignment conflicts are resolved. What are all uses of course registration? Course registration informs professors about student enrollment. Course registration is used as the feeder into the bill generation process. As a semester progresses, students must be able to access the on-line system to add or drop courses.

3.5.5 Transformation Focus

Transformations are changes of various kinds, of existing or known information in its attributes, meaning, role, or use (Guilford, 1967). A transformation is a matter of redefinition of an element. In essence, it is the ability to see potential changes of

interpretations of elements and situations dependent upon a particular activity (Meeker, 1969). Therefore, it represents an element in the context of its activities.

Activity relationships are created by combining SADT activity diagrams (Mylopoulos, 1998) and case relationships (Fillmore, 1968). These relationship types cover activities that involve input or output, and deal with agents and elements involved in the activities.

Guilford defines convergent production of semantic transformation as the ability to produce new uses for elements by taking them out of their given context and redefining them (Guilford, 1967). Convergent transformation is how an item can be modified focusing on the item itself and how it can change. As such, information is acted upon and modified. The following questions are examples to help determine modify relationships.

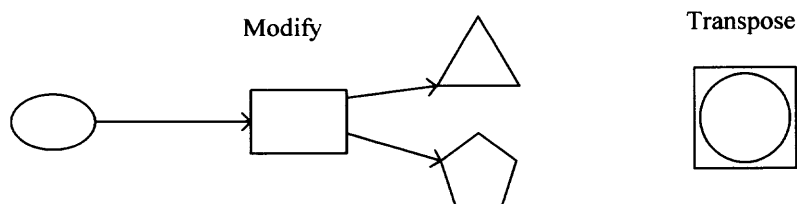
- What can this item change into?
- What output results from the item's inputs?
- What resources and mechanisms are required to modify this item?
- Who can modify this item?

Guilford views divergent production of semantic transformation as the ability to produce responses involving reinterpretations or new emphasis on some aspect of an element or situation (Guilford, 1967). Meeker extends this definition and argues that it is the ability to produce responses remote in time, remote in space, and remote in sequence (Meeker, 1969). Divergent transformations are those that reuse the item in different contexts or view the item in different ways. This is a transpose relationship, which is to change in form or nature, or to re-conceptualize the item. The following list is an example of questions to help determine transpose relationships.

- How can this item be reused?
- How can this item be viewed differently?
- Can this item be used in a different context?

Transformation focus is depicted in Figure 3.7 and depicts how an item can be modified or changed. In addition, Figure 3.7 depicts one shape being squeezed or transposed into another, in this case a square being squeezed into a circle.

Figure 3.7 Transformation Focus



An example of a modify and transpose relationship from the task problem statement of Appendix F exists for student's course schedules. What output results from the student's course schedule inputs? A bill is generated based on a student registering for courses. Who can modify a list of courses? The registrar can modify the course list. How can the courses selected by the students be reused? The most highly selected courses are those of most interest and perhaps need more than a single section.

3.5.6 Implication Focus

Implication emphasizes expectancies, anticipations, and predictions, the fact that one item of information leads naturally to another (Guilford, 1967). Meeker argues that cognition of semantic implication is the ability to anticipate consequences of a given situation in meaningful terms (Meeker, 1969). In essence, it is the ability to anticipate consequences of an item of interest in an organization or a social setting.

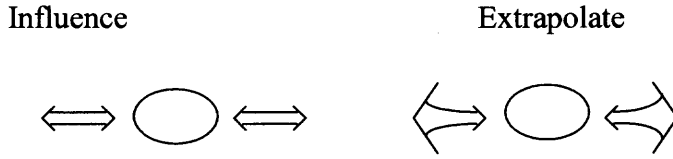
Convergent production of semantic implication is the ability to deduce meaningful information in the given information (Guilford, 1967) (Meeker, 1969). Convergent implication is dependence and control relationships both on an element and by an element and exhibits some type of influence on other elements. It is how an element of information influences, controls, impacts, or if conscience thinks about other people or things in the social environment. The following questions are examples to help determine influence relationships (Yoo, 2000).

- What items or people cause this item to be created, changed, or deleted?
- What items or people have control over this item?
- What is this item dependent on?
- What is dependent on this item?

Divergent production of semantic implication is the ability to produce many antecedents, concurrents, or consequents of given information (Guilford, 1967) (Meeker, 1969). In contrast, to influence relationships, there is more freedom to produce information in divergent production of semantic implications. In context of a social setting, relationships are extrapolated from the given information. Divergent implication is impacts, consequences, extrapolations, rationale, deductions, and opinions both on an element and by an element. The following questions are examples to help determine extrapolate relationships (Yoo, 2000).

- Which goals, issues, and arguments involve this item?
- What are the positions and statements on the item?
- What are the comments on this item?
- What are the opinions on this item?
- What is the rationale for this decision?

Implication focus is depicted in Figure 3.8.

Figure 3.8 Implication Focus

An example of an influence and extrapolate relationship from the task problem statement of Appendix F exists for University courses. What items or people cause a course to be created, changed, or deleted? The registrar can create, change or delete a course. What is the rationale for the decision to delete a course? Insufficient enrollment is the reason to cancel a course. Excess enrollment is the reason another section of a course is offered.

3.5.7 Relationships Among the Relationships

Interrelationships are relationships between relationships and can exist among the primary relationship categories. Determining the interrelationships of a domain further helps analysts understand the domain by identifying its interconnections. In addition, interrelationships often appear as overlaps when analysts discover relationships during brainstorming sessions. For example, it is intuitive to assume that members of a collection may have an ordering or path association. Another example involves the way a component can influence an aggregation or a system containing it. So the question is, how does one determine interrelationships?

Understanding the order of Guilford's original model, which has logical reasoning behind it, best identifies the types of interrelationships. Units are regarded as basic and appear at the top. Units enter collections and collections are sets of elements with one or more common properties. A comparison is some kind of connection between the two

things. Systems are complexes, patterns, or organizations of interacting parts. Transformations are changes, revisions, redefinitions, or modifications, by which any element in one state goes into another state. Finally, an implication is something expected, anticipated, or predicted from the given information. These definitions and the characteristics of the relationship categories themselves yield the following types of interrelationships:

- The aggregation relationship can become the membership relationship if the dependency between the parts or the parts and the whole disappears.
- The occurrence relationship can become the transpose relationship if two or more occurrences of the same item can be treated as a null transformation.
- The generalization/specialization relationship can also be viewed as a transpose relationship since the generalization can be considered a broader view of an item.
- The structure relationship can be viewed as membership if the collection constitutes a system.
- The elaboration relationship can become the extrapolate relationship when descriptions are intention related.
- The extrapolate relationship can become the modify relationship through an activity performed.
- The modify relationship can become influence relationship through control.
- The influence relationship can become the occurrence relationship through control.
- The membership relationship can become a structure relationship based on ordering characteristics.

3.5.8 Mapping RAM to RAF

Table 3.5 maps the RAM relationships to the relationships determined by Yoo (Yoo, 2000) to classify an application domain. Yoo's dissertation presents a taxonomy that encompasses many existing modeling languages, theories, models, and existing taxonomies. Therefore, since the RAM incorporates Yoo's taxonomy, it is possible to conclude that the RAM also identifies all the relationships of these existing methods.

Table 3.5 Relationship Map Between Yoo's Categories and RAM

Yoo's Generic Relationships	RAM Relationships
Descriptive	Specification, Elaboration
Characteristic	Specification, Elaboration
Classification/Instantiation	Membership
Membership/Grouping	Membership
Configuration/Aggregation	Aggregation
Generalization/Specialization	Generalization/Specialization
Equivalence	Similar
Similar/Dissimilar	Similar/Dissimilar
Ordering	Structure
Occurrence	Occurrence
Activity	Modify
Temporal	Transpose
Spatial	Transpose
Socio-organizational	Influence, Extrapolate
Influence	Influence
Intentional	Extrapolate

Table 3.6 provides an overview of generic elicitation questions for each relationship type outlined in the description of each relationship focus of the previous sub-sections.

Table 3.6 Relationship Determination Questions

Relationship Type	Relationship Determination Questions
Specification	Does the item have a description? Does the item have a definition? Does the item have an explanation? Does the item have a set of instructions? Does the item have an illustration?
Elaboration	Does the description fully describe the item? Does the definition fully encompass the item? Does the explanation make assumptions? Are the set of instructions complete? How can this item be expanded or broadened?
Membership	Is this item a segment of a whole item? Is this item a member of a collection? What is this item a part-of? What components consist of this item? What phrases are in this whole activity?

Table 3.6 Relationship Determination Questions (Continued)

Relationship Type	Relationship Determination Questions
Aggregation	Which components comprise this item? What materials are used to make this item? What is part-of this item?
Generalization/ Specialization	Is the item a kind of parent item? Does the item completely include or encompass other items? Is there a broader term for this item? Is there a narrower term for this item?
Similar/ Dissimilar	Which other items are similar to this item? What serves the same purposes as this item? Which others items are opposite to this item?
Structure	What prerequisites or preconditions exist for this item? What follows this item for a given purpose? What precedes this item for a given purpose? Which items are close to this item?
Occurrence	Where else does this item appear in the domain? Where else is this item used in this system and in other systems? What are all uses of this item? Where was this item used before? Where else is the item used now? Where will this item be used later?
Modify	What can this item change into? What output results from the item's inputs? What resources and mechanisms are required to modify this item? Who can modify this item?
Transpose	How can this item be reused? How can this item be viewed differently? Can this item be used in a different context?
Influence	What items or people cause this item to be created, changed, or deleted? What items or people have control over this item?
Extrapolate	Which goals, issues, and arguments involve this item? What are the positions and statements on the item? What are the comments on this item? What are the opinions on this item? What is the rationale for this decision?

To summarize, the RAM provides a theoretical foundation to classify the complete set of relationships around any component or element within an information system. The model applies Guilford's SI theory to the computer application domain and identifies relationships utilizing a question-based elicitation technique. Chapter 4

presents a process to utilize the RAM, and provides a systematic and rigorous technique to explicitly identify and document the relationship structure of an application domain.

3.6 Summary

Software analysis and design is just as much a creative process as it is an engineering process. (Gomes et al., 2001) (Gero, 1994) (Partridge & Rowe, 1994). Creativity in software engineering is a cognitive process that analysts use to generate products to satisfy certain kinds of properties. An aspect of software product creation involves identifying what is needed to solve a problem. The ability to creatively generate a software product is a difficult process facilitated by modeling techniques. Conceptual or semantic modeling techniques discussed in this chapter (Chen, 1976) (Booch et al., 1998) (Schlaer & Mellor, 1992) (Coad & Yourdon, 1990) (Kobryn, 2000) (Boggs & Boggs, 2002) provide a strong representation of entities and attributes, but provide a weak representation of relationships. Semantic models express relationships better than traditional relational and network models due to its expressiveness of relationship constructs supported by the model (Jarvenpaa & Machessky, 1989) (Burt & Kinnuean, 1990). Although the need for models with richer semantics is widely recognized, no single approach has won general acceptance (Peckham & Maryanski, 1998). To standardize the type of relationships a semantic model should represent, a classification system of relationship types is needed.

The Relationship Analysis Model (RAM) fills this need by supplying a semantic model to classify relationships. As such, the RAM classifies all the relationships of the domain organized by focus and relationship type. Why is RAM the best semantic model

to identify the relationship structure of a problem domain? It is better than other models because it classifies the complete set of relationships, whereas other models only identify a limited subset of relationships as listed in Table 3.1 and discussed in Section 3.4. Although a limited subset is useful, one cannot say that it is categorically complete. Also, the RAM has its foundations in Guilford's Structure of Intellect (SI) (Guilford, 1967) theory to develop a classification model that encompasses the scope of human intellectual abilities in forming concepts and the relationships among concepts. In doing so, it provides a foundation to improve the process of relationship discovery and classification.

Chapter 4 applies the RAM and develops a systematic technique to relationship elicitation. A systematic process is an essential element to process improvement (Becker-Kornstaedt, 2001). A systematic approach to knowledge elicitation makes requirements gathering and problem understanding less dependent on the experience level of the process engineer (Bandinelli, 1995). A systematic approach to requirements elicitation helps to improve accuracy and provide a greater level of detail. Relationships are systematically identified and classified by applying a question-based elicitation technique (Yoo, 2000). The RAM is the only systematic approach that classifies the complete set of relationships.

CHAPTER 4

RELATIONSHIP ANALYSIS APPLIED

A significant aspect of systems analysis and design involves discovering and representing entities and their relationships. However, existing techniques leave relationship determination implicit; they are supposed to appear as a byproduct of other analysis activities. This chapter describes how to apply the Relationship Analysis Model (RAM), described in Chapter 3, to systematically elicit and document the relationship structure of an application domain. The presented technique addresses a major void in today's software engineering analysis techniques, namely relationship discovery.

Although Relationship Analysis (RA) is methodology independent, this dissertation shows the technique's effectiveness utilizing object-oriented analysis. Object-oriented analysis depicts interactions between use-cases and the actors utilizing use-case diagrams. Subsequently, class diagrams are developed to depict the relationships between the classes that implement the use-cases. However, a step is missing and the transition is too abrupt. The existing techniques leave the relationship determination implicit. RA fills this void by providing a systematic technique to determine and document the relationship structure of an application. The RA technique can be integrated into object-oriented analysis between the use-case and class diagram identification steps (Catania & Bieber, 2003). Thus, RA adds a step to the process, but provides a technique to explicitly determine and depict the application's relationship structure, thereby enhancing the analysis process.

4.1 Relationship Analysis Process (RAP)

The Relationship Analysis Process (RAP) is a rigorous and systematic technique to identify the relationship structure of an application domain. A systematic process is an essential element to process improvement (Becker-Kornstaedt, 2001). A systematic approach to knowledge elicitation makes requirements gathering and problem understanding less dependent on the experience level of the process engineer (Bandinelli, 1995). A systematic approach to requirements elicitation helps to improve accuracy and provide a greater level of detail. Process elicitation should be performed in two stages (Becker-Kornstaedt, 2001), the first stage is process familiarization and the second stage is detailed elicitation. The aim of familiarization is to obtain an overview of the general structure and is mainly used for knowledge elicitation. Detailed elicitation obtains in-depth more detailed information.

The RAP also uses two primary steps in its elicitation process. The first step utilizes use-case analysis as a way to acquire system familiarity. The process then acquires detailed knowledge from information obtained from use-cases by explicitly identifying the relationships of the system using a Relationship Analysis Template (RAT). The resulting relationship information is then depicted in a Relationship Analysis Diagram (RAD). The process consists of the following four process steps:

- Perform a use-case analysis to identify items of interest
- Isolate items of interest
- Identify the Relationship Structure utilizing the Relationship Analysis Templates (RAT)
- Graphically depict the relationships utilizing the Relationship Analysis Diagrams (RAD)

The RAP is best-realized utilizing expertise from different team members in a collaborative fashion. System analysts work collaboratively to identify the system's main

use-cases and other items of interest. As described in the use-case analysis section of the background literature, it is the process of capturing requirements from the user's point of view and helps describe what functionality is contained within the system. The identified actors and use-cases represent the high-level items of interest and in addition, use-case descriptions provide narratives in which low-level items of interest can be selected. Booch describes the identification of objects by a process of noun extraction (Booch, 1994). This technique can be employed to identify low-level items of interest. This perspective is not implementation-oriented but stresses instead what the user expects from the system. This approach to problem description helps to track the project by goals. In addition, use-case analysis increases the chances that the system being developed meets user needs and expectations thereby increasing user satisfaction and acceptance.

The identified use-cases are the feeder into the RAP. The resulting analysis explicitly identifies the relationship structure of the domain and results in a more complete and helpful analysis. As a result, the domain is thoroughly described in terms of its entities and relationships. The RAP explicitly identifies the relationship structure of an application domain and provides more information than use-case analysis alone and helps in the creation of class diagrams.

4.2 Relationship Analysis Template (RAT)

Each item of interest can be described in terms of relationships based on the RAM described in Chapter 3. Each relationship focus has its own template, outlined in Table 4.1, that can be used to document the relationships discovered during the elicitation

process. Each RAT is used to record the results of the analysis and help to track decisions (Booch et al., 1998).

Table 4.1 Generic Relationship Analysis Template

Item of Interest	Each item of interest should have a unique name suggesting its purpose.
Description	Each item of interest should describe its purpose. The description should list the source(s) for the requirements.
Focus	Unit, Collection, Comparison, System, Transformation, and Implication.
Convergent Relationship	Specification, Membership, Generalization/Specialization, Structure, Modify, and Influence.
Generic Question(s) (Optional)	The generic questions provided in Chapter 3 per the appropriate convergent relationships could be contained in this cell. It helps generate the specific question(s) to ask for a particular application domain.
Specific Question(s)	This cell contains the exact question(s) to ask to determine appropriate convergent relationships and is/are tailored to the particular application domain.
Results	The results generated from the specific question(s) are recorded in this cell.
Divergent Relationship	Elaboration, Aggregation, Similar/Dissimilar, Occurrence, Transpose, and Extrapolate.
Generic Question(s) (Optional)	The generic questions provided in Chapter 3, per the appropriate divergent relationships, could be contained in this cell. It helps generate the specific question(s) to ask for a particular application domain.
Specific Question(s)	This cell contains the exact question(s) to ask to determine divergent relationship and is/are tailored to the particular application domain.
Results	The results generated from the specific question(s) are recorded in this cell.

It is important to note that the template provides a way for analysts to communicate and document the process of discovering relationships. To this end, the template provides cells that contain brainstorming questions to help elicit and identify specific relationships. In particular, the template contains a cell that captures domain independent generic questions. These generic questions can be used to help create more domain dependent specific questions, which are also captured in the template. The

results are recorded in the template thereby systematically documenting the process and the relationships.

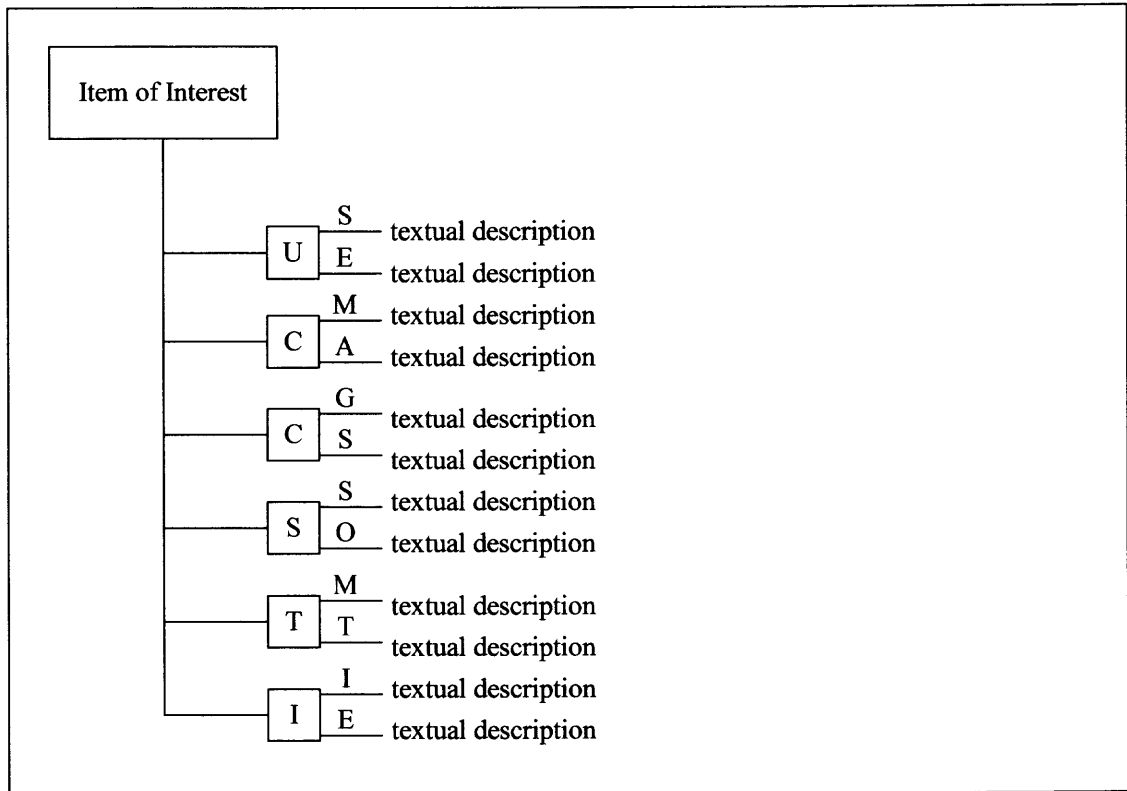
4.3 Relationship Analysis Diagram (RAD)

In addition to capturing the relationship structure in the aforementioned templates, it is also possible to present the information in a graphical representation. Although graphical diagrams and tabular representations may contain the same information, they present that information in fundamentally different ways. Graphical diagram representations emphasize spatial information, while tables emphasize symbolic information (Vessey, 1991). Some studies have demonstrated that information extraction is best-realized utilizing graphical diagrams instead of tables (Benbasat & Schroeder, 1977) (Tullis, 1981). Yet another study found that tables are superior to graphical diagrams at information extraction (Lucas, 1981). However, other studies have shown no performance differences when using either graphical diagrams or tabular representations of information (DeSanctis, 1984) (Jarvenpaa & Dickson, 1988). These confusing results may be due to the variability of task environment. The literature describes that the graphical diagram versus table controversy is due to task effects causing the unexpected results (Benbasat & Dexter, 1986) (DeSanctis, 1984) (Dickson et al., 1986) (Jarvenpaa & Dickson, 1988) (Jarvenpaa et al., 1985). Vessey points out that tasks can be divided into two types, spatial and symbolic, based on the type of information that facilitates their solution (Vessey, 1991). Performance on a task will be enhanced when there is cognitive fit between the information emphasized in the representation type and that required by

the task type; that is, when graphs support spatial tasks and when tables support symbolic tasks.

The UML toolkit supports both textual and graphical representations of information. An eventual goal is to incorporate the Relationship Analysis Process into the UML process. Therefore, both the RAT and RAD can be leveraged as the first step to that end. Whereas, the RAT is text-based, RAD will utilize boxes, connection lines, and textual descriptions to indicate the relationships and is illustrated in Figure 4.1. Both the RAT and RAD represent a new way to document the relationship structure of a domain, which will greatly assist in developing class diagrams.

Figure 4.1 Generic Relationship Analysis Diagram (RAD)



Since the purpose of the RAD is to capture the relationship structure of a specific item of interest, abbreviations are used. These abbreviations map one-to-one with RA

descriptions provided in previous sections. The relationships of each item of interest are documented in six templates and one diagram. The collection of all RAT and RAD components comprise the relationship structure of the problem domain. The RAD provides an information rich graphic, while more details can be accessed via the information recorded in the templates. Utilizing the collection of RADs of a problem domain, which depict the discovered relationships, should enhance the generation of class diagrams.

4.4 Summary

This chapter describes the components of RA, namely RAP, RAT and RAD. In addition, an example of how to perform the RAP is provided in Appendix B. The RAP is a technique to systematically and explicitly determine the relationship structure of a problem domain. The templates provide a mechanism that permits the results of the process to be documented. The diagrams generated from the information contained within the templates, provides an information rich graphic to depict the relationship structure of an item of interest. The RAP enhances the system analyst's effectiveness in the area of relationship discovery and documentation. As a result, the RAP affords analysts the opportunity to develop higher quality software applications by providing a deeper and broader understanding of the domain.

CHAPTER 5

EXPERIMENTAL DESIGN

5.1 Overview

A significant aspect of Systems Analysis and Design involves discovering and representing entities and their relationships. Guidelines exist to help identify system entities. However, no defined processes, templates, or diagrams exist to explicitly and systematically assist in eliciting and documenting relationships (Catania & Bieber, 2003). The existing techniques leave relationship determination as an implicit process, which are supposed to appear as a byproduct of the other analysis activities. Relationships constitute a large part of a domain's implicit structure. Completely understanding a domain relies on knowing how all the entities are interconnected (Bieber & Yoo, 1999) (Yoo, 2000). The experiment is designed to test if Relationship Analysis (RA) improves the process of system understanding by explicitly identifying the relationship structure of a problem domain. RA enhances the system analyst's effectiveness by providing a procedure to identify and document system relationships during the analysis phase.

The literature indicates that the object-oriented paradigm has replaced the traditional structured analysis process-oriented approach to software development (Booch et al., 1998) (Rumbaugh, 1991) (Sommerville, 2001) (Bailin, 2000). In particular, the Rational Unified Process (RUP) is the premier software development life-cycle platform for projects using object-oriented techniques (Booch et al., 1998) (Sommerville, 2001) (Boggs & Boggs, 2002). The literature also indicates that groups generate better solutions than individual solutions (Baroudi et al., 1986) (DeSanctis & Gallupe, 1987)

(Connolly et al., 1990) (Gallupe et al., 1988) (Sommerville, 2001) (Kontonya & Sommerville, 1996) (Rumbaugh, 1994) (Goguen & Linde, 1993). The experiment should show that the proposed Relationship Analysis Process (RAP), Relationship Analysis Templates (RAT), and Relationship Analysis Diagrams (RAD) are an effective technique to explicitly and systematically elicit and document relationships. The technique could be integrated into current object-oriented analysis processes to fill a gap in the current approach to identifying relationships. Neither the software engineering books (Coad & Yourdon, 1990) (Martin & Odell, 1995) (Larman, 2001) nor the comprehensive RUP (Booch et al., 1998) provide a well-defined process for relationship discovery. The RA technique fills this gap.

The beginning step to problem solving using the RUP object-oriented analysis and design technique is to perform use-case analysis. The generated use-case diagrams describe the desired functionality of the system from the user's point of view. Class diagrams are then generated from the use-case diagrams. However, the transition is too abrupt. RA helps to bridge this gap by providing a rigorous and systematic process to explicitly identify and document relationships. Knowing the domain's relationship structure will improve the effectiveness of class diagram generation.

5.2 Hypotheses

The hypotheses are designed to assess whether RA is an effective technique to explicitly identify the relationship structure of a problem domain. RA is geared at improving the analysis phase of the software development life-cycle process. Software product development is a team effort. Teams have been shown to generate better solutions than

individual solutions (Baroudi et al., 1986) (DeSanctis & Gallupe, 1987) (Connolly et al., 1990) (Gallupe et al., 1988) (Sommerville, 2001) (Kontonya & Sommerville, 1996) (Rumbaugh, 1994) (Goguen & Linde, 1993). While not an integral aspect of this study, the experiments will be conducted using groups of analysts.

In addition, the hypotheses are designed to assess whether a rigorous and systematic process helps low experienced groups achieve a similar level of quality as high experienced groups process (Amento et al., 2000) (Schenk et al., 1998) (Saleem, 1996) (Spence & Brucks, 1997) (Hillerbrand & Claiborn, 1990) (Carter et al., 1988) (Becker-Kornstaedt, 2001) (Bandinelli, 1995).

The dissertation hypotheses were derived from previous research based upon Communication Satisfaction, Solution Satisfaction, Process Satisfaction, Perceived Analysis Ability, Speed of Problem Solving, and Analysis Quality (Moody et al., 2003) (Schenk et al., 1998) (Spence & Brucks, 1997) (Hillerbrand & Claiborn, 1990) (Nosek, 1998) (Ocker et al., 1998) (Shaft & Vessey, 1998) (Cockburn, 1998) (Coleman, 1998) (Booch, 1998) (Becker-Kornstaedt, 2001) (Bandinelli, 1995) (Vessey, 1985).

The hypotheses will be tested using Pearson's r , factor analysis, Cronbach's Alpha, normality test, data transformation, non-parametric Kruskal-Wallis ANOVA, and Factorial ANOVA, which are standard statistical analysis methods to test differences of two groups (Rosenthal & Rosnow, 1991).

Hypothesis 1: Analysis Quality

- a. The class diagram generated by performing a Relationship Analysis will be more accurate and complete than groups using use-case analysis alone.
- b. The groups with high experience will generate more accurate and complete class diagrams than low experience groups.

- c. The high experience groups utilizing Relationship Analysis will generate the most accurate and complete class diagrams.

Analysis quality measures the quality of the class diagram generated by all group subjects that accomplished the same task utilizing different means. This hypothesis is based on Schenk's et al. findings that novices exhibited less detail in problem-solving tasks than did experts, resulting in lower quality (Schenk et al., 1998). In addition, a study of novice and expert programmers found that novices tended to employ weak methods for their tasks (Vessey, 1985). The results indicate that novices were unable to formulate an overall structure to the task. However, these experiments did not include a model or process to follow. This dissertation speculates that low experience groups utilizing the systematic RA process will produce documents of equal quality as high experience groups. This speculation is supported by Spence & Brucks, whom provide convincing empirical evidence that the benefits of expertise are less pronounced when analyzing and solving a problem with a well-defined technique (Spence & Brucks, 1997). In addition, another study concluded that experts, compared to novices make qualitatively different inferences in their reasoning, focus on different problem features, and thereby reason to different conclusions (Hillerbrand & Claiborn, 1990). This dissertation speculates that the very nature of a well-defined process, namely RA, will permit novices to reach the same conclusions as experts.

Hypothesis 2: Class Diagram Analysis Time

- a. The class diagram generated by first performing a Relationship Analysis will take less time to complete than groups using use-case analysis alone.
- b. The groups with high experience will need less time to generate the class diagram than low experience groups.

- c. The high experience groups utilizing Relationship Analysis will generate a class diagram in the least amount of time.

Time to generate the class diagram after the relationships are elicited measures speed of problem solving using the RA technique. This hypothesis is based on findings that experts evoke a knowledge framework that is based on prior experience that expedites problem solving (Spence & Brucks, 1997). However, Spence and Brucks have also shown that solving a problem in a more structured way can cause the performance, quality and speed of problem solving, of novices to improve significantly. Therefore, the benefits of expertise are less pronounced when solving a problem in a structured manner.

It is speculated that although RA is an additional process step in the analysis phase, it will not significantly increase the time necessary to complete the overall assignment. This is due to a highly structured series of steps to be followed. A systematic approach to knowledge elicitation makes requirements gathering, analysis, and problem understanding less dependent on the experience level of the process engineer (Bandinelli, 1995).

Hypothesis 3: Total Analysis Time

- a. The total analysis time by first performing a Relationship Analysis will take more time to complete than groups using use-case analysis alone.
- b. The groups with high experience will need less total analysis time than low experience groups.
- c. The high experience groups utilizing Relationship Analysis will need less time than low experience groups utilizing Relationship Analysis.

Similar to hypothesis 2, the total analysis time measures speed of problem solving using the RA technique. This hypothesis is based on findings that experts evoke a knowledge framework that is based on prior experience that expedites problem solving

(Spence & Brucks, 1997). However, Spence and Brucks have also shown that solving a problem in a more structured way can cause the performance, quality and speed of problem solving, of novices to improve significantly. Therefore, the benefits of expertise are less pronounced when solving a problem in a structured manner.

It is speculated that although RA is an additional process step in the analysis phase, it will not significantly increase the time necessary to complete the overall assignment. This is due to a highly structured series of steps to be followed. A systematic approach to knowledge elicitation makes requirements gathering, analysis, and problem understanding less dependent on the experience level of the process engineer (Bandinelli, 1995).

Hypothesis 4: Perceived Analysis Ability

- a. Subjects performing a Relationship Analysis will have a higher perception of their analysis ability than subjects using use-case analysis alone.
- b. Subjects with high experience will have a higher perception of their analysis ability than low experience subjects.
- c. The high experience subjects utilizing Relationship Analysis will have the highest perception of their analysis ability.

Perceived analysis ability measures how the subjects feel about their analysis ability. This hypothesis is based on prior findings that a subject's perceived ability directly impacts their intention to use a method (Moody et al., 2003). The experiment conducted by Moody et al. was to define and train all subjects on an evaluation model. The model was an extended entity relationship model called referent modeling language. Afterwards, each subject was given a problem statement case and had two weeks to develop an information model to meet the requirements of the problem case. However,

Moody's et al. experiment utilized twenty-one different problem statement cases. This dissertation used one problem statement and speculates that subjects performing the RA will have a higher level of perceived analysis ability.

Hypothesis 5: Process Satisfaction

- a. Subjects performing Relationship Analysis will be more satisfied with the analysis process than those using use-case analysis alone.
- b. Subjects with high experience will be more satisfied with the analysis process than low experience subjects.
- c. The high experience subjects utilizing Relationship Analysis will be the most satisfied with the analysis process.

Process satisfaction measures the satisfaction experienced by all group subjects that accomplished the same task utilizing different means. This measure is based upon Ocker's et al. measurement of process satisfaction with respect to modes of communication (Ocker et al., 1998). The results indicate that there were no significant differences between different modes of communication. Ocker's et al. experiment did not include a model or process to follow. This dissertation speculates that subjects utilizing a systematic process will be more satisfied with the process than subjects not using an explicit systematic process.

Hypothesis 6: Team Communication Satisfaction

- a. Subjects performing a Relationship Analysis will be more satisfied with their team communications due to the Relationship Analysis Templates than subjects using use-case analysis alone.
- b. Subjects with high experience will be more satisfied with their team communications than low experience subjects.

- c. The high experience subjects utilizing Relationship Analysis will be more satisfied with their team communications than low experience subjects utilizing Relationship Analysis.

This hypothesis is based on the adoption of use-case templates to facilitate knowledge elicitation and document that knowledge (Cockburn, 1998) (Coleman, 1998) (Booch, 1998). In addition, templates help to create a systematic process, which is an essential element to process improvement (Becker-Kornstaedt, 2001). A systematic approach to knowledge elicitation makes requirements gathering, analysis, and problem understanding less dependent on the experience level of the process engineer (Bandinelli, 1995). A systematic approach to requirements elicitation and analysis helps to improve communication and accuracy thereby improving the process. This dissertation speculates that subjects will communicate more effectively with their team members by the use of a template to facilitate the relationship elicitation process.

Hypothesis 7: Class Diagram Solution Satisfaction

- a. Subjects performing a Relationship Analysis will be more satisfied with their generated class diagram than subjects using use-case analysis alone.
- b. Subjects with high experience will be more satisfied with their generated class diagram than low experience subjects.
- c. The high experience subjects utilizing Relationship Analysis will be more satisfied with their generated class diagram than low experience subjects utilizing Relationship Analysis.

Solution satisfaction measures the satisfaction experienced by all group subjects with respect to the final class diagram that represents the solution to the assignment. The hypothesis is based upon the results of a collaborative experiment with software engineers, which indicate that experienced software engineers were personally satisfied and confident with their solution (Nosek, 1998). Also, Ocker's et al. measured solution

satisfaction between different collaborative modes of communication (Ocker et al., 1998). However, Ocker's et al. experiment did not include a model or process to follow. This dissertation speculates that subjects utilizing a systematic process will be more satisfied with their solution than subjects not using an explicit systematic process.

5.3 Method

The method of the experiment is a 2 x 2 factorial design. The two independent variables are experience and analysis tool, depicted in Table 5.1.

Table 5.1 Factorial Design Experiment

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case		
	Use-case & RA		

Therefore, the four conditions in this experiment are:

- Use-case, low experience
- Use-case & RA, low experience
- Use-case, high experience
- Use-case & RA, high experience

The use-case analysis tool represents the control group category and the treatment group represents the use-case & RA category. Thus, it is possible to measure the effects of the RA technique.

Experience has been used extensively to determine its effect on the learning process (Amento et al., 2000) (Schenk et al., 1998) (Saleem, 1996) (Spence & Brucks, 1997) (Hillerbrand & Claiborn, 1990) (Carter et al., 1988). To determine experience level, subjects completed a pre-experiment questionnaire that identified academic background, software background, and professional work experience relating to software

system analysis and design. The pre-experiment questionnaire is listed in Appendix C. Experts, John Discepola and Ronald Lazeration, divided the subjects into low and high experience based upon the criteria determined from the pre-experiment questionnaire. The low experience subjects were randomly selected and placed in a team consisting of a total of three low experience individuals. Similarly, high experience subjects were randomly selected and placed in a team consisting of a total of three high experience individuals. Therefore, each class can consist of groups of both low and high experience.

5.4 Subjects

The subjects in the experiment consisted of both undergraduate and graduate students enrolled in the College of Computing Science Department at the New Jersey Institute of Technology. Undergraduate students from CIS 390 (Analysis and System Design), CIS 490 (Design in Software Engineering), and graduate students from CIS 673 (Software Design and Production Methodology) were used. As part of the course curriculum, all students were taught how to perform use-case analysis and generate class diagrams. The subjects in the treatment group were taught Relationship Analysis.

5.5 Procedures

The experiment was conducted at the end of the semester so all subjects had some level of modeling experience. All subjects were taught how to develop use-case analysis diagrams and generate class diagrams prior to the experiment. The treatment groups were trained in Relationship Analysis. To eliminate any training effect, the control

groups were provided an equivalent enrichment topic, namely entity relationship (E/R) analysis. After the training, all groups were provided the same task to solve with their team members. All groups had one hour to create the use-case analysis diagram. This will provide all groups time to familiarize themselves with the problem domain. At the conclusion of the session, all groups were provided with an expert generated use-case analysis diagram to the problem statement. All groups used this as a basis to complete the remaining experimental steps. The control groups generated class diagrams after use-case analysis. The treatment groups performed Relationship Analysis and then generated class diagrams. This allows the effect of Relationships Analysis to be measured. All groups had one week to complete the task and submit all analysis documents and class diagrams. At the completion of the experiment, all subjects completed a post experiment questionnaire and were debriefed.

5.6 Measures

The data collected from the experiment was from post-task questionnaires elicited from the subjects. Also, each team recorded the time needed to complete the assignment. In addition, expert judges provided the quantitative quality assessment measure. Table 5.2 summarizes the measurements used for the dependent variables. These measures were derived from previous research discussed in Section 5.2.

Table 5.2 Dependent Variable Measurement Methods

Dependent Variable	Measurement
Analysis Quality	Ratings from Experts
Class Diagram Analysis Time	Recorded by Subjects
Total Analysis Time	Recorded by Subjects
Perceived Analysis Ability	Ratings from Post-Task Questionnaire
Team Communication Satisfaction	Ratings from Post-Task Questionnaire
Class Diagram Solution Satisfaction	Ratings from Post-Task Questionnaire
Process Satisfaction	Ratings from Post-Task Questionnaire

Each group was required to submit their analysis documents, which the expert judges will use to rate each group's analysis. In order to measure team communication satisfaction, group members only communicated among each other. This helps to ensure that the analyst's understanding of the domain is through communicating with only their group members.

At the completion of the process, groups submitted the following documents depending on their experimental condition.

- Relationship Analysis Template (RAT) Document
A collection of six templates for each item of interest discovered during the analysis.
- Relationship Analysis Diagram (RAD) Document
Corresponding to the six templates for each item of interest discovered during the analysis, a single graphical based diagram depicting the combined information provided by the six templates for each item of interest was generated. Therefore, the RAD document is a collection of diagrams, one for each item of interest discovered during the analysis.
- Class Diagram Document
Depicts the existence of classes and their relationships of the system. The class diagram document represents the complete class structure of the system.

5.6.1 Questionnaires

At the completion of the experimental task, a questionnaire to elicit the perception measures was administered. The post-task questionnaire elicited the perceived analysis ability, communication satisfaction, solution satisfaction, and satisfaction with the

analysis process (Moody et al., 2003) (Ocker et al., 1998). The questionnaire also measured for prior domain knowledge of the assigned task and validity of the task. In addition, each group tracked and documented the time spent creating each type of analysis document.

Software Systems Analysis Ability Questionnaire Items:

1. I am confident in my software system analysis abilities.
7. I do not use analysis techniques to develop software systems.
13. I understand the software systems analysis process.
18. I do not feel that software systems analysis is needed to develop software.

Team Communication Satisfaction Questionnaire Items:

2. Communication with my group members helped me to solve the problem.
8. I did not need to communicate with my group to solve the problem.
14. My group members communicated clearly.
19. I feel that communicating with my group did not help me to better solve the problem.
23. Performing the analysis did help me to communicate more effectively.

Class Diagram Solution Satisfaction Questionnaire Items:

3. I am satisfied with the quality of my group's class diagram document.
9. I am not confident in the group's final class diagram document.
15. I am committed to my group's final class diagram document.
20. The final class diagram document does not reflect my inputs.
24. I feel I had an equal part in my group's final class diagram document.

Process Satisfaction Questionnaire Items:

4. My group's problem solving process was efficient.
10. My group's problem solving process was coordinated.
16. My group's problem solving process was unfair.
21. My group's problem solving process was confusing.
25. My group's problem solving process was satisfying.

Prior Domain Knowledge Questionnaire Items:

5. I have already analyzed an on-line registration system for my job.
11. I have not analyzed an on-line registration system for my college studies.

Task Comprehension Questionnaire Items:

6. I feel the task was too difficult.
12. I understood the task.
17. I feel there wasn't enough time to complete the task.
22. I feel that everyone on my team understood the task.

Scale for questionnaire items 1,2,3,4,5,7,10,12,13,14,15,22,23,24,25
Strongly Agree 7 -- 6 -- 5 -- 4 -- 3 -- 2 -- 1 Strongly Disagree

Scale for questionnaire items 6,8,9,11,16,17,18,19,20,21
Strongly Agree 1 -- 2 -- 3 -- 4 -- 5 -- 6 -- 7 Strongly Disagree

5.6.2 Expert Judges

Two experts (John Discepola and Ronald Lazeration), who are professional software engineers, judged the quality of each group's systems analysis project. Expert judges have been used in many studies to evaluate quality of system design and decision-making (Shaft & Vessey, 1998) (Ocker et al., 1998). The expert judges rated the generated class diagram of all the groups.

The expert judges had their own training session to ensure that their evaluations are compatible. The expert judges used a 10-point scale whereby a 10 represents a perfect score. In order to eliminate potential bias of individual experts, each expert judge evaluated each group's class diagrams independently and the average evaluation will be computed and used as the final score. Whenever the difference in their evaluations exceeded 1 point (an acceptable 10% threshold), the two experts meet to resolve the issues and cooperatively assigned a final score.

Analysis Quality:

The system analysis quality was based on expert judges rating each group's generated class diagrams based on a 10-point scale, whereby a 10 represents a perfect score.

5.7 Task

Prior to the main experiment, three rounds of pilot studies were conducted. The goals of pilot testing are to:

- Evaluate and refine the training material
- Evaluate and refine the RAT, RAD, and RAP to ensure they are usable in an actual analysis and design environment
- Develop and evaluate the systems analysis task for the main experiment
- Validate performance measures

The first pilot test took place during the Spring 2003 semester to evaluate the training material. Students from CIS 490 (Design in Software Engineering) and CIS 491 (Computer Science Project) courses totaling 18 participants were utilized. All participants had experience with object-oriented analysis and design techniques. All 18 participants were trained on both use-case and RA techniques through hands-on exercises. The training materials are listed in Appendix B. After training, feedback from the participants was collected through small group interviews and individual questionnaires. The questionnaires are contained in Appendix C. RAT, RAD, and RAP tools and procedures were revised based on the feedback provided.

The second pilot test also took place during the Spring 2003 semester to evaluate the revised training material. Students from Bloomfield College's CMP 328 (Object-oriented Analysis and Design) course totaling 6 participants were utilized. All participants had experience with object-oriented analysis and design techniques. All 6 participants were trained on both use-case and RA techniques utilizing the revised training material. The training materials are listed in Appendix B. After training, feedback from the subjects was collected through a group interview and individual

questionnaires. The questionnaires are contained in Appendix C. The feedback indicated that RAT, RAD, RAP, tools and procedures were satisfactory.

The third pilot test was conducted during the Summer 2003 semester to evaluate the systems analysis experimental task and performance measures. Students from CIS 390 (Analysis & System Design) totaling 42 subjects were utilized. All subjects were trained on use-case analysis and class diagram creation through hands-on exercises as part of the course curriculum. In addition, all subjects were trained in RA through hands-on exercises. The training materials are listed in Appendix B. After training, the subjects were assigned to a team of 3 members. Each team will solve the experimental task listed in Appendix F. At the conclusion of the experiment, feedback from the subjects was collected through small group interviews and individual questionnaires. To validate the questionnaire, the subjects were asked to mark any unclear questions. The experimenter utilized an open-ended interview to validate the experimental task. The feedback and results indicated that the questionnaire and experimental task were satisfactory. Table 5.3 summarizes the three pilot studies.

Table 5.3 Pilot Study Summary

Pilot Test Summary		
Pilot 1	Objective	Evaluate the Relationship Analysis training material using a post-training questionnaire and group interviews.
	Schedule	Spring 2003 Semester
	Subjects	18 Undergraduate Students at NJIT CIS 490 Design in Software Engineering CIS 491 Computer Science Project
	Results	The training material contained too much theory. The overall consensus was that the subjects only wanted to be trained on how to perform the Relationship Analysis Process. Therefore, another pilot study was required to evaluate the revised Relationship Analysis training material.
Pilot 2	Objective	Evaluate the revised Relationship Analysis training material using a post-training questionnaire and a group interview.
	Schedule	Spring 2003 Semester
	Subjects	6 Undergraduate Students at Bloomfield College CMP 328 Object-oriented Analysis and Design
	Results	All students were satisfied with the training material and the Relationship Analysis process.
Pilot 3	Objective	Evaluate and validate both the systems analysis experimental task and performance measures using a pre and post- experimental task questionnaire and group interviews.
	Schedule	Summer 2003 Semester
	Subjects	42 Undergraduate Students at NJIT CIS 390 Analysis & System Design
	Results	There were no unclear questions on either the pre or post experimental questionnaire. All groups were satisfied with the experimental task.

RA's effectiveness was tested using an experimental task in the main experiment that was conducted during the Fall 2003 semester. The courses utilized were CIS 390 (Analysis and System Design), CIS 490 (Design in Software Engineering), and CIS 673 (Software Design and Production Methodology). All subjects worked on exactly the same systems analysis task, namely the systems analysis of an on-line registration system. The details are listed in Appendix F. The subjects were trained prior to

performing the experiment. The training materials are listed in Appendix B. There was one training session scheduled for the treatment groups that were taught the RA technique. The control groups were provided an enrichment topic to eliminate any training effect. All teams had one week to generate the treatment dependent documents and class diagram to the task. All teams kept track of how much time they spent performing each section of the analysis. At the second-class meeting, all teams submitted their analysis documents. All subjects completed a post-experiment questionnaire. Immediate following, all subjects were debriefed. Therefore, the entire experiment spanned a 1-week time period. To facilitate the process, the subjects assigned to the treatment condition group were able to download the RA templates from the WebBoard.

The number of subjects was 171 and divided into groups of 3, resulting in 57 groups. The low experience subjects were assigned to low experience groups randomly. The high experience subjects were assigned to high experience groups randomly. Each group, from the 4 different group types, performed the same task. This permits the pure effect of the treatments to be isolated because the difference in tasks is controlled. This will increase the internal validity of the research (Straub, 1989).

CHAPTER 6

EXPERIMENTAL RESULTS AND DATA ANALYSIS

This chapter describes the results of an experiment to measure RA's effectiveness. RA is a technique that can be used during the computer systems analysis phase to explicitly discover and document the relationship structure of a problem domain. The analysis performed was compared against a set of hypotheses to determine if the RA technique enhances the analysis process.

The main experiment took place in the Fall 2003 semester utilizing evening sections of CIS 390 (Analysis and System Design), CIS 490 (Design in Software Engineering), and CIS 673 (Software Design and Production Methodology) at the New Jersey Institute of Technology and lasted one week, whereby the first day included a training session. One week prior to the experiment, the subjects completed a pre-experiment questionnaire to determine analysis experience level. Experts then divided the subjects into high and low experience levels. The low experience subjects were randomly selected and placed in a team consisting of three low experience individuals. Similarly, high experience subjects were randomly selected and placed in a team consisting of three high experience individuals. Then each group was placed in one of the four conditions. Each team, from the four different group types, performed the same task. This permits the pure effect of the treatments to be isolated because the difference in tasks is controlled. This will increase the internal validity of the research (Straub, 1989).

Data was collected from experts who rated the quality of the class diagram generated by each group and a post-experiment questionnaire completed by the subjects. In addition, each group recorded the time they spent on completing the assignment. The analysis of this data, using statistical procedures in SAS release 8.02, is presented in this chapter.

6.1 Subject Background Information

There were a total of 171 subjects who participated in the experiment. To determine experience level, all subjects completed a pre-experiment questionnaire listed in Appendix C. The questionnaire was adopted from Prof. Eljabiri's CIS 491 (Computer Science Project) experience questionnaire. To determine experience level, experts used a decision criteria determination sheet also adopted from Prof. Eljabiri's CIS 491 course, listed in Appendix C, to rate academic background, general software background, software development background, and software engineering professional background. This section analyzes the results from the pre-experiment questionnaire and provides the details to the subject distribution.

6.1.1 Subject Pre-experiment Evaluation

The range of subject scores possible was from 26 to 203. Those below 100 were placed in low experience groups, while those above 100 were placed in high experience groups. Table 6.1 shows the scores for all subjects, grouped by course number, and the conditions in which they were placed.

Table 6.1 Subject Experience Score Details

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	<u>CIS390:</u> 48,39,39,67,49,53,76,53,62, 54,42,58,62,60,77,59,63,39, 46,41,43 <u>CIS 490:</u> 74,93,77,70,66,74,88,80,70, 71,63,75,53,40,46 <u>CIS 673:</u> 62,61,70,74,75,81,61,79,63	<u>CIS390:</u> 130,141,121,110,114,112, 119,125,121 <u>CIS490:</u> 115,109,124,115,130,116, 140,137,130,147,135,121 <u>CIS673:</u> 135,120,137,114,148,129, 118,155,149,148,118,143, 114,118,140,126,134,109
	Use-case & RA	<u>CIS390:</u> 43,50,86,36,71,42,53,66,79, 88,51,58,48,73,55,59,73,61, 58,40,69,35,34,47 <u>CIS490:</u> 70,64,44,54,53,70,57,58,42 51,96,50 <u>CIS673:</u> 75,51,72,78,60,65,66,86,39, 40,47,68	<u>CIS390:</u> 128,108,148,120,114,113, 128,146,119 <u>CIS490:</u> 116,120,129,132,124,129, 145,118,118,125,122,126, 138,127,108 <u>CIS673:</u> 119,128,106,133,139,117, 107,105,113,109,135,108, 111,128,112

Table 6.2 provides the means and standard deviation calculations for each of the conditions.

Table 6.2 Subject Experience Score Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 62.13 SD = 14.17	Mean = 127.36 SD = 12.87
	Use-case & RA	Mean = 58.98 SD = 15.14	Mean = 122.33 SD = 11.59

The mean scores of low experience subjects in the UC and UC&RA conditions were 62.13 and 58.98 respectively. In contrast, the mean scores of high experience subjects in the UC and UC&RA conditions were 127.36 and 122.33 respectively. These mean scores indicate proper subject distribution.

6.1.2 Subject Distribution

Table 6.3 provides details of the number of subjects from the various courses with either low or high experience level.

Table 6.3 Summary of Subject Distribution

Course	Low Experience	High Experience	Dropout = 9.4%
CIS 390	45	18	3
CIS 490	27	27	6
CIS 673	21	33	7
Totals	105	66	16

The subjects were placed in teams of three and Table 6.4 provides the exact group id numbers and the conditions in which they were placed.

Table 6.4 Group Identification Numbers per Condition

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	<u>CIS390 = 7 groups:</u> 390_7, 390_9, 390_12, 390_15, 390_17, 390_21, 390_23	<u>CIS390 = 3 groups:</u> 390_2, 390_4, 390_6
		<u>CIS490 = 5 groups:</u> 490_12, 490_14, 490_15, 490_18, 490_20	<u>CIS490 = 4 groups:</u> 490_1, 490_2, 490_4, 490_10
		<u>CIS673 = 3 groups:</u> 673_13, 673_16, 673_17	<u>CIS673 = 6 groups:</u> 673_1, 673_2, 673_3, 673_5, 673_8, 673_9

Table 6.4 Group Identification Numbers per Condition (Continued)

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case & RA	<u>CIS390 = 8 groups:</u> 390_10, 390_11, 390_13, 390_14, 390_16, 390_18, 390_19, 390_22	<u>CIS390 = 3 groups:</u> 390_1, 390_3, 390_5
		<u>CIS490 = 4 groups:</u> 490_13, 490_16, 490_17, 490_19	<u>CIS490 = 5 groups:</u> 490_3, 490_5, 490_7, 490_8, 490_9
		<u>CIS673 = 4 groups:</u> 673_12, 673_14, 673_15, 673_18	<u>CIS673 = 5 groups:</u> 673_4, 673_6, 673_7, 673_10, 673_11

Table 6.5 provides a summary of the total number of groups per condition, added up according to class CIS390 + CIS490 + CIS673.

Table 6.5 Summary of Total Number of Groups per Condition

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	7 + 5 + 3 = 15	3 + 4 + 6 = 13
	Use-case & RA	8 + 4 + 4 = 16	3 + 5 + 5 = 13

The distribution of subjects equated to 15 groups in condition 1, 16 groups in condition 2, 13 groups in condition 3, and 13 groups in condition 4. The subject groups were placed into one of four conditions:

- Condition 1: Use-case and Low Experience
- Condition 2: Use-case & RA and Low Experience
- Condition 3: Use-case and High Experience
- Condition 4: Use-case & RA and High Experience

Subsequent sections describe both the qualitative and quantitative results of the experiment.

6.2 Expert Judge Reliability

The two expert judges were unaware of the experimental conditions. They were both given each group's generated class diagram document and the ideal solution to the problem statement. Both judges independently evaluated each group's class diagram and assigned a score from 1 to 10, whereby 10 represented a perfect score. Afterwards, judges meet and compared the scores assigned, whenever the difference in their evaluations exceeded 1 point, the two experts resolved the issues and cooperatively assigned a final score. Thereby, the judges evaluated the quality of the class diagram generated by each group.

The results from both judges were evaluated to determine if they were trained properly and to determine the reliability of their quality grade, a dependent variable. An inter-judge reliability check was performed using a bivariate Pearson's r test both before and after the judge's meet. The results indicate a significant correlation of 0.858 at the 0.0001 level before meeting (Table 6.6) and a significant correlation of 0.943 at the 0.0001 level after meeting (Table 6.7).

Table 6.6 Correlation Between Judges Before Meeting

Judge Identifier	N	Mean	Std. Dev.	Association
Judge 1	57	7.333	1.629	r=0.8584 p=0.0001
Judge 2	57	6.947	1.736	

Table 6.7 Correlation Between Judges After Meeting

Judge Identifier	N	Mean	Std. Dev.	Association
Judge 1	57	7.211	1.687	r=0.9430 p=0.0001
Judge 2	57	6.982	1.674	

6.3 Experiment Hypotheses Analysis

This section describes the evaluation of the data collected. The evaluation involved factor analysis, Cronbach's Alpha, normality test, data transformation, non-parametric Kruskal-Wallis ANOVA, and Factorial ANOVA. The results were compared against the hypotheses described in Chapter 5.

To determine if the data is normally distributed, we used SAS to run the Kolmogorov-Smirnov normality test. If the test results in $p > 0.05$, then the data is normally distributed. SAS offers five transform functions that can be used to normalize the data. To be comprehensive, each transform function was tried to normalize the data and the results are listed in the sub-sections that analyze and describe each dependent variable.

The results of data transformation did not normalize all the data. In those cases, to determine the interaction effect among the two experience level groups, we divide the data into two groups, namely low and high, and apply a non-parametric method to both groups. In other words, this compares the difference between UC and UC&RA in low and high experience groups separately. If the "differences" are significantly different, then there is an interaction effect. For example, suppose the difference between UC and UC&RA in the low experience groups is 4 and the difference between UC and UC&RA in high experience groups is 1. Then, the effect of UC/UC&RA is bigger in low experience groups than high experience groups, which means that the two variables (Analysis Tool: UC/UC&RA and Experience: Low/High) interact with each other. The details are listed in the various sub-sections.

Table 6.8 provides an overview of the results. The sub-sections describe the details of these results.

Table 6.8 Results Overview

Dependent Variable	Analysis Tool	Experience Level		Distribution
		Low	High	
Quality	UC	m = 5.87 σ = 2.53	m = 6.81 σ = 0.97	not normal p = 0.01
	UC&RA	m = 7.78 σ = 0.55	m = 7.96 σ = 0.78	
Class Diagram Time	UC	m = 11.97 σ = 4.34	m = 14.29 σ = 3.56	normal after transformation p = 0.15
	UC&RA	m = 11.39 σ = 3.55	m = 11.90 σ = 4.03	
Total Analysis Time	UC	m = 161 σ = 114.16	m = 214.62 σ = 90.66	normal p = 0.13
	UC&RA	m = 362.19 σ = 158.79	m = 374.23 σ = 154.38	
Perceived Analysis Ability	UC	m = 6.42 σ = 2.64	m = 5.21 σ = 2.60	not normal p = 0.01
	UC&RA	m = 6.92 σ = 2.87	m = 5.67 σ = 2.92	
Process Satisfaction	UC	m = 50.76 σ = 10.46	m = 55.21 σ = 11.13	not normal p = 0.01
	UC&RA	m = 54.31 σ = 8.64	m = 53.59 σ = 8.76	

Abbreviations to Table 6.8:

UC: use-case

UC&RA: use case and relationship analysis

M: mean

σ : standard deviation

6.3.1 Analysis Quality Grade Variable

Hypothesis 1 involved analysis quality. The hypotheses of the main and interaction effects are:

- 1a. The class diagram generated by performing a Relationship Analysis will be more accurate and complete than groups using use-case analysis alone.

1b. The groups with high experience will generate more accurate and complete class diagrams than low experience groups.

1c. The high experience groups utilizing Relationship Analysis will generate the most accurate and complete class diagrams.

Table 6.9 provides the details of quality grades determined by the expert judges.

The zero scores are not dropout, but the group's actual quality score.

Table 6.9 Quality Grade Details

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	<u>CIS390:</u> 7, 0, 6.5, 0, 6.5, 5.5, 7 <u>CIS490:</u> 6, 7, 5, 6.5, 7 <u>CIS673:</u> 8, 8, 8	<u>CIS390:</u> 6, 7.5, 6.5 <u>CIS490:</u> 6.5, 7.5, 8, 7.5 <u>CIS673:</u> 5, 5.5, 8, 7.5, 7, 6
	Use-case & RA	<u>CIS390:</u> 7.5, 8, 8, 7, 8.5, 8, 7, 7.5 <u>CIS490:</u> 7, 8, 8, 7 <u>CIS673:</u> 8, 8, 8.5, 8.5	<u>CIS390:</u> 8, 7.5, 6.5 <u>CIS490:</u> 8, 8, 8, 8.5, 8 <u>CIS673:</u> 9, 8, 7, 9.5, 7.5

The analysis quality grade is not normally distributed as indicated in Table 6.10.

Table 6.10 Quality Grade Normality Test Details

Data Transformation	Kolmogorov-Smirnov Normality Test
None	$p < 0.01$
Log(Y)	$p < 0.01$
Sqrt(Y)	$p < 0.01$
1/Y	$p < 0.01$
Y*Y	$p < 0.01$
Exp(Y)	$p < 0.01$

Table 6.11 provides the mean and standard deviation calculations for each of the conditions.

Table 6.11 Quality Grade Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 5.87 SD = 2.53	Mean = 6.81 SD = 0.97
	Use-case & RA	Mean = 7.78 SD = 0.55	Mean = 7.96 SD = 0.78

Main Effect 1:

The analysis tool independent variable (IV1) main effect shows a significant effect at alpha = 0.05 level ($p=0.0001$) and mean score of 38.36 and 19.30 (Table 6.12) for UC&RA and UC respectively.

Table 6.12 Quality Grade Main Effect 1 (Analysis Tool)

Independent Variable	N	Mean	Significance
Use-case	28	19.303	0.0001
Use-case & Relationship Analysis	29	38.362	

The results shown in Table 6.12 support H 1a and indicate that this variable is statistically significant at alpha = 0.05 level. The mean score of those using RA (38.36) is much better than not using RA (19.30) and indicate that RA significantly improves analysis quality.

Main Effect 2:

The experience level independent variable (IV2) main effect does not show a significant effect at alpha = 0.05 level ($p=0.5892$) and mean scores of 30.27 and 27.94 (Table 6.13) for high and low experience respectively.

Table 6.13 Quality Grade Main Effect 2 (Experience Level)

Independent Variable	N	Mean	Significance
High Experience	26	30.269	0.5892
Low Experience	31	27.935	

The results shown in Table 6.13 do not support H 1b and indicate that this variable is not statistically significant at $\alpha = 0.05$ level. This is very good because the mean scores of Table 6.11 show that UC&RA with low experience (7.78) is higher than the mean score of UC with high experience (6.81) indicating an interaction effect. Although this finding is inclusive, it suggests that low experience analysts utilizing RA could be more effective than experience analysts without RA.

Interaction Effect:

The interaction effect between analysis tool and low experience level does show a significant effect at the $\alpha = 0.05$ level ($p=0.0003$).

Table 6.14 Quality Grade Interaction Effect (Part 1)

Independent Variable	N	Mean	Significance
Use-case	15	10.433	0.0003
Use-case & Relationship Analysis	16	21.219	

The interaction effect between analysis tool and high experience level does show a significant effect at the $\alpha = 0.05$ level ($p=0.0017$).

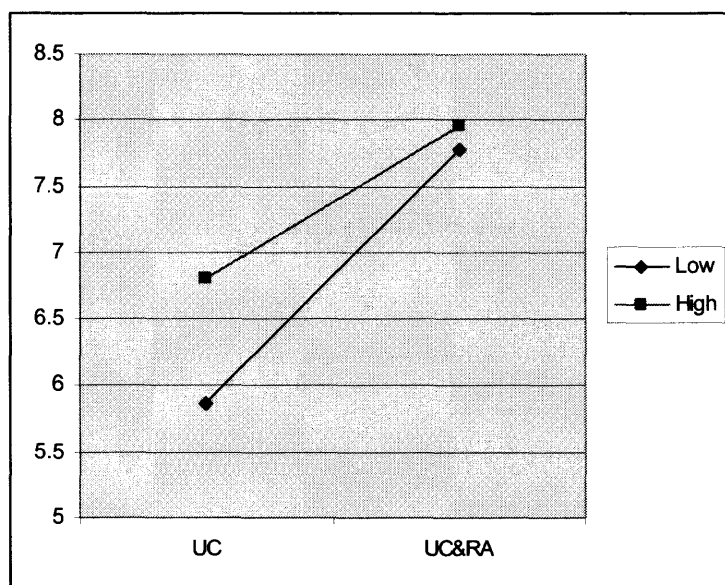
Table 6.15 Quality Grade Interaction Effect (Part 2)

Independent Variable	N	Mean	Significance
Use-case	13	9.192	0.0017
Use-case & Relationship Analysis	13	17.808	

The results shown in Tables 6.14 and 6.15 support H 1c and indicate that this variable is statistically significant at the $\alpha = 0.05$ level. The mean score difference between UC and UC&RA in low experience groups is 10.79 (21.22-10.43 from Table

6.14). In contrast, the mean score difference between UC and UC&RA in high experience groups is 8.62 (17.81-9.19 from Table 6.15). Thus, the effect of UC and UC&RA is bigger in low experience groups than high experience groups. Also, the mean score of those using RA is much better than not using RA for both groups. This represents a positive synergistic effect and suggests that low experience analysts utilizing RA could be more effective than high experience analysts without RA. Figure 6.1 depicts the quality grade for the high and low experience level groups.

Figure 6.1 Quality Grade for Groups



6.3.2 Class Diagram Analysis Time Generation Variable

Hypothesis 2 involves time needed to create the class diagram from the analysis. The hypotheses of the main and interaction effects are:

- 2a. The class diagram generated by first performing a Relationship Analysis will take less time to complete than groups using use-case analysis alone.
- 2b. The groups with high experience will need less time to generate the class diagram than low experience groups.

2c. The high experience groups utilizing Relationship Analysis will generate a class diagram in the least amount of time.

Table 6.16 provides the details of time needed in minutes for each group to complete the class diagram as indicated by the groups. Each group recorded the time spent developing the class diagram and provided the time on the cover sheet of the project submission.

Table 6.16 Class Diagram Time Details

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	<u>CIS390:</u> 330, 195, 60, 45, 150, 315, 180	<u>CIS390:</u> 70, 300, 285
		<u>CIS490:</u> 90, 90, 30, 90, 90	<u>CIS490:</u> 300, 120, 285, 120
		<u>CIS673:</u> 420, 180, 150	<u>CIS673:</u> 270, 180, 195, 240, 335, 90
	Use-case & RA	<u>CIS390:</u> 105, 210, 30, 195, 90, 30, 150, 240	<u>CIS390:</u> 45, 60, 315
		<u>CIS490:</u> 150, 75, 120, 255	<u>CIS490:</u> 360, 90, 165, 165, 180
		<u>CIS673:</u> 90, 285, 180, 60	<u>CIS673:</u> 120, 240, 115, 150, 30

The class diagram analysis time data is normalized using the Sqrt(Y) transformation as indicated in Table 6.17.

Table 6.17 Class Diagram Time Normality Test Details

Data Transformation	Kolmogorov-Smirnov Normality Test
None	$p < 0.03$
Log(Y)	$p < 0.05$
Sqrt(Y)	$p < 0.15$
1/Y	$p < 0.01$
Y*Y	$p < 0.01$
Exp(Y)	Function did not compute due to large values.

Table 6.18 provides the mean and standard deviation calculations for each of the conditions.

Table 6.18 Class Diagram Time Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 11.97 SD = 4.34	Mean = 14.29 SD = 3.56
	Use-case & RA	Mean = 11.39 SD = 3.55	Mean = 11.90 SD = 4.03

Analysis Tool:

$F = 2.11$ $p = 0.1520$

Experience Level:

$F = 1.91$ $p = 0.1732$

Analysis Tool x Experience Level

$F = 0.78$ $p = 0.3801$

The results shown in Table 6.18 do not support H 2a, H 2b, and H 2c and indicate that this variable is not significant at the $\alpha = 0.05$ level. Although not significant, it is still interesting to compare the means of Table 6.18. The mean times of the UC&RA for low (11.39) and high (11.90) experience groups are lower than the mean times of UC for low (11.97) and high (14.29) experience groups respectively. Therefore, it suggests that first performing RA does decrease the time needed to generate the class diagram. However, the results are inconclusive since the results have no significance.

6.3.3 Total Analysis Time Generation Variable

Hypothesis 3 involves total analysis time to complete the assignment. The hypotheses of the main and interaction effects are:

- 3a. The total analysis time by first performing a Relationship Analysis will take more time to complete than groups using use-case analysis alone.
- 3b. The groups with high experience will need less total analysis time than low experience groups.
- 3c. The high experience groups utilizing Relationship Analysis will need less time than low experience groups utilizing Relationship Analysis.

Table 6.19 provides the details of total analysis time to complete the assignment as indicated by the groups. Each group recorded the total time spent to complete the analysis assignment and provided the time on the cover sheet of the project submission.

Table 6.19 Total Analysis Time Details

		Low	High
Analysis Tool	Use-case	<u>CIS390:</u> 330, 195, 60, 45, 150, 315, 180	<u>CIS390:</u> 70, 300, 285
		<u>CIS490:</u> 90, 90, 30, 90, 90	<u>CIS490:</u> 300, 120, 285, 120
		<u>CIS673:</u> 420, 180, 150	<u>CIS673:</u> 270, 180, 195, 240, 335, 90
	Use-case & RA	<u>CIS390:</u> 300, 390, 150, 475, 150, 210, 240, 420	<u>CIS390:</u> 180, 180, 510
		<u>CIS490:</u> 600, 145, 360, 600	<u>CIS490:</u> 600, 270, 435, 285, 555
		<u>CIS673:</u> 330, 615, 420, 390	<u>CIS673:</u> 390, 570, 245, 435, 210

The total time variable has a Kolmogorov-Smirnov p value of 0.13 indicating normal data distribution.

Table 6.20 depicts the mean and standard deviation calculations for each of the conditions.

Table 6.20 Total Analysis Time Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 161 SD = 114.16	Mean = 214.62 SD = 90.66
	Use-case & RA	Mean = 362.19 SD = 158.79	Mean = 374.23 SD = 154.38

Analysis Tool:

F = 25.79 p = 0.0001

Experience Level:

F = 0.85 p = 0.3596

Analysis Tool x Experience Level

F = 0.34 p = 0.5609

The results shown in Table 6.20 support the main effect, hypothesis H 3a and indicate that this variable is statistically significant at the alpha = 0.05 level. The mean scores indicate that the UC&RA low and high experience groups took significantly more time to complete the analysis than the UC low and high experience groups, specifically in minutes 362 vs. 161 and 374 vs. 214. This is also a good indicator that the groups actually performed the RA.

However, the results shown in Table 6.20 do not support H 3b and H 3c and indicate that this variable is not statistically significant at the alpha = 0.05 level. It is still interesting to compare the mean scores in minutes of UC&RA for low and high experience. These are 362 and 374 respectively. This indicates that the time to perform the analysis was about the same and independent of experience. This is actually good

because it indicates that the RAP can be equally applied regardless of experience level. However, the results are inconclusive since the results have no significance.

6.3.4 Relationship Analysis Time Variable

The subjects were also asked to record the time needed to elicit relationships using the templates. The objective was to determine, if both low and high experience level groups need approximately the same time to identify and document the relationships. There is no hypothesis to this measure since there is no control. Table 6.21 provides the details of time needed to complete the RA as indicated by the groups.

Table 6.21 Relationship Elicitation Time Details

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	-	-
	Use-case & RA	<u>CIS390:</u> 195, 180, 120, 280, 60, 180, 90, 180 <u>CIS490:</u> 450, 70, 240, 345 <u>CIS673:</u> 240, 330, 240, 330	<u>CIS390:</u> 135, 120, 195 <u>CIS490:</u> 240, 180, 270, 120, 375 <u>CIS673:</u> 270, 330, 130, 285, 180

The total time variable has a Kolmogorov-Smirnov p value of 0.15 indicating normal data distribution.

Table 6.22 depicts the mean and standard deviation calculations for each of the conditions.

Table 6.22 Relationship Elicitation Time Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	-	-
	Use-case & RA	Mean = 220.63 SD = 108.92	Mean = 217.70 SD = 84.08

Experience Level:

F = 0.01 p = 0.9371

The mean scores in minutes of low and high experience level groups are 220 and 217 respectively. This indicates that both groups need approximately the same time to identify and document the relationships of the problem domain. Therefore, the time to perform the analysis was about the same and independent of experience. This is actually good because it indicates that the RAP can be equally applied regardless of experience level. However, the results shown in Table 6.22 indicate that this variable is not statistically significant at $\alpha = 0.05$ level.

6.4 Post-experiment Questionnaire Evaluation

The post-experiment questionnaire, Appendix C, was evaluated using Factor Analysis and Cronbach's Alpha to determine the factors. The resulting factors were further analyzed using normality test, data transformation, and non-parametric Kruskal-Wallis ANOVA.

6.4.1 Factor Analysis

The results from Factor Analysis using the principal components factoring method and varimax rotation method yielded three factors.

- Factor 1 measured satisfaction and included questions 2,3,4,10,14,15,23,24,25. This contains 9 of the 15 satisfaction questions and no other question types.
- Factor 2 measured perceived analysis ability and included questions 7,18. This contains 2 of the 4 perceived analysis ability questions and no other question types.
- Factor 3 measured task comprehension and included questions 12,22. This contains 2 of the 4 task comprehension questions and no other question types.

Calculating Cronbach's Alpha for them tested the reliabilities of these factors.

Satisfaction achieved a Cronbach's Alpha of .917 and indicates a high internal consistency for the satisfaction questions of the questionnaire. Perceived analysis ability achieved a Cronbach's Alpha of .486 and indicates a very low internal consistency for the perceived analysis ability questions of the questionnaire. Task validation achieved a Cronbach's Alpha of .667 and indicates good internal consistency for the task comprehension questions of the questionnaire. A Cronbach's Alpha of 0.65 and above has been used by other researchers as a measure of good internal consistency for sets of factored items (Lederer et al., 1998) (Barki & Hartwick, 1994). Table 6.23 is a summary of the results.

Table 6.23 Summary of Cronbach's Alpha, Questions, and Factors

Constructs (Cronbach's Alpha)	Questions	Factors (Variance Explained = 66.25%)		
		1	2	3
Process Satisfaction (.917)	Communication with my group members helped me to solve the problem. (Q2)	.873	-.112	.025
	I am satisfied with the quality of my group's class diagram document. (Q3)	.842	-.004	.199
	My group's problem solving process was efficient. (Q4)	.840	-.159	-.049
	My group's problem solving process was coordinated. (Q10)	.820	-.237	-.038
	My group members communicated clearly. (Q14)	.817	-.109	.057

Table 6.23 Summary of Cronbach's Alpha, Questions, and Factors (Continued)

Constructs (Cronbach's Alpha)	Questions	Factors (Variance Explained = 66.25%)		
		1	2	3
Process Satisfaction (.917)	I am committed to my group's final class diagram document. (Q15)	.786	.024	.269
	Performing the analysis did help me to communicate more effectively. (Q23)	.718	-.061	.169
	I feel I had an equal part in my group's final class diagram document. (Q24)	.576	-.257	-.148
	My group's problem solving process was satisfying. (Q25)	.505	-.475	-.189
Perceived Analysis Ability (.486)	I do not use analysis techniques to develop software systems. (Q7)	.055	.832	.705
	I do not feel that software systems analysis is needed to develop software. (Q18)	-.105	.724	.496
Task Comprehension (.667)	I understood the task. (Q12)	-.048	-.048	.746
	I feel that everyone on my team understood the task. (Q22)	-.055	-.055	.498

To complete the analysis, each of the factors was further analyzed using normality test, data, transformation, and non-parametric Kruskal-Wallis ANOVA. The results were compared against the hypotheses described in Chapter 5.

6.4.2 Satisfaction

The original hypotheses included the following three that measured satisfaction.

Hypothesis 5: Process Satisfaction

5a. Subjects performing Relationship Analysis will be more satisfied with the analysis process than those using use-case analysis alone.

5b. Subjects with high experience will be more satisfied with the analysis process than low experience subjects.

5c. The high experience subjects utilizing Relationship Analysis will be the most satisfied with the analysis process.

Hypothesis 6: Team Communication Satisfaction

6a. Subjects performing a Relationship Analysis will be more satisfied with their team communications due to the Relationship Analysis Templates than subjects using use-case analysis alone.

6b. Subjects with high experience will be more satisfied with their team communications than low experience subjects.

6c. The high experience subjects utilizing Relationship Analysis will be more satisfied with their team communications than low experience subjects utilizing Relationship Analysis.

Hypothesis 7: Class Diagram Solution Satisfaction

7a. Subjects performing a Relationship Analysis will be more satisfied with their generated class diagram than subjects using use-case analysis alone.

7b. Subjects with high experience will be more satisfied with their generated class diagram than low experience subjects.

7c. The high experience subjects utilizing Relationship Analysis will be more satisfied with their generated class diagram than low experience subjects utilizing Relationship Analysis.

However, using the results from factor analysis, it is only possible to use process satisfaction factor for the “satisfaction” variable, namely satisfied with the analysis process. This makes sense because the nature of the satisfaction questions focus on process satisfaction. For example, satisfied with the communications process and satisfied with the generated class diagram process are similar to process satisfaction in general. This is why the factor analysis process merged these questions together to form a single factor. Therefore, the results were compared against the process satisfaction hypotheses. Therefore, the questionnaire did not provide data on team communication satisfaction or class diagram solution satisfaction. The resulting satisfaction variable is not normally distributed as indicated in Table 6.24.

Table 6.24 Satisfaction Normality Test Details

Data Transformation	Kolmogorov-Smirnov Normality Test
None	$p < 0.01$
Log(Y)	$p < 0.01$
Sqrt(Y)	$p < 0.01$
1/Y	$p < 0.01$
Y*Y	$p < 0.01$
Exp(Y)	$p < 0.01$

Table 6.25 provides the mean and standard deviation calculations for each of the conditions.

Table 6.25 Satisfaction Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 50.76 SD = 10.46	Mean = 55.21 SD = 11.13
	Use-case & RA	Mean = 54.31 SD = 8.64	Mean = 53.59 SD = 8.76

Main Effect 1:

The analysis tool independent variable (IV1) main effect does not show significant effect at the $\alpha = 0.05$ level ($p=0.4587$).

Table 6.26 Satisfaction Main Effect 1 (Analysis Tool)

Independent Variable	N	Mean	Significance
Use-case	84	85.601	0.4587
Use-case & Relationship Analysis	87	86.385	

The results shown in Table 6.26 do not support H 5a and indicate that this variable is not statistically significant at $\alpha = 0.05$ level. However, it is still interesting to compare the means of Table 6.26. The mean score of UC&RA is 86.39 and the mean score of UC only is 85.60 indicating that subjects performing RA were more satisfied.

Although this finding is inconclusive, it suggests that subjects using RA have a higher level of process satisfaction than subjects using use-case alone.

Main Effect 2:

The experience level independent variable (IV2) main effect does show a significant effect at the $\alpha = 0.05$ level ($p=0.0536$) and mean score of 92.65 and 80.42 for high and low experience respectfully.

Table 6.27 Satisfaction Main Effect 2 (Experience Level)

Independent Variable	N	Mean	Significance
High Experience	78	92.654	0.0536
Low Experience	93	80.419	

The results shown in Table 6.27 do support H 5b and indicate that this variable is statistically significant at the $\alpha = 0.05$ level, when rounded to the nearest hundredths. Table 6.27 shows the mean scores of high and low experience are 92.65 and 80.42 respectively. These results indicate that high experience subjects are more satisfied with the process than low experience groups.

Interaction Effect:

The interaction effect between analysis tool and low experience level does not show a significant effect at the $\alpha = 0.05$ level ($p=0.0624$).

Table 6.28 Satisfaction Interaction Effect (Part 1)

Independent Variable	N	Mean	Significance
Use-case	45	42.567	0.0624
Use-case & Relationship Analysis	48	51.156	

The interaction effect between analysis tool and high experience level does not show a significant effect at the $\alpha = 0.05$ level ($p=0.0673$).

Table 6.29 Satisfaction Interaction Effect (Part 2)

Independent Variable	N	Mean	Significance
Use-case	39	43.333	0.0673
Use-case & Relationship Analysis	39	35.667	

The results shown in Tables 6.28 and 6.29 do not support H 5c and indicate that this variable is not statistically significant at alpha = 0.05 level. However, it is still interesting to compare the mean scores. The mean score difference between UC and UC&RA in low experience subjects is 8.59 (51.16-42.57 from Table 6.28). In contrast, the mean score difference between UC and UC&RA in high experience subjects is 7.66 (43.33-35.67 from Table 6.29). Thus, the effect of UC and UC&RA is bigger in low experience subjects than high experience subjects. Although this finding is inconclusive, it suggests that low experience analysts utilizing RA could be more satisfied than high experience analysts without RA. This could be due to the systematic process afforded by the RA process.

6.4.3 Analysis Ability

Hypothesis 4 involves perceived analysis ability, measured after the experiment. As discussed in Section 6.4.2, perceived analysis ability earned a very low Cronbach's Alpha, namely .486. Consequently, the results should not be considered significant. The hypotheses of the main and interaction effects are:

- 4a. Subjects performing a Relationship Analysis will have a higher perception of their analysis ability than subjects using use-case analysis alone.
- 4b. Subjects with high experience will have a higher perception of their analysis ability than low experience subjects.

4c. The high experience subjects utilizing Relationship Analysis will have the highest perception of their analysis ability.

The perceived analysis ability variable is not normally distributed as indicated in Table 6.30.

Table 6.30 Analysis Ability Normality Test Details

Data Transformation	Kolmogorov-Smirnov Normality Test
None	$p < 0.01$
Log(Y)	$p < 0.01$
Sqrt(Y)	$p < 0.01$
1/Y	$p < 0.01$
Y*Y	$p < 0.01$
Exp(Y)	$p < 0.01$

Table 6.31 provides the mean and standard deviation calculations for each of the conditions.

Table 6.31 Analysis Ability Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 6.42 SD = 2.64	Mean = 5.21 SD = 2.60
	Use-case & RA	Mean = 6.92 SD = 2.87	Mean = 5.67 SD = 2.92

Main Effect 1:

The analysis tool independent variable (IV1) main effect does not show significant effect at the alpha = 0.05 level ($p=0.1491$).

Table 6.32 Analysis Ability Main Effect 1 (Analysis Tool)

Independent Variable	N	Mean	Significance
Use-case	84	82.006	0.1491
Use-case & Relationship Analysis	87	89.856	

The results shown in Table 6.32 do not support H 4a and indicate that this variable is not statistically significant at $\alpha = 0.05$ level. However, Table 6.32 shows the mean score of subjects performing UC&RA is 89.86 and the mean score of subjects performing UC only is 82.01 indicating that subjects performing RA had a higher level of perceived analysis ability. Although this finding is inconclusive, it suggests that subjects using RA have a higher level of perceived ability than those using use-case alone.

Main Effect 2:

The experience level independent variable (IV2) main effect does show significant effect at the $\alpha = 0.05$ level ($p=0.0015$).

Table 6.33 Analysis Ability Main Effect 2 (Experience Level)

Independent Variable	N	Mean	Significance
High Experience	78	73.769	0.0015
Low Experience	93	96.258	

The results shown in Table 6.33 indicate that this variable is statistically significant at $\alpha = 0.05$ level. However H 4b is not supported because Table 6.33 shows the mean scores high experience subjects and low experience subjects are 73.77 and 96.26 respectively. The mean scores indicate that low experience subjects are more confident in their perceived analysis ability than high experience subjects. This is counter-intuitive and perhaps due to the systematic process provided by RA, which could cause low experience subjects to be confident in their analysis abilities. Analyzing the interaction effect may provide further insight and is described next.

Interaction Effect:

The interaction effect between analysis tool and low experience level does not show a significant effect at the $\alpha = 0.05$ level ($p=0.2025$).

Table 6.34 Analysis Ability Interaction Effect (Part 1)

Independent Variable	N	Mean	Significance
Use-case	45	44.611	0.2025
Use-case & Relationship Analysis	48	49.239	

The interaction effect between analysis tool and high experience level does not show a significant effect at the $\alpha = 0.05$ level ($p=0.2497$).

Table 6.35 Analysis Ability Interaction Effect (Part 2)

Independent Variable	N	Mean	Significance
Use-case	39	37.782	0.2497
Use-case & Relationship Analysis	39	41.218	

The results shown in Tables 6.34 and 6.35 do not support H 4c and indicate that this variable is not statistically significant at $\alpha = 0.05$ level. However, it is still interesting to compare the means. The mean score difference between UC and UC&RA in low experience subjects is 4.63 (49.24-44.61 from Table 6.34). In contrast, the mean score difference between UC and UC&RA in high experience subjects is 3.44 (41.22-37.78 from Table 6.35). Thus, the effect of UC and UC&RA is slightly bigger in low experience subjects than high experience subjects. Although this finding is inconclusive, it suggests that low experience analysts using RA could be more confident with their analysis ability than high experience analysts without RA. This could be due to the systematic process provided by the RA process.

6.4.4 Task Comprehension

Further analysis was performed on task comprehension questions (12,22) of the questionnaire to measure whether all subjects understood the task equally.

The task validation variable is not normally distributed as indicated in Table 6.36.

Table 6.36 Task Comprehension Normality Test Details

Data Transformation	Kolmogorov-Smirnov Normality Test
None	$p < 0.01$
Log(Y)	$p < 0.01$
Sqrt(Y)	$p < 0.01$
1/Y	$p < 0.01$
Y*Y	$p < 0.01$
Exp(Y)	$p < 0.01$

Table 6.37 provides the mean and standard deviation calculations for each of the conditions.

Table 6.37 Task Comprehension Mean and Standard Deviation Calculations

2 x 2 Factorial Design		Experience	
		Low	High
Analysis Tool	Use-case	Mean = 10.16 SD = 2.71	Mean = 11.15 SD = 2.59
	Use-case & RA	Mean = 10.71 SD = 2.58	Mean = 11.95 SD = 2.20

Main Effect 1:

The analysis tool independent variable (IV1) main effect does show a significant effect at the $\alpha = 0.05$ level ($p=0.0547$).

Table 6.38 Task Comprehension Main Effect 1 (Analysis Tool)

Independent Variable	N	Mean	Significance
Use-case	84	79.863	0.0547
Use-case & Relationship Analysis	87	91.925	

The results shown in Table 6.38 indicate that this variable is statistically significant at $\alpha = 0.05$ level, when rounded to the nearest hundredths. Table 6.38 shows the mean score of UC&RA is 91.93 and the mean score of UC only is 79.86 indicating that the subjects using RA understood the task more than subjects not using RA. This could be due to the systematic process provided by the RA process.

Main Effect 2:

The experience level tool independent variable (IV2) main effect does show a significant effect at the $\alpha 0.05$ level ($p=0.0012$).

Table 6.39 Task Comprehension Main Effect 2 (Experience Level)

Independent Variable	N	Mean	Significance
High Experience	78	98.378	0.0012
Low Experience	93	75.618	

The results shown in Table 6.39 indicate that this variable is statistically significant at $\alpha = 0.05$ level. Table 6.39 shows the mean score of high experience subjects is 98.38 and the mean score of low experience subjects is 75.62 indicating that the subjects with high experience understood the task more than low experience subjects. This finding suggests experience level can affect problem understanding.

Interaction Effect:

The interaction effect between analysis tool and low experience level does not show a significant effect at the $\alpha = 0.05$ level ($p=0.1607$).

Table 6.40 Task Comprehension Interaction Effect (Part 1)

Independent Variable	N	Mean	Significance
Use-case	45	44.156	0.1607
Use-case & Relationship Analysis	48	49.667	

The interaction effect between analysis tool and high experience level does not show a significant effect at the alpha = 0.05 level ($p=0.0929$).

Table 6.41 Task Comprehension Interaction Effect (Part 2)

Independent Variable	N	Mean	Significance
Use-case	39	36.167	0.0929
Use-case & Relationship Analysis	39	42.833	

The results shown in Tables 6.40 and 6.41 indicate that this variable is not statistically significant at alpha = 0.05 level. However, it is still interesting to compare the means. The mean score difference between UC and UC&RA in low experience subjects is 5.51 (49.67-44.16 from Table 6.40). In contrast, the mean score difference between UC and UC&RA in high experience subjects is 6.66 (42.83-36.17 from Table 6.41). Thus, the effect of UC and UC&RA is slightly bigger in high experience subjects than low experience subjects. Although this finding is inconclusive, it suggests that high experience analysts have a higher level of problem understanding.

6.5 Summary of Hypotheses Analysis

Table 6.42 shows a summary of the hypotheses results of the experiment.

Table 6.42 Hypotheses Summary

Hypotheses		Result
H 1a	The class diagram generated by performing a Relationship Analysis will be more accurate and complete than groups using use-case analysis alone.	Supported
H 1b	The groups with high experience will generate more accurate and complete class diagrams than low experience groups.	Not Supported
H 1c	The high experience groups utilizing Relationship Analysis will generate the most accurate and complete class diagrams.	Supported
H 2a	The class diagram generated by first performing a Relationship Analysis will take less time to complete than groups using use-case analysis alone.	Not Supported
H 2b	The groups with high experience will need less time to generate the class diagram than low experience groups.	Not Supported
H 2c	The high experience groups utilizing Relationship Analysis will generate a class diagram in the least amount of time.	Not Supported
H 3a	The total analysis time by first performing a Relationship Analysis will take more time to complete than groups using use-case analysis alone.	Supported
H 3b	The groups with high experience will need less total analysis time than low experience groups	Not Supported
H 3c	The high experience groups utilizing Relationship Analysis will need less time than low experience groups utilizing Relationship Analysis.	Not Supported
H 4a	Subjects performing a Relationship Analysis will have a higher perception of their analysis ability than subjects using use-case analysis alone.	Not Supported
H 4b	Subjects with high experience will have a higher perception of their analysis ability than low experience subjects.	Opposite Supported
H 4c	The high experience subjects utilizing Relationship Analysis will have the highest perception of their analysis ability.	Not Supported
H 5a	Subjects performing Relationship Analysis will be more satisfied with the analysis process than those using use-case analysis alone.	Not Supported
H 5b	Subjects with high experience will be more satisfied with the analysis process than low experience subjects.	Supported
H 5c	The high experience subjects utilizing Relationship Analysis will be the most satisfied with the analysis process.	Not Supported

CHAPTER 7

DISCUSSION, CONCLUSIONS, AND FUTURE RESEARCH

This chapter provides a summary of evaluation results and their implication on the hypotheses. This is then followed by a summary of subject comments during the debriefing sessions. Next a discussion on an experimentation enhancement is provided. This is then followed by a discussion on the contributions provided by this dissertation. Lastly, future research and next steps are discussed.

7.1 Summary of Hypotheses Evaluation Results

One goal of Relationship Analysis (RA) is to provide the software community a usable technique that improves an analyst's effectiveness in relationship discovery and documentation. The first step in this direction is to show that RA improves analysis quality. In addition, if a technique is to be accepted, users must show satisfaction with its process. To this end, this dissertation provides experimental results on RA in terms of analysis quality, process satisfaction, perceived analysis ability and implementation time.

The most significant finding of the experiment involves analysis quality. The results of the experiment do significantly show that RA improves analysis quality. Subjects using RA achieved a higher level of analysis quality than those not using RA. This finding further suggests that low experience analysts utilizing RA could be more effective than high experience analysts without RA.

Another important factor to any analysis process is time to perform the analysis. One of my original arguments was that RA fills the void in the analysis process with respect to explicitly identifying relationships. Consequently, this step will help generate a class diagram without adding to the overall analysis time needed to construct a task specific class diagram. My hypotheses concerning the time construct was based on the fact that although the process takes time to perform, it should not effect the overall total process time. The results indicate that RA does add to the overall analysis time. However, this could be a task dependent variable. It may be that a systematic process does help to improve quality without adding to the overall analysis time for more complex tasks. More research is needed in this area.

Also, it may be possible to increase analysts' perceived analysis ability by providing them a well-defined systematic process. There is a growing body of literature that suggests that the benefits of expertise are less pronounced when solving a problem in a well-structured manner. The results of the experiment were inconclusive as to whether RA increases confidence in analysts' ability due to the RA process.

Lastly, the final supported hypothesis was that high experience groups were satisfied with the RA process. The results were inconclusive as to whether low experience groups using RA were as satisfied as high experience groups. It can therefore be stated that RA did not make either low or high experience groups less satisfied.

7.2 Debrief Session Subject Comments

During the debriefing sessions performed at the conclusion of the experiment, the following comments (paraphrased) deserve mention.

- I am not able to draw class diagrams by looking at use-case diagrams. I would have preferred to be in a group that did the RA.
- It is difficult to create class diagrams and I agree that something is missing to help do this. The technique was helpful, but very long.
- After our group did one, the process was pretty straightforward.
- I liked the way the final class diagram was developed by the previous pieces.
- The technique was helpful in creating the class diagram, but I saw the solution about half way through.
- It would have been nice to have a computer tool to help generate the templates and the diagrams instead of doing it on paper.

From the discussion of the debriefing sessions a conclusion that can be drawn is that the RA process does aid in class diagram generation. The subjects also felt that the templates were easy to use to elicit and document the relationships. However, most subjects expressed that the time needed to perform the analysis was too long and required too much paper. Therefore, a way must be found to reduce the time needed to perform the analysis without reducing the effectiveness of the technique. These comments helped to formulate future research discussed in Section 7.5.

7.3 Experimentation Enhancements

The time results of the experiment could be reduced if the subjects performed the process themselves after the training session but before the start of the experiment. This has to do with the learning curve associated with learning a new technique. In addition, it ensures that all subjects fully understand the process before they start the experiment. This could be realized utilizing an interim homework assignment. This would ensure that all subjects fully understand the process before the start of the experiment, thereby reducing the learning curve effect, which may affect the time results.

7.4 Research Contributions

The accomplishments provided by this dissertation offer both theoretical and practical contributions. The contributions encompass a theory-based systematic discovery process to classify, identify, and document the complete relationship structure of an application domain.

The Relationship Analysis Model (RAM) presented is the first theory-based taxonomy to classify relationships. The model, based on Guilford's Structure of Intellect (SI) Theory provides a strong foundation to categorically classify the complete range of relationships of a problem domain.

The model was applied and a technique developed to explicitly identify and document the relationship structure of an application domain, thereby filling a void in the systems analysis process. The Relationship Analysis Process (RAP) is a systematic analysis technique to explicitly identify and document relationships. Supporting the process are the Relationship Analysis Template (RAT) and Relationship Analysis Diagram (RAD) to facilitate the relationship discovery and documentation process.

A rigorous evaluation was conducted, including a formal experiment comparing novice and experienced analysts with and without Relationship Analysis. It was shown that the RAP based on the model does provide a fuller and richer systems analysis, resulting in improved quality of and reduced time in generating class diagrams. It also was shown that Relationship Analysis enables analysts of varying experience levels to achieve a similar level of quality of class diagrams. Relationship Analysis significantly enhances the systems analyst's effectiveness, especially in the area of relationship discovery and documentation resulting in improved analysis and design artifacts.

7.5 Future Research

A significant finding of this research showed empirically that RA improves analysis quality. However, if any analysis technique is to be accepted, users must show satisfaction with its process. The empirical results of the experiment were inconclusive as to measured satisfaction. Ease of use impacts user satisfaction. Therefore, one way to make the process easier is to provide computer support tools to facilitate the process. To this end, a logical next step is to provide a computer based application program to facilitate the process and that computerizes the RA templates and provides a mechanism to draw the RA diagram.

In addition, the templates are based on the RA model that explicitly classifies all the relationships of an application domain based upon elicitation questions. However, are all the relationships necessary to understand the application domain and feed into the design phase? Future research should determine if a limited subset is useful and if so, which subsets would best serve which domains? The effects of a limited process may decrease the time needed to perform the analysis, which may also impact satisfaction. The templates provide generic questions that should be catered to the application domain. Another research area is how to best cater the questions to the particular domain?

The empirical results of this experiment were based on teams comprised of individuals of either low or high experience level. What about mixed experience level teams? Would mixed experience level teams cause analysis quality and process satisfaction to be improved? This is also an area of future research.

The experiment was performed on NJIT students. Since the subjects were chosen from evening courses, many had practical industry based experience developing software

applications using structured analysis and object-oriented analysis techniques. The experiment should also be performed as a field study in a corporation.

Although RA is a methodology independent technique, it seamlessly fits within the object-oriented development phases. RA can be positioned between the use-case analysis and class diagram generation steps. It should be possible to fold RA within the Rational Unified Process and the UML toolkit. This research should be presented to Rational Rose Corporation to acquire support and possible funding. The best way to accomplish this is to provide positive empirical results of the RA technique from field studies performed in software corporations. Therefore, a series of RA experiments are needed at various types of corporations that implement software solutions using the RUP and the UML to develop software systems, especially in the area of analysis and design. If it can be shown that the RA technique improves the process and that practitioners are satisfied and accept the technique, then an argument can be made to its inclusion into the RUP and the UML toolkit.

The RA technique can also support structured analysis (SA) techniques. The background literature indicates that SA techniques explicitly identify entities but leave relationship identification as an implicit process. In particular, RA could enhance the way entity relationship (E/R) diagrams are generated. An experiment can be designed to test the effectiveness of RA with respect to the generation of E/R diagrams.

Many websites implement hypertext functionality. As such, hypertext endpoints form relationships among the corresponding pieces of information. Can these relationships provide clues to improve website design? Therefore, another area of study is to determine if RA is useful in website design.

In addition, this dissertation inserts RA at the analysis phase of the software development life-cycle model. Is it possible to use RA at the requirements definition phase? Feedback from other analysis researchers acquired from conferences and submitted publications are that some prefer to identify all aspects of the problem domain before the analysis phase, namely in the requirements definition phase. Future research can empirically establish the best phase in which to perform RA or show that it is equally effective at either phase.

The previous areas of future research all concerned the computer application domain. Can the RA taxonomy be applied to other domains such as medical systems or transportation systems? Future research is needed to determine RA applicability outside information systems development.

APPENDIX A CONSENT FORM

Appendix A contains a consent and release form.

**NEW JERSEY INSTITUTE OF TECHNOLOGY
323 MARTIN LUTHER KING BLVD.
NEWARK, NJ 07102**

CONSENT TO PARTICIPATE IN A RESEARCH STUDY FORM

TITLE OF STUDY:

RESEARCH STUDY:

I, _____, have been asked to participate in a research study under the direction of Joseph Thomas Catanio. Other professional persons who work with him as study staff may assist to act for him.

PURPOSE:

To test the effectiveness of a systems analysis tool.

DURATION:

My participation in this study will last for 1 week.

PROCEDURES:

I have been told that, during the course of this study, the following will occur:

I will be trained in analysis techniques.

I will be placed in a team.

The team will be given a problem statement to analyze using analysis techniques. I will be able to communicate with my team members to complete the assignment.

My team will be required to submit all generated analysis documents and record the time spent completing the assignment.

I am required to fill out a questionnaire and participate in a debriefing session at the conclusion of the experiment.

I was given a choice of either participating in this experiment or working on a similar project.

PARTICIPANTS:

I will be one of about 180 participants to participate in this trial.

EXCLUSIONS:

I will inform the researcher if any of the following apply to me: N/A

RISK/DISCOMFORTS:

I have been told that the study described above may involve the following risks and/or discomforts:

There are no known risks or discomforts.

There also may be risks and discomforts that are not yet known.

CONFIDENTIALITY:

Every effort will be made to maintain the confidentiality of my study records. Officials of NJIT will be allowed to inspect sections of my research records related to this study. If the findings from the study are published, I will not be identified by name. My identity will remain confidential unless disclosure is required by law.

PAYMENT FOR PARTICIPATION:

I have been told that I will receive \$0 compensation for my participation in this study.

CONSENT AND RELEASE:

I fully recognize that there are risks that I might be exposed to by volunteering in this study which are inherent in participating in any study; I understand that I am not covered by NJIT's insurance policy for any injury or loss I might sustain in the course of participating in the study.

RIGHT TO REFUSE OR WITHDRAW:

I understand that my participation is voluntary and I may refuse to participate, or may discontinue my participation at any time with no adverse consequence. I also understand that the investigator has the right to withdraw me from the study at any time.

INDIVIDUAL TO CONTACT:

If I have any questions about my treatment or research procedures that I discuss them with the principle investigator. If I have any addition questions about my rights as a research subject, I may contact:

Richard Greene, M.D., Ph.D., Chair, IRB (973) 596-3281.

SIGNATURE OF PARTICIPANT

I have read this entire form, or it has been read to me, and I understand it completely. All of my questions regarding this form or this study have been answered to my complete satisfaction. I agree to participate in this research study.

Subject's Name: _____

Signature: _____

Date: _____

SIGNATURE OF INVESTIGATOR OR RESPONSIBLE INDIVIDUAL

To the best of my knowledge, the participant,

_____,

has understood the entire content of the above consent form, and comprehends the study. The participants and those of his/her parent/legal guardian have been accurately answered to his/her/their complete satisfaction.

Investigator's Name: _____

Signature: _____

Date: _____

APPENDIX B TRAINING MATERIALS

Appendix B contains the training material example on how to perform a Use-case Analysis and a Relationship Analysis to generate class diagrams to the given problem statement.

PROBLEM STATEMENT: Inventory Control System

A store is setup to fill customer and vendor orders. To fill these orders products are maintained in a warehouse. These products are not given directly to the customer instead they are packaged and shipped to the customer. Once an order is placed, the order fulfillment employee fills the order by locating and packaging the products in the warehouse. As the ordered is filled, the fulfillment employee updates the inventory list to reflect the fact that the particular product item was taken from inventory. Once all the items of the order are packaged, the order-processing department is notified that the order has been filled.

The package is then prepared for shipping. The shipping employee updates the inventory list to reflect the date items of an order were shipped and notifies the order-processing department that the order has been shipped.

The stock clerks handle stocking the warehouse. The products may come from cancelled orders, returned orders, or vendor shipments. The products are placed in the warehouse in predefined locations. Once a product is stocked, the stock clerk updates the inventory list to reflect location and quantity.

The receiving clerks receive incoming shipments by matching purchase orders against the shipment stock. The receiving clerk informs the accounts payable department when purchase order items are received. To help offset the stocking department workload, the receiving clerks may directly stock the product in the warehouse and update the inventory list.

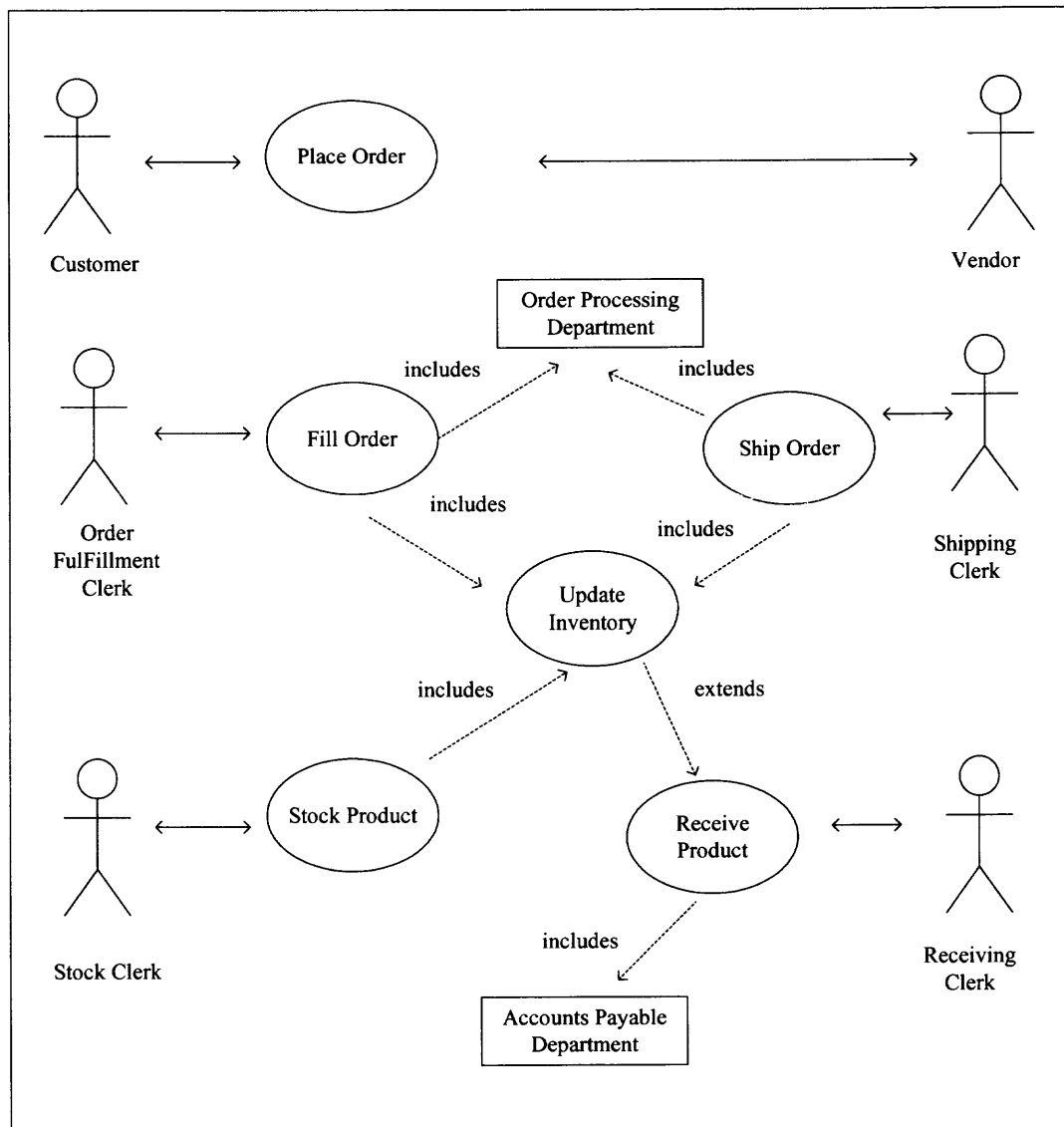
Step 1: Identify the actors.

Actors are people or sub-systems that communicate with the inventory control system.

- Customer
- Vendor
- Order Fulfillment Clerk
- Order Processing Department (system type actor)
- Shipping Clerk
- Stocking Clerk
- Receiving Clerk
- Accounts Payable Department (system type actor)

Step 2: Create a use-case diagram to depict high-level system features.

Figure B1.1 Use-case Analysis Training



Step 3: Build use-case narratives to sublimate the diagram. These narratives describe what the user expects from the use-case. The focus is on what not how.

Place Order

Customers and vendors place orders to a sales representative either in person or over the telephone. Orders consist of 1 or more products that the customers or vendors wish to purchase. To place an order the sales representative creates a unique order number that contains customer or vendor information and ordered products. In addition, since the order is to be packaged and shipped, delivery information is also recorded on the order process form. This is the first step in the process and order forms are used to start the

inventory system. A future system enhancement will be the ability to place orders on-line to feed the inventory system directly.

Fill Order

Fill order clerks are provided order forms. It is assumed that these order forms are completed properly and that the order number is valid. It is also assumed that all products are available in inventory at the warehouse. Fill order clerks collect the products required to fill the order from the warehouse. As products are removed from storage shelves, the clerks update the inventory list. All products are packaged, placed in boxes, and placed in the shipment department with the original order form.

Ship Order

Shipping clerks assume that the package contains all the products listed on the order form. The clerk marks the package with the destination address listed on the order form and updates the inventory list. The packaged is then loaded into a truck for delivery.

Stock Product

The task of stocking the warehouse falls on the stock clerks. Products are placed in predefined shelf locations. As products are stocked, stock clerks update the inventory list to reflect the new quantity.

Receive Product

To replenish the warehouse inventory, products are received from various distributors. It is assumed that products are properly ordered and sent to receiving so that the products contained in the inventory list are always available. As an item is received, the receiving clerk updates the inventory list and placed in the stock department. Receiving clerks may also stock the product directly.

Update Inventory

Update inventory allows clerks to mark the status of inventory products. It is assumed that update inventory users have permission to update the inventory list. Products in the inventory list can be marked as filled, shipped, received or stocked with the date recorded for each action. This permits products to be tracked.

Step 4: Identify items of interest and their relationships.

What is an item of interest? Anything in the system you want to know more about.

Identify and document the relationships using six templates.

Item if interest = Order

Unit Template

The unit template determines specification and elaboration relationships.

Table B1.1 Order Training Unit Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	Unit
Convergent Relationship	Specification
Generic Question(s) (Optional)	Does the item have a description? Does the item have a definition? Does the item have an explanation? Does the item have a set of instructions? Does the item have an illustration?
Specific Question(s)	Does order have an explanation?
Result	Customers place orders. Vendors place orders. Sales Representatives take orders. Orders consist of products. Orders have an order number. Orders are packaged by fulfillment clerks. Orders are shipped by shipping clerks. Orders are filled by fulfillment clerks. Orders are processed. Orders written on an order form.
Divergent Relationship	Elaboration
Generic Question(s) (Optional)	Does the description fully describe the item? Does the definition fully encompass the item? Does the explanation make assumptions? Are the set of instructions complete? How can this item be expanded or broadened?
Specific Question(s)	Does the explanation make assumptions?
Results	An order is assumed to be completed properly. An order is assumed to contain an order number. An order is assumed to comprise all the products (all products in stock). An order is assumed to be sent to the correct address.

Collection Template

The collection template determines membership and aggregation relationships.

Table B1.2 Order Training Collection Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	Collection
Convergent Relationship	Membership
Generic Question(s) (Optional)	Is this item a segment of a whole item? Is this item a member of a collection? What is this item a part-of? What components consist of this item? What phrases are in this whole activity?
Specific Question(s)	Is order a member of a collection?
Results	An order is a collection of 1 or more products.
Divergent Relationship	Aggregation
Generic Question(s) (Optional)	Which components comprise this item? What materials are used to make this item? What is part-of this item?
Specific Question(s)	What materials are used to make an order?
Results	An order requires an order number, order date, product numbers, product quantity, customer information, payment information, and delivery information.

Comparison Template

The comparison template determines generalization/specialization and similar/dissimilar relationships.

Table B1.3 Order Training Comparison Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	Comparison
Convergent Relationship	Generalization/Specialization
Generic Question(s) (Optional)	Is the item a kind of parent item? Does the item completely include or encompass other items? Is there a broader term for this item? Is there a narrower term for this item?
Specific Question(s)	Is there a broader term for order?
Results	No
Divergent Relationship	Similar/Dissimilar
Generic Question(s) (Optional)	Which other items are similar to this item? What serves the same purposes as this item? Which others items are opposite to this item?
Specific Question(s)	Which other items are similar to order?
Results	An order is a unique system feature. However, an order is opposite to the delivered product.

System Template

The system template determines structure and occurrence relationships.

Table B1.4 Order Training System Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	System
Convergent Relationship	Structure
Generic Question(s) (Optional)	What prerequisites or preconditions exist for this item? What follows this item for a given purpose? What precedes this item for a given purpose? Which items are close to this item?
Specific Question(s)	What prerequisites or preconditions exist for an order?
Results	The customer must be part of the client database before the order can be placed.
Divergent Relationship	Occurrence
Generic Question(s) (Optional)	Where else does this item appear in the domain? Where else is this item used in this system and in other systems? What are all uses of this item? Where was this item used before? Where else is the item used now? Where will this item be used later?
Specific Question(s)	What are all uses of the order?
Results	An order is placed before shipping begins. An order is placed before inventory is updated. An order is placed before it is filled. An order is placed before it is processed.

Transformation Template

The transformation template determines modify and transpose relationships.

Table B1.5 Order Training Transformation Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	Transformation
Convergent Relationship	Modify
Generic Question(s) (Optional)	What can this item change into? What output results from the item's inputs? What resources and mechanisms are required to modify this item? Who can modify this item?
Specific Question(s)	What can the order change into?
Results	The order can change into a backorder if all inventory items are not in stock.
Divergent Relationship	Transpose
Generic Question(s) (Optional)	How can this item be reused? How can this item be viewed differently? Can this item be used in a different context?
Specific Question(s)	Can the order be used in a different context?
Results	Orders can identify what inventory items are most desired by the customers, allowing the business to stock more of a certain more desirable item.

Implication Template

The implication template determines influence and extrapolate relationships.

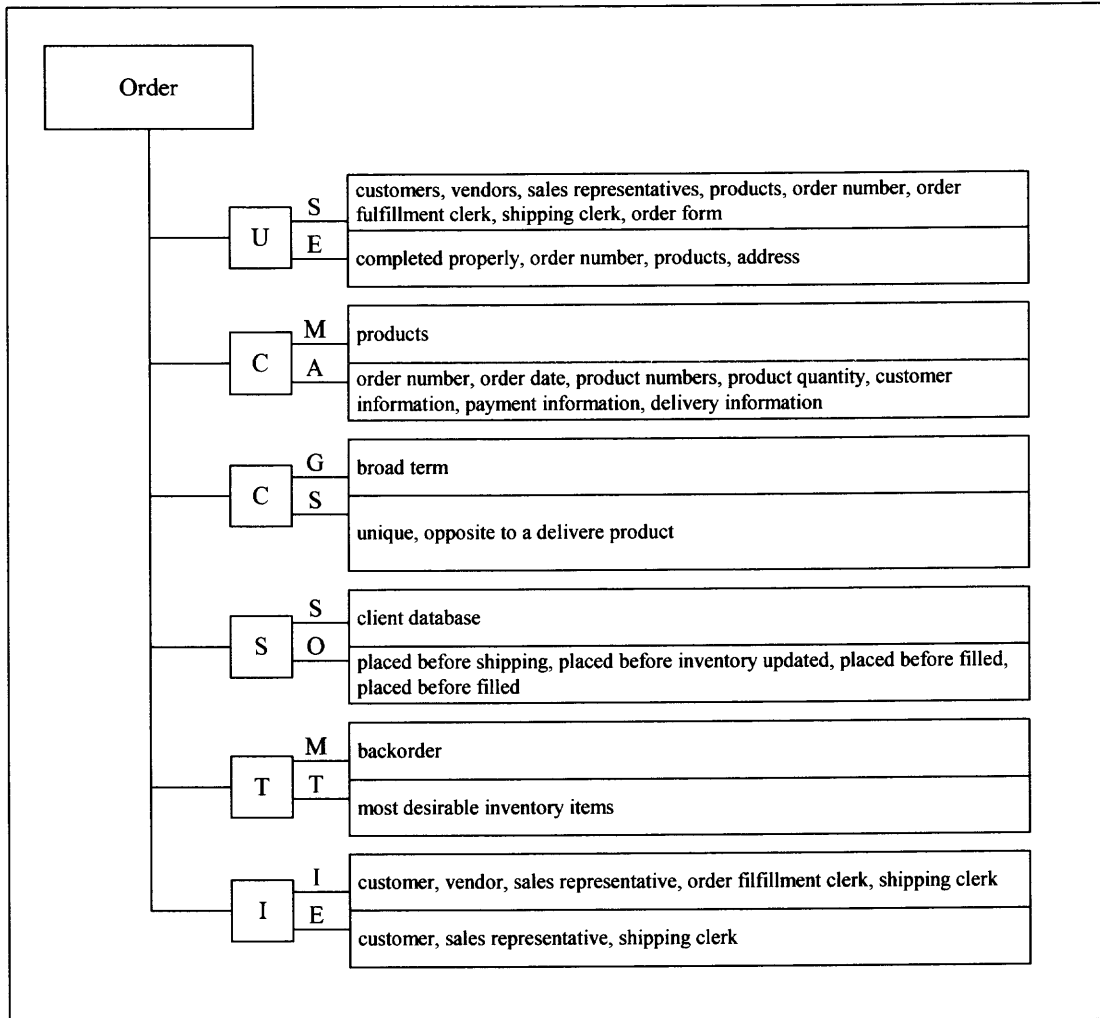
Table B1.6 Order Training Implication Template

Item of Interest	Order
Description	An order is a list of 1 or more products that a customer or vendor desires to purchase.
Focus	Implication
Convergent Relationship	Influence
Generic Question(s) (Optional)	What items or people cause this item to be created, changed, or deleted? What items or people have control over this item?
Specific Question(s)	What items or people have control over an order?
Results	A customer can cancel or change an order. A sales representative creates the order form. A shipping clerk can change the way the package is shipped, for example regular or express.
Divergent Relationship	Extrapolate
Generic Question(s) (Optional)	Which goals, issues, and arguments involve this item? What are the positions and statements on the item? What are the comments on this item? What are the opinions on this item? What is the rationale for this decision?
Specific Question(s)	What is the rationale for this decision?
Results	A customer can cancel or change an order due to a change of mind. A sales representative creates the order form to earn money. A shipping clerk can change the way the package is shipped to achieve greater customer satisfaction.

Step 5: Depict the Relationships Utilizing the Relationship Analysis Diagram

Extracting the resulting template information into a graphical representation yields the following RAD for the item of interest, namely “order”.

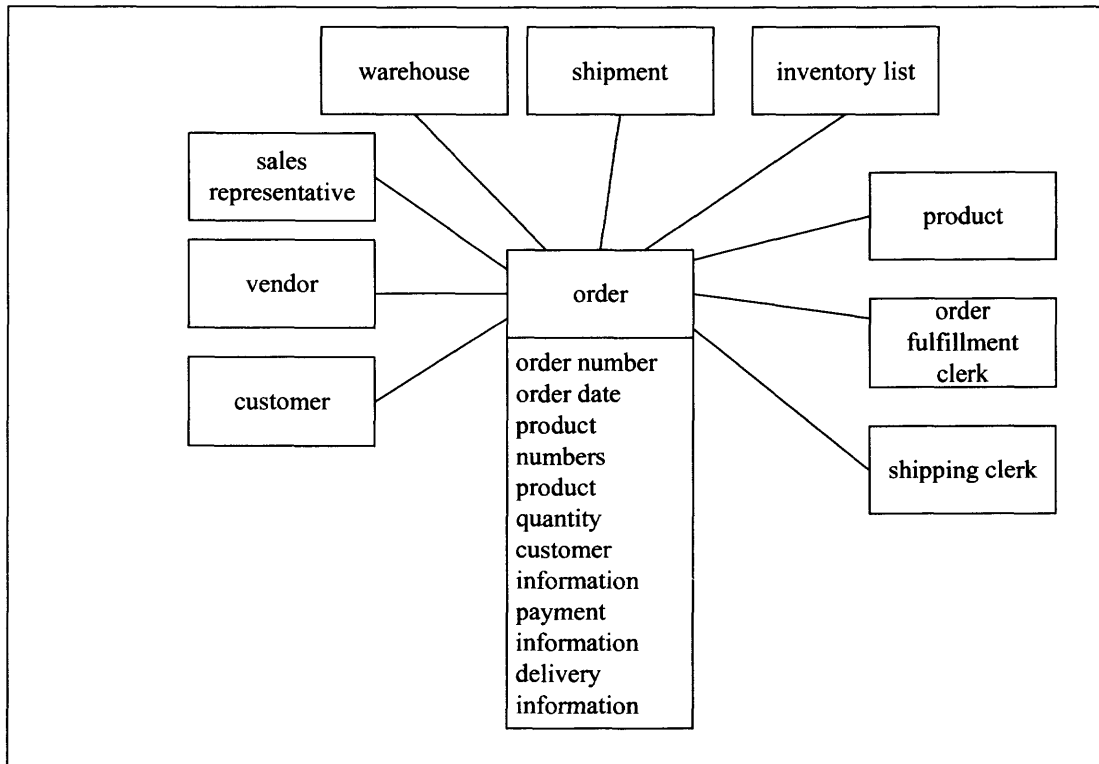
Figure B1.2 Order Training RAD



Step 6: Create a Class Diagram

A class diagram represents main system objects and their relationships.

Figure B1.3 Training Class Diagram 1



Repeat the process for other items of interest and refine the class diagram.

Item of interest = customer

Unit Template

The unit template determines specification and elaboration relationships.

Table B1.7 Customer Training Unit Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus	Unit
Convergent Relationship	Specification
Generic Question(s) (Optional)	Does the item have a description? Does the item have a definition? Does the item have an explanation? Does the item have a set of instructions? Does the item have an illustration?
Specific Question(s)	Does a customer have a description?
Result	A customer places an order with a sales representative.
Divergent Relationship	Elaboration
Generic Question(s) (Optional)	Does the description fully describe the item? Does the definition fully encompass the item? Does the explanation make assumptions? Are the set of instructions complete? How can this item be expanded or broadened?
Specific Question(s)	Does the explanation make assumptions?
Results	It is assumed that the customer places an order and the request goes to the warehouse to be filled.

Collection Template

The collection template determines membership and aggregation relationships.

Table B1.8 Customer Training Collection Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus Type	Collection
Convergent Relationship	Membership
Generic Question(s) (Optional)	Is this item a segment of a whole item? Is this item a member of a collection? What is this item a part-of? What components consist of this item? What phrases are in this whole activity?
Specific Question(s)	Is customer a member of a collection?
Results	No
Divergent Relationship	Aggregation
Generic Question(s) (Optional)	Which components comprise this item? What materials are used to make this item? What is part-of this item?
Specific Question(s)	What is part of customer?
Results	Customer information consists of name, address, telephone number, and payment information.

Comparison Template

The comparison template determines the generalization/specialization and similar/dissimilar relationships.

Table B1.9 Customer Training Comparison Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus	Comparison
Convergent Relationship	Generalization/Specialization
Generic Question(s) (Optional)	Is the item a kind of parent item? Does the item completely include or encompass other items? Is there a broader term for this item? Is there a narrower term for this item?
Specific Question(s)	Is there a broader term for customer?
Results	Yes, a customer is a kind of purchaser.
Divergent Relationship	Similar/Dissimilar
Generic Question(s) (Optional)	Which other items are similar to this item? What serves the same purposes as this item? Which others items are opposite to this item?
Specific Question(s)	What is similar to a customer?
Results	A vendor is similar to a customer. However, a sales representative is opposite to a customer.

System Template

The system template determines structure and occurrence relationships.

Table B1.10 Customer Training System Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus	System
Convergent Relationship	Structure
Generic Question(s) (Optional)	What prerequisites or preconditions exist for this item? What follows this item for a given purpose? What precedes this item for a given purpose? Which items are close to this item?
Specific Question(s)	Which items are close to a customer?
Results	Vender
Divergent Relationship	Occurrence
Generic Question(s) (Optional)	Where else does this item appear in the domain? Where else is this item used in this system and in other systems? What are all uses of this item? Where was this item used before? Where else is the item used now? Where will this item be used later?
Specific Question(s)	Where else does customer appear in the domain?
Results	In the shipping process.

Transformation Template

The transformation template determines modify and transpose relationships.

Table B1.11 Customer Training Transformation Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus	Transformation
Convergent Relationship	Modify
Generic Question(s) (Optional)	What can this item change into? What output results from the item's inputs? What resources and mechanisms are required to modify this item? Who can modify this item?
Specific Question(s)	What can a customer change into?
Results	The customer can turn into a supplier.
Divergent Relationship	Transpose
Generic Question(s) (Optional)	How can this item be reused? How can this item be viewed differently? Can this item be used in a different context?
Specific Question(s)	Can the customer be used in a different context?
Results	Customers can be used to determine what items should be pre-ordered to prevent a backorder.

Implication Template

The implication template determines influence and extrapolate relationships.

Table B1.12 Customer Training Implication Template

Item of Interest	Customer
Description	A customer is someone that places an order to purchase products contained in the warehouse.
Focus	Implication
Convergent Relationship	Influence
Generic Question(s) (Optional)	What items or people cause this item to be created, changed, or deleted? What items or people have control over this item?
Specific Question(s)	What items of interest or people have control over a customer?
Results	A sales associate can influence a customer's order by discounting desired products.
Divergent Relationship	Extrapolate
Generic Question(s) (Optional)	Which goals, issues, and arguments involve this item? What are the positions and statements on the item? What are the comments on this item? What are the opinions on this item? What is the rationale for this decision?
Specific Question(s)	What is the rationale for this decision?
Results	Certain products may be overstocked in the inventory and the goal is to reduce inventory to make room for other products.

Figure B1.4 Customer Training RAD

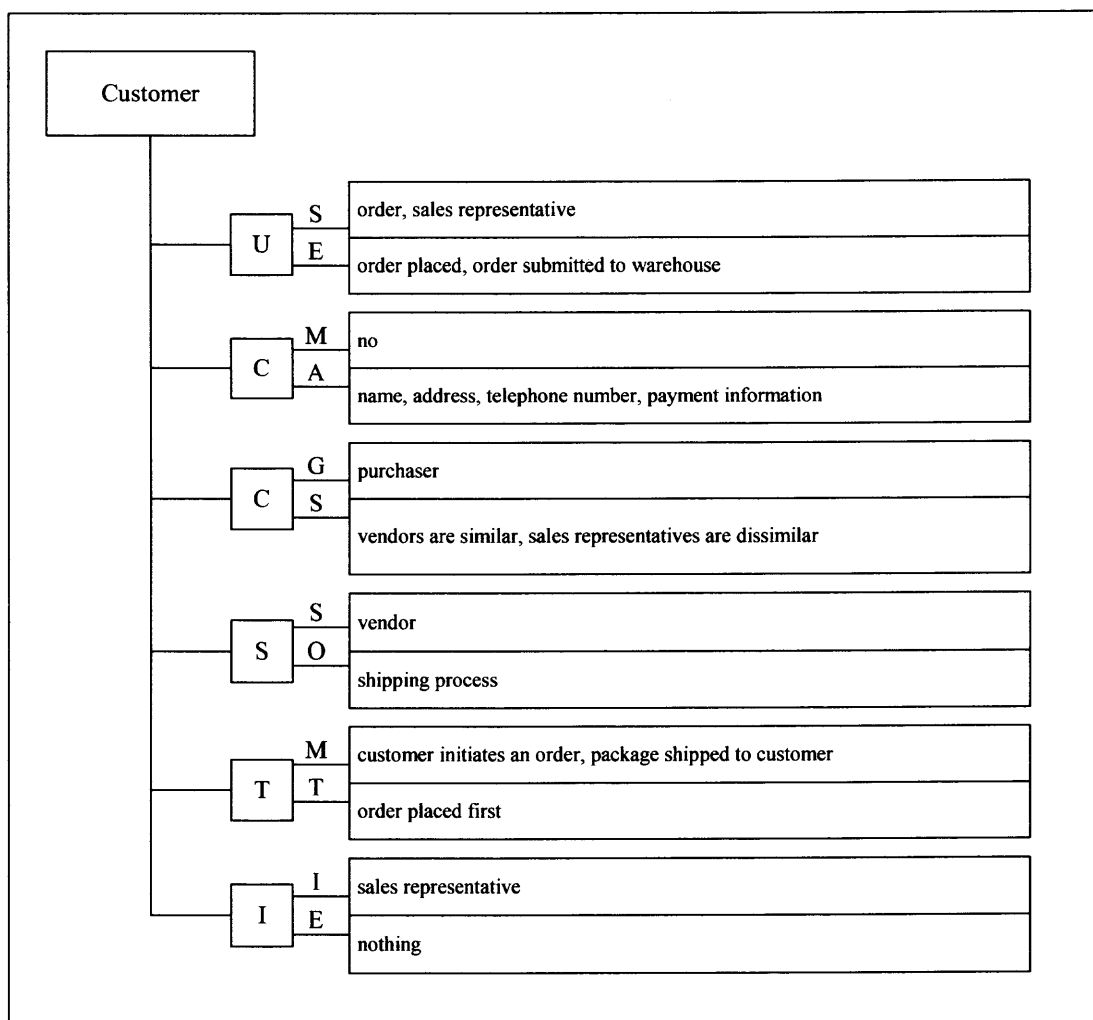
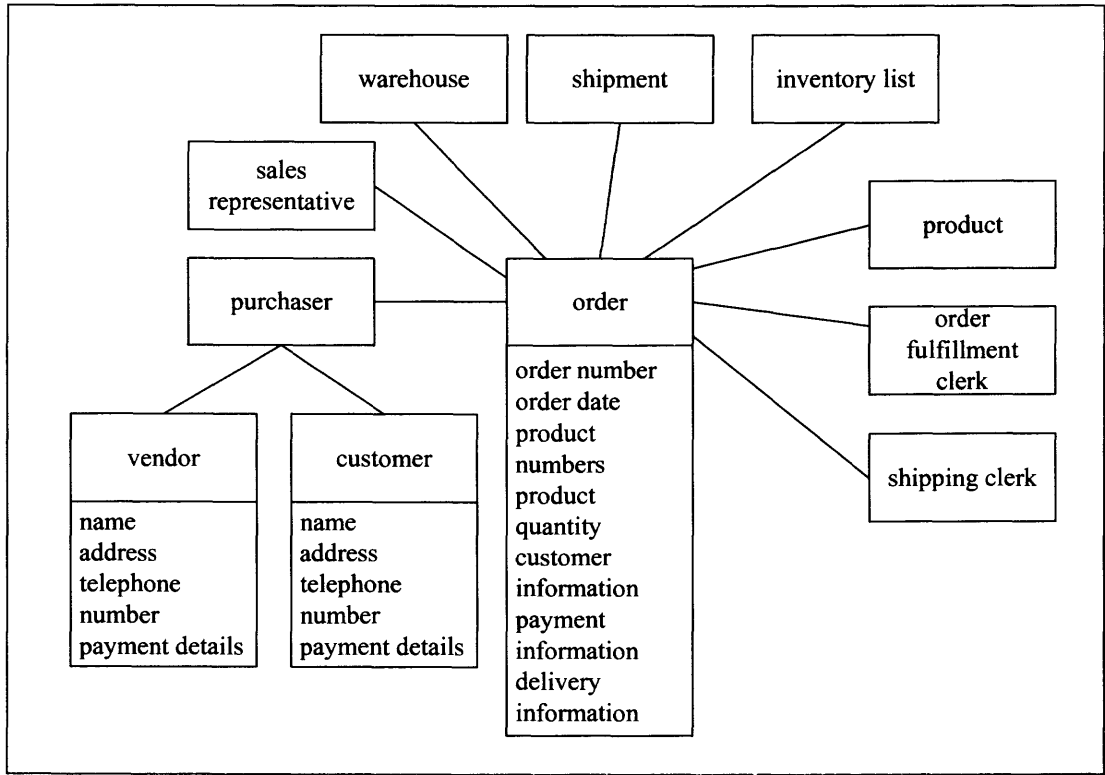
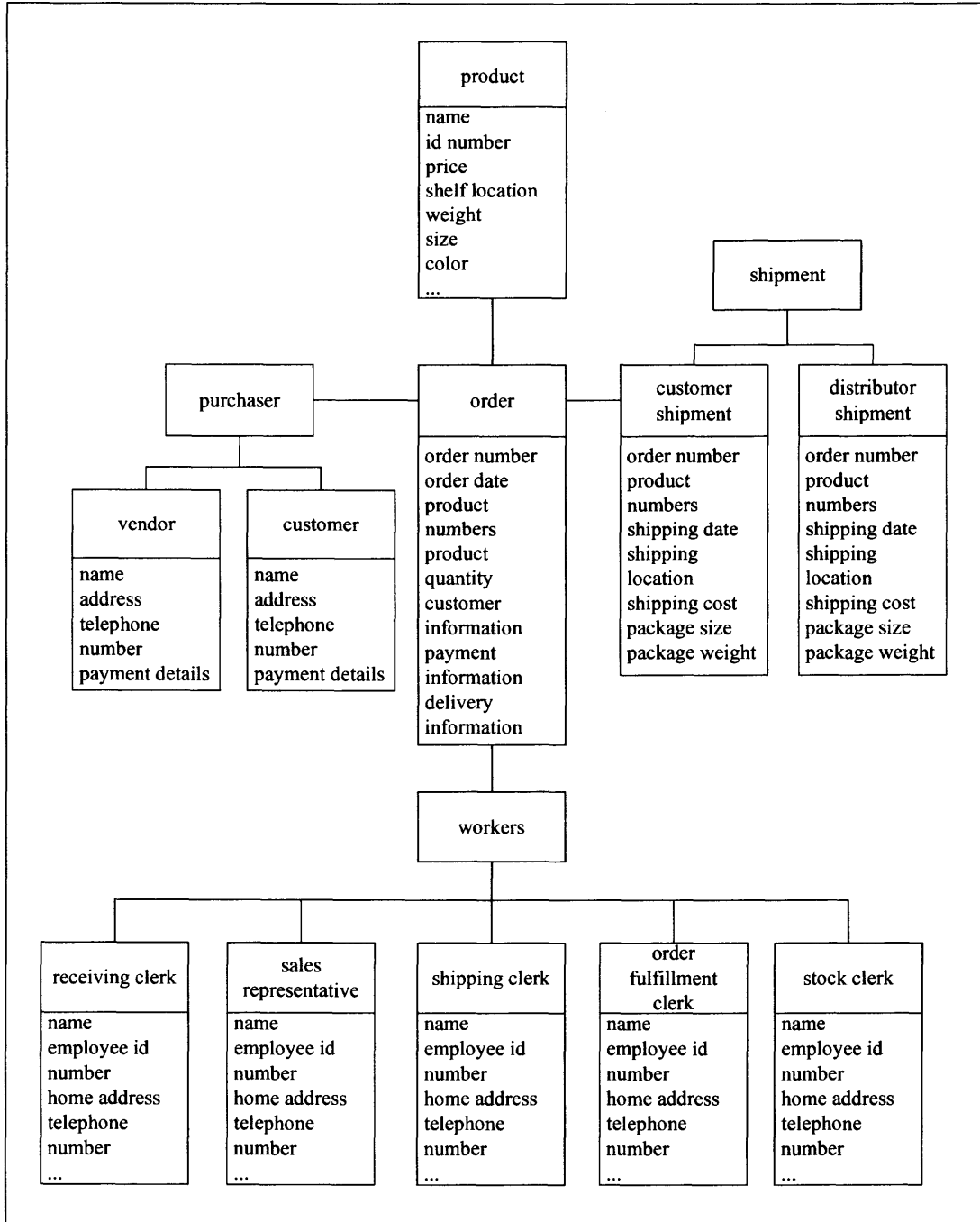


Figure B1.5 Training Class Diagram 2



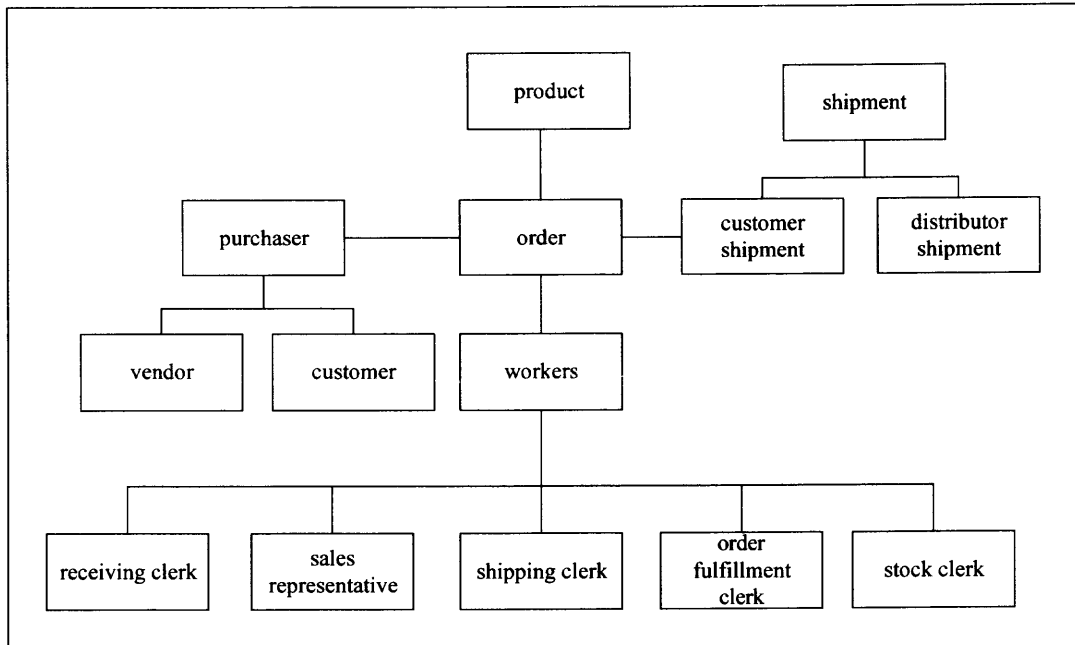
Using the other items of interest the same way should result in a final class diagram.

Figure B1.6 Training Class Diagram Final



Final class diagram without the attributes.

Figure B1.7 Training Class Diagram Final Without Attributes



APPENDIX C SURVEY INSTRUMENTS

Appendix C contains four questionnaires. The use-case and Relationship Analysis training questionnaires were used to validate the training material. The experience questionnaire will be used to determine experience level prior to team creation. The post experiment questionnaire will be used for perception measures.

POST USE-CASE ANALYSIS TRAINING QUESTIONNAIRE

Last 4 digits of your SS#: _____

1. I am satisfied with the use-case analysis training.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

2. The use-case analysis teaching material was clear.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

3. I am not confident in my ability to perform a use-case analysis.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

4. I feel that use-case analysis is difficult.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

5. I understood the use-case analysis example presented.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

6. I did not feel that the use-case analysis example was clear.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

7. I feel that use-case analysis will provide needed information to designers.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

8. I do not feel use-case analysis needs to be used.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

End of Use-case Analysis Training Questionnaire

POST RELATIONSHIP ANALYSIS TRAINING QUESTIONNAIRE

Last 4 digits of your SS#: _____

1. I am satisfied with the relationship analysis training.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

2. The relationship analysis teaching material was clear.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

3. The relationship analysis templates are easy to understand.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

4. I do not feel that the relationship analysis templates properly document the relationships.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

5. I do not feel that relationship analysis diagrams provide a good graphical depiction of the discovered relationships.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

6. The relationship analysis diagrams are easy to understand.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

7. The relationship analysis process is easy to follow.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

8. The relationship analysis process is not systematic.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

9. I am not confident in my ability to perform a relationship analysis.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

10. I feel that relationship analysis is difficult.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

11. I understood the relationship analysis example presented.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

12. I did not feel that the relationship analysis example was clear.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

13. I feel that relationship analysis will provide needed information to designers.

Strongly Agree ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Strongly Disagree

14. I do not feel relationship analysis should be used.

Strongly Agree ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Strongly Disagree

End of Relationship Analysis Training Questionnaire

EXPERIENCE QUESTIONNAIRE

Name: _____

Last 4 digits of your SS#: _____

Course #: _____

Section #: _____

Academic Coursework:

Please place a check mark (✓) in the box next to any of the following courses that you have already completed or are currently enrolled. Also, indicate the grade you received in the already completed courses.

Course Number & Name	✓	Grade
CIS 101 Computer Programming and Problem Solving		
CIS 103 Computer Science Business Problem Solving		
CIS 113 Introduction to Computer Science I		
CIS 114 Introduction to Computer Science II		
CIS 280 Programming Language Concepts		
CIS 375 Application Development on the WWW		
CIS 381 Object-oriented Software Systems		
CIS 390 Analysis & System Design		
CIS 431 Introduction Database Systems		
CIS 434 Advanced Database Systems		
CIS 464 Advanced Information Systems		
CIS 490 Design in Software Engineering		
CIS 491 Computer Science Project		
CIS 601 Object-oriented Programming in C++		
CIS 602 Java Programming		
CIS 631 Data Management Systems Design		
CIS 632 Advanced Data Management Systems Design		
CIS 663 Advanced Systems Analysis & Design		
CIS 673 Software Design & Production Methodology		
CIS 676 Requirements Engineering		
CIS 683 Object-oriented Software Development		

General Software Background:

1. Circle the number that indicates your familiarity with the following programming languages.

- a. C# Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- b. C++ Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- c. C Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- d. Java Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- e. Basic Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- f. Pascal Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- g. Cobol Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- h. Assembly Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- i. Fortran Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used

2. Circle the number that indicates your familiarity with the following scripting languages.

- a. XML Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- b. HTML Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- c. VBScript Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- d. JavaScript Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used

3. Circle the number that indicates your familiarity with the following Database Management System (DBMS) packages.

- a. Oracle Very Familiar With $\leftarrow 7 - 6 - 5 - 4 - 3 - 2 - 1 \rightarrow$ Have Not Used
- b. SQL Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used
- c. Access Very Familiar With $\leftarrow 1 - 2 - 3 - 4 - 5 - 6 - 7 \rightarrow$ Have Not Used

Software Development Background: (Please indicate your skill level in the following areas)

1. Software development life-cycle models (waterfall, spiral, iterative, RUP, etc.)

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

2. Software Economics (cost-benefit analysis, software cost estimation, feasibility studies, etc.)

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

3. Generating software system analysis documents.

High Skill ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Low Skill

4. Generating software system design documents.

High Skill ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Low Skill

5. Generating software system class diagrams.

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

6. Developing software code.

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

7. Using Modeling Languages and Techniques (UML, etc.)

High Skill ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Low Skill

Work Related Experience: (Please indicate your skill level in the following areas)

1. Working as a software engineer.

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

2. Working as a software developer.

High Skill ← 1 – 2 – 3 – 4 – 5 – 6 – 7 → Low Skill

3. Working as a system analyst.

High Skill ← 7 – 6 – 5 – 4 – 3 – 2 – 1 → Low Skill

End of Experience Questionnaire

DECISION CRITERION

The score possible ranges from 26 to 203 points, lowest to highest respectfully.

The cutoff point was 100, those below were classified as low experience and those above were classified as high experience.

Criterion	Possible Points	Earned Points
Academic Performance	0 to 21	
General Software Background	16 to 112	
Software Development Background	7 to 49	
Software Engineering Professional Background	3 to 21	
Total		

POST-EXPERIMENT QUESTIONNAIRE

Group ID: _____
Last 4 digits of your SS#: _____
Course #: _____
Section #: _____

For each of the following statements, please circle the rating of your choice.

1. I am confident in my software system analysis abilities.

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

2. Communication with my group members helped me to solve the problem.

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

3. I am satisfied with the quality of my group's class diagram document.

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

4. My group's problem solving process was efficient.

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

5. I have already analyzed an on-line registration system for my job.

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

6. I feel the task was too difficult.

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

7. I do not use software system analysis techniques to develop software.

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

8. I did not need to communicate with my group to solve the problem.

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

9. I am not confident in the group's final class diagram document.

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

10. My group's problem solving process was coordinated.

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

11. I have not analyzed an on-line registration system for my college studies.

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

12. I understood the task.

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

13. I understand the software system analysis process.

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

14. My group members communicated clearly.

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

15. I am committed to my group's final class diagram document.

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

16. My group's problem solving process was unfair.

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

17. I feel there wasn't enough time to complete the task.

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

18. I do not feel that software systems analysis is needed to develop software.

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

19. I feel that communicating with my group did not help me to better solve the problem.

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

20. The final class diagram document does not reflect my inputs.

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

21. My group's problem solving process was confusing.

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

22. I feel that everyone on my team understood the task.

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

23. Performing the analysis did help me to communicate more effectively.

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

24. I feel I had an equal part in my group's final class diagram document.

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

25. My group's problem solving process was satisfying.

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

End of Post-experiment Questionnaire

APPENDIX D TASK LISTS

Appendix D contains three different task lists and assignment instructions. Task 1 is for the control category. Task 2 is for the treatment category. Task 3 is the alternate task for those whom choose not to participate in the experiment. Assignment instructions are to be given to the subjects.

Prior to the day of the experiment, all subjects will have been taught use-case analysis and how to generate class diagrams as part of the requirements to the course. In addition, all subjects will have completed an experience questionnaire that will be used by an expert to determine low and high experience individuals. The low experience subjects will be randomly selected and placed in a team consisting of a total of three low experience individuals. Similarly, high experience subjects will be randomly selected and placed in a team consisting of a total of three high experience individuals.

Task 1

Day 1: 11/xx/2003

1. Complete and submit the consent form.
2. Subjects divided into teams based on pre-experiment questionnaire.
3. Team selects a group leader.
(Leader will record time spent on project.)
(Leader will submit all project documents.)
4. Enrichment topic provided to prevent confounding of variables.
5. Experimenter hands out problem assignment to each team.
6. Teams perform a use-case analysis in class for 1 hour.
7. Teams provided with an expert generated use-case analysis to the problem statement.
8. Teams have 1 week to complete assignment.

Week 1: 11/xx/2003 to 11/xx/2003

1. Team generates class diagrams.
2. Leader records time to complete class diagram generation.
3. Leader assembles documents and completes cover sheet to project submission.

Day 7: 11/xx/2003

1. Team leaders will submit final class diagrams document.
2. All subjects complete a post-task questionnaire.
3. Experimenter debriefs all subjects.

Task 2**Day 1: 11/xx/2003**

1. Complete and submit the consent form.
2. Subjects divided into teams based on pre-experiment questionnaire.
3. Team selects a team leader.
(Leader will record time spent on project.)
(Leader will submit all project documents.)
4. Subjects trained in Relationship Analysis.
5. Experimenter hands out problem assignment to each team.
6. Teams perform a use-case analysis in class for 1 hour.
7. Teams provided with an expert generated use-case analysis to the problem statement.
8. Teams have 1 week to complete assignment.

Week 1: 11/xx/2003 to 11/xx/2003

1. Group performs Relationship Analysis.
2. Leader records time to complete Relationship Analysis.
3. Group generates class diagrams.
4. Leader records time to complete class diagram generation.
5. Leader assembles documents and completes cover sheet to project submission.

Day 7: 11/xx/2003

1. Team leaders will submit final Relationship Analysis document, and class diagrams document.
2. All subjects complete a post-task questionnaire.
3. Experimenter debriefs all subjects.

Task 3

The alternate task is the same problem statement that the experimental groups are to solve.

Rules:

1. You will have 1 week to complete this project.
2. The project consists of performing a series of techniques to analyze the given problem statement.
3. Your grade will be based on the quality of the analysis documents you generate.
4. Documents to be submitted:
 - a. Use-case Analysis Document
 - b. Relationship Analysis Document
 - c. Class Diagrams Document
 - d. A list of problems encountered during the analysis process
 - e. A description of how you solved the problems encountered.
 - f. Cover Sheet Outlining the time spent on each part.
5. To create the Use-case Analysis Document, determine all the stakeholders of the web-based document archival software system. For each of these stakeholders, determine how they intend and are required to use the on-line registration system. Describe any problems encountered you had performing the use-case analysis. Describe how you solved each problem. Discuss the limitations of each stakeholder. Discuss the limitations of each intended use. Record the time spent on determining the use-cases.
6. To create the Relationship-Analysis Document, determine all the relationships of the systems. These relationships must include on an individual basis; stakeholders, desired system use, desired system functionality, system entities. In addition, you will need to describe how all of the individual entities are interrelated. Describe any problems encountered you had developing the relationship model. Describe how you solved each problem. Discuss the limitations of the relationship model. Record the time spent on performing the Relationship Analysis.
7. To create the Class Diagrams Document, use the information provided by the use-case analysis and relationship analysis. These class diagrams must include all classes and attributes of the on-line registration system. Describe any problems encountered you had developing the class diagrams. Describe how you solved each problem. Discuss the limitations of the class diagram. Record the time spent on generating the class diagrams.
8. Describe ways to improve each of the analysis processes.
9. Describe the relevance of each of the analysis techniques in describing the system to be solved.

Instructions: Control Group

Please follow the instructions to complete the assignment. The assignment is due 1 week from today and must be handed in on time for credit. The assignment is worth 30 points and grading is based on the quality of your team's submitted work. Please accurately record the amount of time you spend on the various assignment sections. Time will not be used as grading criteria! Please work with your team members only. Do not share information with other teams.

Email Address: JosephCatania@att.net
WebBoard Address: <http://webboard.njit.edu>
Board: RA Team Study

Day 1: 11/xx/2003

1. Complete and submit the consent form so that you may participate in the experiment.
2. You have already been placed in a team with others of similar experience.
3. As a team select a team leader.
(Leader will record time spent on project.)
(Leader will submit all project documents.)
4. Work as a team to solve the problem statement.
5. Use the guidelines below to coordinate your activities.
6. A sample cover sheet is available from the above WebBoard Address.

Week 1: 11/xx/2003 to 11/xx/2003

1. As a team generate class diagrams to the problem statement.
2. Team leader records time to complete class diagram generation.
3. Team leader assembles class diagrams document and completes cover sheet to project submission.

Day 7: 11/xx/2003

1. Team leaders will submit class diagrams document, and cover sheet.
2. All team members individually complete a post-task questionnaire.

Instructions: Treatment Group

Please follow the instructions to complete the assignment. The assignment is due 1 week from today and must be handed in on time for credit. The assignment is worth 30 points and grading is based on the quality of your team's submitted work. Please accurately record the amount of time you spend on the various assignment sections. Time will not be used as grading criteria! Please work with your team members only. Do not share information with other teams.

Email Address: JosephCatania@att.net
 WebBoard Address: <http://webboard.njit.edu>
 Board: RA Team Study

Day 1: 11/xx/2003

1. Complete and submit the consent form so that you may participate in the experiment.
2. You have already been placed in a team with others of similar experience.
3. As a team select a team leader.
 (Leader will record time spent on project.)
 (Leader will submit all project documents.)
4. Work as a team to solve the problem statement.
5. Use the guidelines below to coordinate your activities.
6. A sample cover sheet is available from the above WebBoard Address.
7. Relationship Analysis Templates are available from the above WebBoard Address.

Week 1: 11/xx/2003 to 11/xx/2003

1. As a team perform Relationship Analysis and generate Relationship Analysis Templates to the problem statement.
2. Generate the Relationship Analysis Diagram utilizing the information recorded in the Relationship Analysis Templates.
3. Team leader records time to complete Relationship Analysis.
4. As a team generate class diagrams to the problem statement.
5. Team leader records time to complete class diagram generation.
6. Team leader assembles Relationship Analysis document, class diagrams document and completes cover sheet to project submission.

Day 7: 11/xx/2003

1. Team leaders will submit Relationship Analysis document, class diagrams document, and cover sheet.
2. All team members individually complete a post-task questionnaire.

APPENDIX E COVER SHEETS

Appendix E contains two different cover sheets. Cover sheet 1 is for the control category. Cover sheet 2 is for the treatment category.

Cover Sheet 1

Group Leader Name: _____

Other Team Member Names: _____

Group ID: _____

Course #: _____

Section #: _____

Please record the amount of time spent on the various sections of the project. Your grade is only determined by the quality of the class diagrams not on the amount of time.

Example:

Topic	Date	Start Time	End Time	Total Time
Class Diagram Creation	11/15/03	10:00 am	1:30 pm	3 hours 30 minutes
Class Diagram Creation	11/16/03	5:30 pm	7:15 pm	1 hour 45 minutes
			Total:	5 hours 15 minutes

Cover Sheet 2

Group Leader Name: _____

Other Team Member Names: _____

Group ID: _____

Course #: _____

Section #: _____

Please record the amount of time spent on the various sections of the project. Your grade is only determined by the quality of the class diagrams not on the amount of time.

Example:

Topic	Date	Start Time	End Time	Total Time
Relationship Analysis	11/13/03	10:00 am	1:30 pm	3 hours 30 minutes
Relationship Analysis	11/14/03	5:30 pm	7:15 pm	1 hour 45 minutes
Total:				5 hours 15 minutes

Topic	Date	Start Time	End Time	Total Time
Class Diagram Creation	11/15/03	10:00 am	1:30 pm	3 hours 30 minutes
Class Diagram Creation	11/16/03	5:30 pm	7:15 pm	1 hour 45 minutes
Total:				5 hours 15 minutes

APPENDIX F PROBLEM STATEMENT

Appendix F contains the problem statement used in the main experiment.

Topic: University On-Line Registration System

At the beginning of each semester, the registrar's office will provide a list of courses to students through a new on-line registration system. Information about each course, such as professor, department, and prerequisites will be included to help students make informed decisions.

The new system will allow students to review available courses and select four of them for the coming semester. In addition, each student will indicate two alternative choices in case a course becomes filled or canceled. No course will have more than ten students. No course will have fewer than three students. A course with fewer than three students will be canceled. If there is enough interest in a course, then a second session will be established.

Professors must be able to access the on-line system to indicate which courses they will be teaching. They will also need to see which students have signed up for their courses. Professors are expected to maintain a list of their research interests and projects as well as a list of publications. All students have access to each professor's research interests and projects lists. Each professor has access rights to their own publications list and can assign individual students permission rights to view these publications.

The registration process will last for three days. The first day will be freshman orientation and registration. All other students will arrive on the second day of the semester to register. The third day will be used to resolve any outstanding course assignment conflicts.

Once the course registration process is completed for a student, the registration system sends information to the billing system, so the student can be billed for the semester.

As a semester progresses, students must be able to access the on-line system to add or drop courses.

APPENDIX G PROBLEM STATEMENT SOLUTION

Appendix G contains the use-case analysis and class diagram to the problem statement of the main experiment listed in Appendix F.

Figure G1.1 Solution Use-case Analysis Diagram

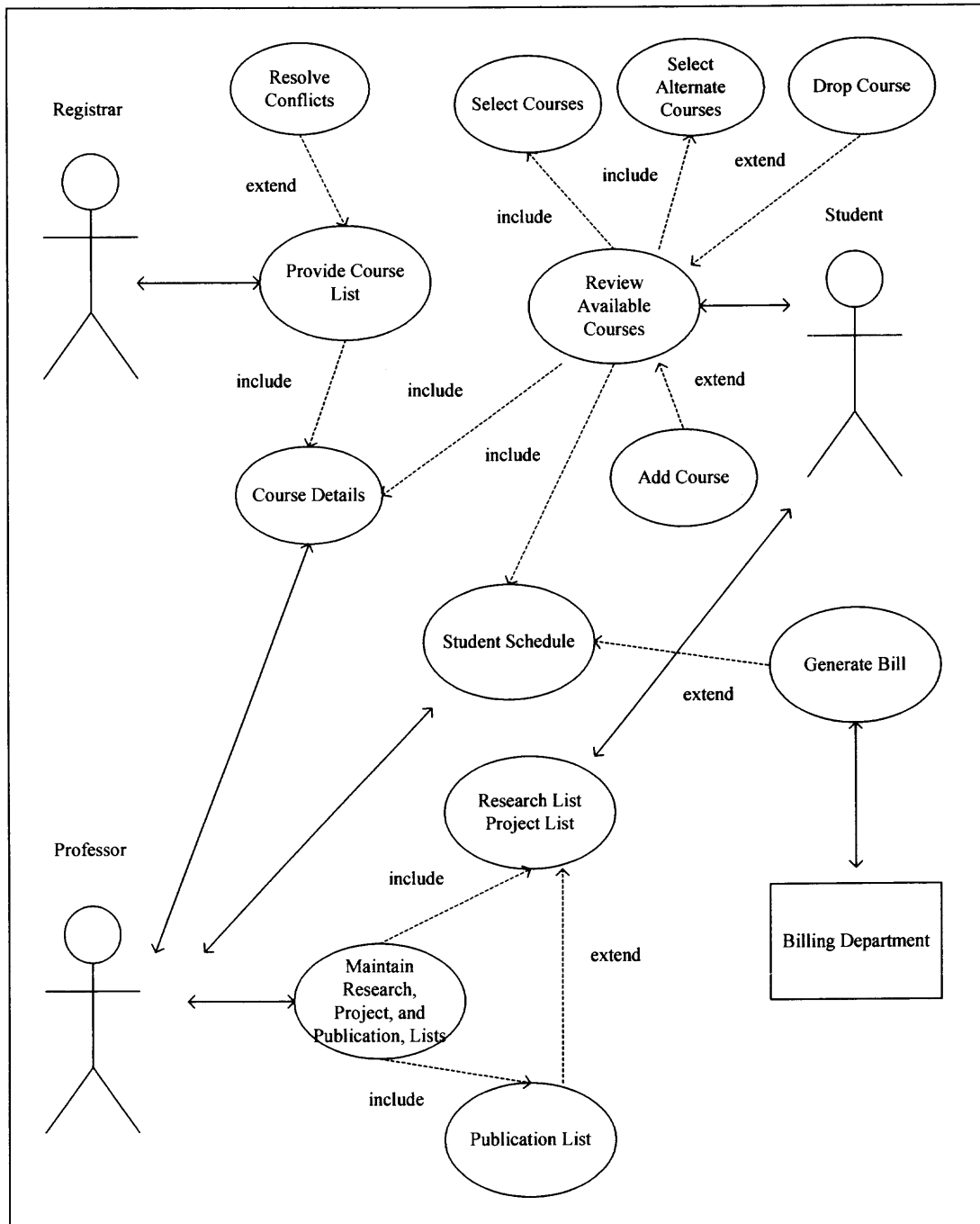


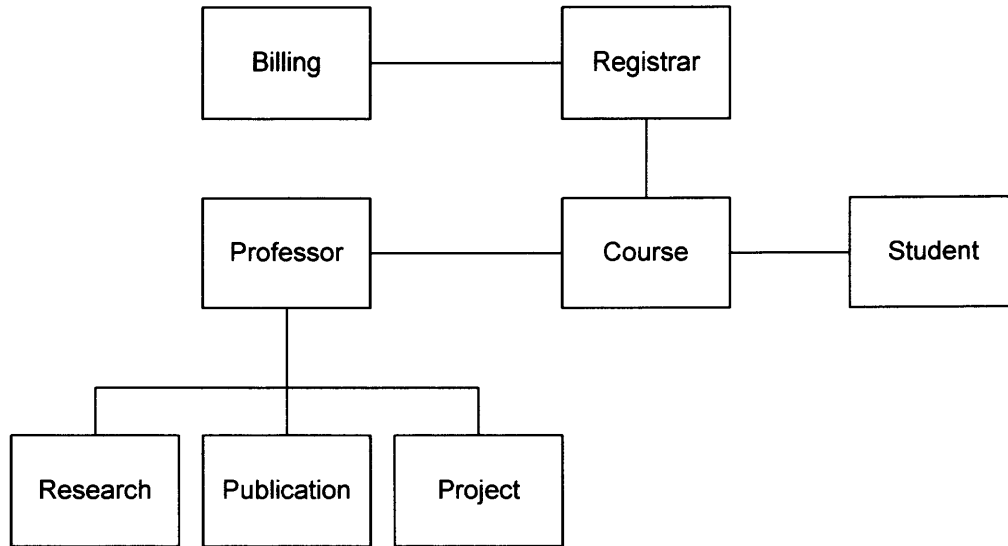
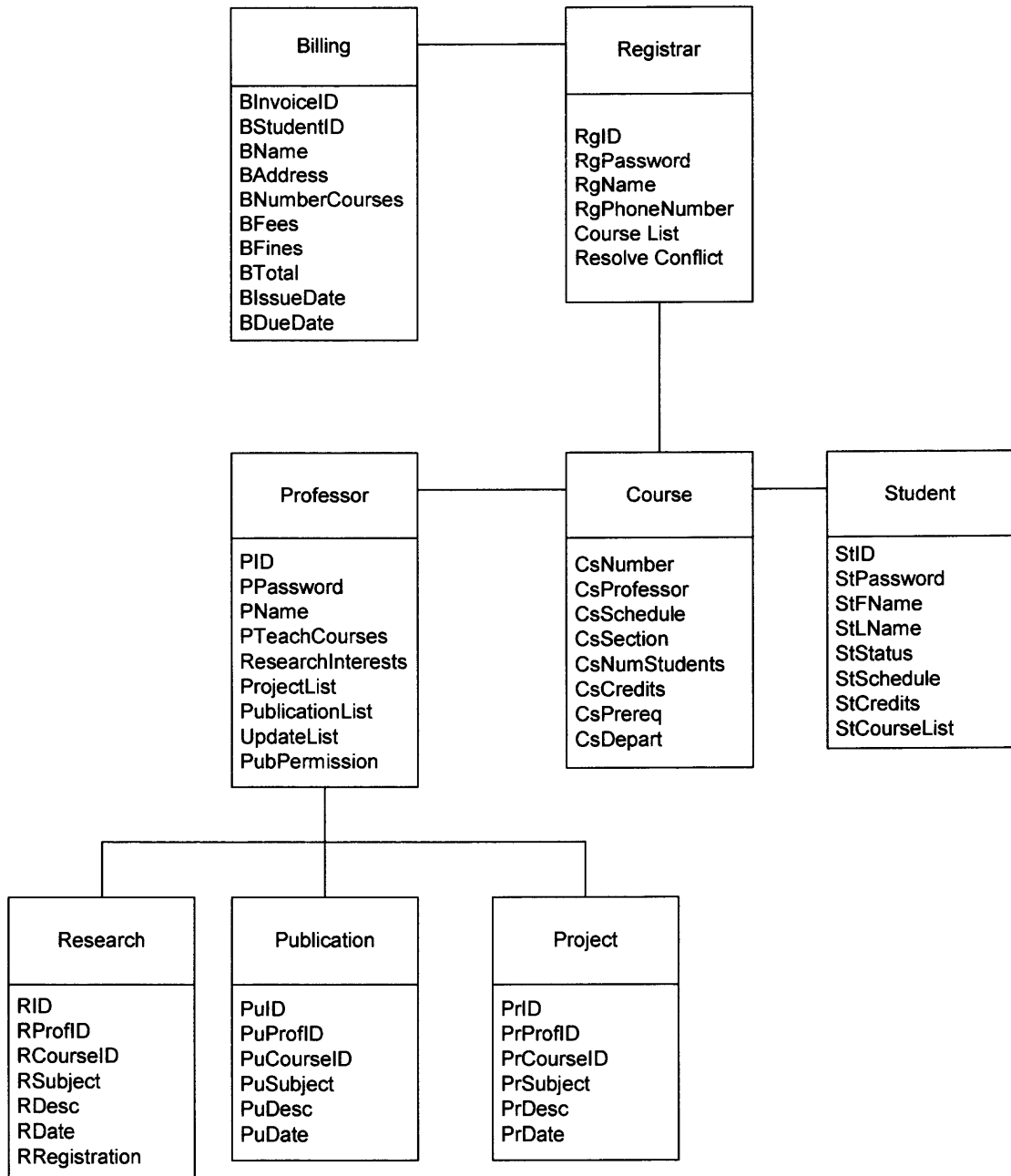
Figure G1.2 Solution Class Diagram Without Attributes

Figure G1.3 Solution Class Diagram



REFERENCES

- Abbott, R. (1983). "Program Design By Informal English Descriptions," *Communications of the ACM*, Vol. 26, No.11, pp. 882-895.
- Abdel-Hamid, T., & Madnick, S. (1989). "Lessons Learned from Modeling the Dynamics of Software Development," *Communications of the ACM*, Vol. 32, No.12, pp. 1426-1438.
- Abdel-Hamid, T., & Madnick, S. (1991). Software Project Dynamics, Prentice Hall, Englewood Cliffs, New Jersey.
- Allen, J. (1983). "Maintaining Knowledge About Temporal Intervals," *Communication of the ACM*, Vol. 26, No. 11, pp. 832-843.
- Amento, B., Terveen, L., & Hill, W. (2000). "Does Authority Mean Quality? Predicting Expert Quality Ratings of Web Documents," AT&T Shannon Laboratories.
- Anderson, J. (1989). "Automated Object-Oriented Requirements Analysis and Design," *The Association for Computing Machinery*, pp. 265-271.
- Arango, G. (1994). "A Brief Introduction to Domain Analysis," *IEEE Computer Society Press*, pp. 42-46.
- Astley, W., & Van de Ven, A. (1983). "Central Perspectives and Debates in Organization Theory," *Administration Science Quarterly*, Vol. 28, No. 2, pp. 245-273.
- Auramäki, E., Lehtinen, E., & Lyytinen, K. (1988). "A Speech-Act-Based Office Modeling Approach," *ACM Transactions on Office Information Systems*, Vol. 6, No. 2, pp. 126-152.
- Austin, J.L. (1962). How to Do Things with Words, Clarendon Press, London.
- Bailin, S. (2000). "Object-Oriented Requirements Analysis," Software Requirements Engineering, Second Edition, IEEE, Los Alamitos, California, pp. 334-355.
- Bandinelli, S. (1995). "Modeling and Improving an Industrial Software Process," *IEEE Transactions on Software Engineering*, Vol. 21, No. 5, pp. 440-454.
- Bansler, J., & Bødker, K. (1993). "A Reappraisal of Structured Analysis: Design in an Organizational Context," *ACM Transactions on Information Systems*, Vol. 11, No. 2, pp. 165-193.
- Barki, H., & Hartwick, J. (1994). "Measuring User Participation, User Involvement, and User Attitude," *MIS Quarterly*, pp. 59-82.

- Baroudi, J., Olson, M., & Ives, B. (1986). "An Empirical Study of the Impact of User Involvement on System Usage and Information Satisfaction," *Communications of the ACM*, pp. 232-238.
- Batra, D., Hoffler, J.A., & Bostrom, R.P. (1990). "Comparing Representations with Relational and EER Models," *Communications of the ACM*, Vol. 33, No. 2, pp. 126-139.
- Beck, K. (2000). Extreme Programming Explained, USA: Addison-Wesley, Massachusetts.
- Becker-Kornstaedt, U. (2001). "Towards Systematic Knowledge Elicitation for Descriptive Software Process Modeling," *Proceedings of PROFES*, pp. 1-18.
- Belkin, N., & Croft, W. (1987). "Retrieval Techniques," *Annual Review of Information Science and Technology (ARIST)*, Vol. 22, Chapter 4, pp. 109-131.
- Benbasat, I., & Schroeder, R. (1977). "An Experimental Investigation of Some MIS Design Variables," *MIS Quarterly* Vol.1, No. 1, pp. 37-50.
- Benbasat, I., & Dexter, A. (1986). "An Investigation of the Effectiveness of Color and Graphical Information Presentation Under Varying Time Constraints," *MIS Quarterly*, Vol. 10, No.1, pp. 59-83.
- Beraha, S., & Su, J. (1999). "Support for Modeling Relationships in Object-Oriented Databases," *Data & Knowledge Engineering*, Vol. 29, No. 3, pp. 227-257.
- Bieber, M. (1998). "Hypertext and Web Engineering," *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, ACM Press, pp. 277-278.
- Bieber, M., & Yoo., J. (1999). "Hypermedia: A Design Philosophy," *ACM Computing Surveys* 31(4es).
- Block, R. (1983). The Politics of Projects, Yourdon Press/Prentice-Hall, Englewood Cliffs, New Jersey.
- Bloomberg, M., & Weber, H. (1976). An Introduction to Classification and Number Building in Dewey, 19th Edition, Libraries Unlimited, Inc., Littleton, Colorado.
- Blum, B. (1994). "A Taxonomy of Software Development Methods," *Communications of the ACM*, Vol. 37, No. 11, pp. 82-94.
- Boehm, B. (1975). "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software," *IEEE Transactions on Software Engineering*, Vol. 1, No. 1, pp. 125-133.

- Boehm, B. (1976). "Quantitative Evaluation of Software Quality," 2nd International Conference on Software Engineering, Vol. 2, pp. 592-605.
- Boehm, B. (1981). Software Engineering Economics, Prentice-Hall, Englewood Cliffs, New Jersey.
- Boehm, B. (1988). "A Spiral Model of Software Development and Enhancement," IEEE Computer, Vol. 21, No. 5, pp. 61-72.
- Boehm, B., & Egyed, A. (1998). "Software Requirements Negotiation: Some Lessons Learned," IEEE Transactions on Software Engineering, pp. 503-506.
- Boggs, W., & Boggs, M. (2002). UML with Rational Rose 2002, SYBEX Inc., California.
- Booch, G., Martin, R.C., & Newkirk, J. (1998). Object-Oriented Analysis and Design with Applications, Addison Wesley, Massachusetts.
- Booch, G. (1994). Object-Oriented Analysis and Design, Second Edition, Benjamin/Cummings Publishing Company, California.
- Booch, G. (1996). "Object-Oriented Development," IEEE Transactions on Software Engineering, SE-12, 2, pp. 211-221.
- Booch, G., Jacobson, I., & Rumbaugh, J. (1998). The Unified Modeling Language Users Guide, Addison Wesley, Massachusetts.
- Borgida, A., Mylopoulos, J., & Wong, H. (1984). "Generalization/Specialization as a Basis for Software Specification," On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, pp. 87-117.
- Brachman, R. (1983). "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Networks," IEEE Computer, pp. 30-36.
- Brodie, M. (1981). "Association: A Database Abstraction for Semantic Modeling," Entity-Relationship Approach to Information Modeling and Analysis, P.P. Chen (ed.), ER Institute, pp. 583-608.
- Brodie, M. (1984). On Conceptual Modeling, Springer-Verlag, New York.
- Brooks, F. (1987). "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, pp. 10-19.
- Brown, R. (1989) "Creativity: What are We to Measure?," Handbook of Creativity, Plenum Press, New York.

- Buneman, O.P., & Nikhil, R. (1984). "The Functional Data Model as its Uses for Interaction with Databases," In *On Conceptual Modeling, Perspectives, from Artificial Intelligence, Databases, and Programming Languages*, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, Eds. Springer-Verlag, New York, pp. 359-380.
- Bunge, M. (1979). Treatise on Basic Philosophy: Vol. 4: Ontology II: A World of Systems, Reidel Publishing Co., Inc., New York.
- Burt, P.V, & Kinnucan, M.T. (1990). "Information Models and Modeling Techniques for Information Systems," *Annual Review of Information Science and Technology*, Vol. 25, pp. 175-208.
- Butler, R., & Finelli, G. (1991). "The Infeasibility of Experimental Quantified Life-Critical Software Reliability," *Proc. SIGSOFT*.
- Cafasso, R. (1994). "Few IS Projects Come In On Time, On Budget," *Computerworld*, Vol. 28, No. 50, page 20.
- Capps, L. (2002). "Toward Pervasive Computing: Managing for Creativity," <http://www.DMReview.com>, (21 July 2002).
- Carter, K., Cushing, K., Sabers, D., Stein, P., & Berliner, D. (1988). "Expert-Novice Differences in Perceiving and Processing Visual Classroom Information," *Journal of Teacher Education*, pp. 25-31.
- Catania, J., & Bieber, M. (2003) "Relationship Analysis: A Technique to Explicitly Identify the Relationship Structure," *Symposium on Research in Systems Analysis and Design*, Miami, Florida.
- Chen, P. (1976). "The Entity-Relationship Model – Toward a Unified View of Data," *ACM Transactions on Database Systems*, Vol. 1, No. 1.
- Chung, W., & Guinan, P. (1994). "Effects of Participative Management on the Performance of Software Development Teams," *SIGCPR*, Alexandria, Virginia, pp. 252-260.
- Coad, P., & Yourdon, E. (1990). Object-Oriented Analysis, Yourdon Press / Prentice-Hall, Englewood Cliffs, New Jersey.
- Cobb, M., & Petry, F. (1998). "Modeling Spatial Relationships within a Fuzzy Framework," *Journal of the American Society for Information Science*, Vol. 49, No. 3, pp. 253-266.
- Cockburn, A. (1999). "Characterizing People as Non-Linear, First-Order Components in Software Development," *Humans and Technology Report*, TR 99.05, pp. 1-19.

- Cockburn, A. (1997). "Structuring Use-Cases with Goals," *Journal of Object-Oriented Programming*, pp. 1-18.
- Cockburn, A. (2000). Writing Effective Use Cases, Addison-Wesley, Massachusetts.
- Cockburn, A. (1998). "Software Development and Process," ECOOP.
- Codd, E.F. (1979). "Extending the Database Relational Model to Capture More Meaning," *ACM Trans. Database Systems*, Vol. 4, No. 4, pp. 397-434.
- Cohen, S. (1992). "Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain," CMU/SEI-91-TR-28, Carnegie Mellon University.
- Coleman, D. (1998). "A Use-Case Template: Draft for Discussion," *Fusion Newsletter*.
- Coleman, D., Hayes, F., & Bear, S. (1992). "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," *IEEE Transactions in Software Engineering* Vol. 18, No. 1, pp. 9-18.
- Connolly, T., Jessup, L., & Valacich, J. (1990). "Effects of Anonymity and Evaluation Tone on Idea Generation in Computer-Mediated Groups," *Management Science*, Vol. 36, No. 6, pp. 305-319.
- Couger, J.D. (1973). "Evolution of Business System Analysis Techniques," *Comput. Surv.*, Vol. 5, No. 3, pp. 167-198.
- Couger, J.D. (1990). "Ensuring Creative Approaches in Information System Design," *Journal of Information Systems Management*, Vol. 4, pp. 36-41.
- Couger, J.D. (1993). "Unstructured Creativity in Information Systems Organization," *MIS Quarterly*, Vol. 17, No. 4, pp. 375-398.
- Couger, J.D. (1994). "Enhancing the Climate for Creativity for Software Designers," *Journal of Creativity and Innovation Management*, Vol. 3, Issue 1, pp. 54-59.
- Daft, R., & Lengel, R. (1986). "Organizational Information Requirements, Media Richness, and Structural Design," *Management Science*, Vol. 32, No. 5, pp. 554-571.
- Daniels, W., & Martin, C. (2000). "Dewey Applications for the Simple Arrangement of a Link Library: The Case of ScienceNet," *Journal of Internet Cataloging*, Vol. 3, No. 1, pp. 67-77.
- Davis, A. (1988). "A Strategy for Comparing Alternative Software Development Life Cycle Models," *IEEE Trans. Software Engineering*, Vol. 14, No. 10, pp. 1453-1461.

- Davis, A. (1993). "Identifying and Measuring Quality in a Software Requirements Specification," Proceedings of the 1st International Software Metrics Symposium, pp. 141-152.
- Davis, G. (1982). "Strategies for Information Requirements Determinations," IBM Systems Journal, Vol. 21, No. 1, pp. 4-30.
- Davis, G., & Olson, M. (1985). Management Information Systems; Conceptual Foundations, Structure, and Development, McGraw-Hill, Boston, Massachusetts.
- DeChampeaux, D., Lea, D., & Faure, P. (1993). Object-Oriented System Development, Addison-Wesley, Reading, Massachusetts.
- Deek, F. P. (1999). "The Software Process: A Parallel Approach through Problem Solving and Program Development," Journal of Computer Science Education, Volume 9, Number 1, pp. 43-70.
- Deek, F.P., DeFranco-Tommarello, J., & McHugh, J. (2001). "A Model for a Collaborative Technologies in Manufacturing," In Review for the International Journal of Computer Integrated Manufacturing.
- DeMarco, T. (1978). Structured Analysis and System Specification, Yourdon Press, New York.
- DeSanctis, G. (1984). "Graphs as Decision Aids," Decision Sciences, Vol. 15, pp. 463-487.
- DeSanctis, G., & Gallupe, B. (1987). "A Foundation for the Study of Group Decision Support Systems," Management Science, Vol. 33, No. 5, pp. 589-609.
- Dickson, G.W., DeSanctis, G., & McBride, D.J. (1986). "Understanding the Effectiveness of Computer Graphics for Decision Support: A Cumulative Experimental Approach," Communications of the ACM, Vol. 29, No. 1, pp. 40-47.
- Donnelly, J., Gibson, J., & Ivancevich, J. (1998). Fundamentals of Management, Tenth Edition, McGraw-Hill, Boston, Massachusetts.
- Egenhofer, M., & Herring, J. (1990). "Categorizing Binary Topological Relations Between Regions, Lines, and Points in Geographic Databases," Technical Report, Department of Surveying Engineering, University of Maine.
- Elmasri, R., & Navathe, S. (2000). Fundamental of Database Systems, Addison-Wesley, Reading, Massachusetts.

- Embley, D., Kurtz, B., & Woodfield, S. (1992). Object-Oriented Systems Analysis: A Model-Driven Approach, Prentice-Hall, Englewood Cliffs, New Jersey.
- Engler, N. (1996). "Bringing in the Users," Computerworld.
- Faulk, S. (2000). "Software Requirements: A Tutorial," Software Requirements Engineering, Second Edition, IEEE, Los Alamitos, California, pp. 158-179.
- Favaro, J. (1997). "Standardizing Production of Domain Components," Standard View, Vol.5, No. 2, pp. 66-69.
- Feldman, P. & Miller, D. (1986). "Entity Model Clustering: a Data Model by Abstraction," Computer Journal, Vol. 29, No.4, pp. 348-360.
- Fillmore, C.J. (1968). "The Case for Case," Universals in Linguistic Theory.
- Firesmith, D. (1993). Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach, Wiley, New York.
- Fornara, N., & Colombetti, M. (2003). "Defining Interaction Protocols using a Commitment-Based Agent Communication Language," AAMAS, Melbourne, Australia, pp. 520-527.
- Fowler, M. & Scott, K. (1997). UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, Reading, Massachusetts.
- Frank, A. (1998). "Different Types of Times in GIS," Spatial and Temporal Reasoning in Geographic Information Systems, Eds. Egenhofer, M., and Golledge, R., Chapter 3, pp. 41-62.
- Frewin, G., & Hatton, B. (1986). "Quality Management: Procedures and Practices," Journal of Software Engineering, Vol. 1, No. 1, pp. 29-38.
- Gallupe, B., DeSanctis, G., & Dickson, G. (1988). "Computer-Based Support for Group Problem-Finding: An Experimental Investigation," MIS Quarterly, Vol.12, No. 2, pp. 277-296.
- Gandhi, M., Robertson, E.L., & Gucht, D.V. (1994). "Leveled Entity Relationship Model," Proceedings of the 13th International Conference on the Entity-Relationship Approach, Manchester, United Kingdom, pp. 420-433.
- Gane, C., & Sarson, T. (1979). Structured Systems Analysis: Tools and Techniques, Prentice-Hall, Englewood Cliffs, New Jersey.
- Gause, D., & Weinberg, G. (1989). Exploring Requirements: Quality Before Design, Dorset House Publishing, New York.

- Gero, J. (1994). Introduction: Creativity and Design, Kluwer Academic Publishers.
- Giles, C.L., Bollacker, K., & Lawrence, S. (1998). "CiteSeer: An Automatic Citation Indexing System," *Digital Libraries 98: Third ACM Conf. on Digital Libraries*, ACM Press, New York, pp. 89-98.
- Gill, B., & Pidduck, A. (2001). "IT Staffing and Retention: A Success Story," *ACM 1-58113-363-1/01/04*, pp. 87-92.
- Goguen, J., & Linde, C. (1993). "Techniques for Requirements Elicitation," *Proceedings from the International Symposium on Requirements Engineering*, pp. 152-164.
- Goldstein, R.C., & Storey, V. (1999). "Data Abstractions: Why and How?," *Data and Knowledge Engineering*, Vol. 29, pp. 293-311.
- Goldstein, R.C., & Storey, V. (1990). "Some Findings on the Intuitiveness of Entity-Relationship Constructs," *Proceedings of the 8th International Conference on Entity-Relationship Approach to Database Design and Querying*, Toronto, Canada, pp. 9-23.
- Gomaa, H. (1990). "The Impact of Prototyping on Software System Engineering," System and Software Requirements Engineering, IEEE, Los Alamitos, California pp. 479-488.
- Gomes, P., Pereira, F., Seco, N., Ferreira, J., & Bento, C. (2001). "Supporting Creativity in Software Design," CISUC.
- Gonzales, R., & Wolf, A. (1996). "A Facilitator Method for Upstream Design Activities with Diverse Stakeholders," *Proceedings of the International Conference on Requirements Engineering*, IEEE Computer Society, pp. 190-197.
- Gordon, V.S., & Bieman, J.M. (1995). "Rapid Prototyping: Lessons Learned," *IEEE Software*, Vol. 12, No. 1, pp. 85-95.
- Graham, I. (1994). Migrating to Object Technology, Addison-Wesley, Reading, Massachusetts.
- Guilford, J.P. (1950). "Creativity," *American Psychologist*, 5, pp. 444-454.
- Guilford, J.P. (1956). "The Structure of Intellect," *Psychological Bulletin* 53(4), pp. 267-293.
- Guilford, J.P. (1967). The Nature of Human Intelligence, McGraw-Hill, New York.
- Hall, A. (1990). "Seven Myths of Formal Methods," *IEEE Software*, Vol. 7, No. 5, pp. 11-20.

- Hammer, M., & McLeod, D. (1981). "Database Description with SDM: A Semantic Database Model," *ACM Trans. Database Systems*, Vol. 6, No. 3, pp. 351-386.
- Harter, D., Slaughter, S. & Krishnan, M. (1998). "The Life Cycle Effects of Software Process Improvement: A Longitudinal Analysis," *A Carnegie Mellon University Article*, pp. 346-351.
- Haywood, M. (1998). Managing Virtual Teams: Practical Techniques for High-Technology Project Managers, Artech House, Boston, Massachusetts.
- Henderson-Sellers, B & Edwards, J. (1990). "The Object-Oriented Systems Life Cycle," *Communications of the ACM*, Vol. 33, No. 9, pp. 142-159.
- Henderson-Sellers, B & Edwards, J. (1994). Book Two of Object-Oriented Knowledge, Englewood Cliffs, New Jersey.
- Henderson-Sellers, B. (1997). "OPEN Relationships-Compositions and Containments," *Journal of Object-Oriented Programming*, pp. 51-72.
- Henderson-Sellers, B. (1998). "OPEN Relationships-Associations, Mappings, Dependencies, and Uses" *Journal of Object-Oriented Programming*, pp. 49-57.
- Herzberg, F., Mausner, B., & Snyderman, B. (1959). The Motivation to Work, Second Edition, Wiley, New York.
- Highsmith, J. (1999). Adaptive Software Development, Dorset House.
- Hillerbrand, E., & Claiborn, C. (1990). "Examining Reasoning Skill Differences Between Expert and Novice Counselors," *Journal of Counseling & Development*, Vol. 68 , pp. 684-691.
- Hitchcock, P. (1980). "Data Dictionaries in Open System Communication," *IEEE Transaction On Software Engineering*, pp. 133-134.
- Hooks, I. (2001). "Managing Requirements," *NJIT Requirements Engineering Handout*, pp. 1-8.
- Huffman, L., & Rosenberg, H. (1998). "Doing Requirements Right the First Time," *CrossTalk, The Journal of Defense Software Engineering*.
- IEEE, (1998). "IEEE Guide for Information Technology-System Definition-Concept of Operations (ConOps) Document," *IEEE-SA Standards Board*.
- ITU (1994). "Criteria for the Use and Applicability of Formal Descriptive Techniques: Message Sequence Charts (MSC)," *International Telecommunication Union, Z 120*, pp. 1-7.

- Jaeschke P., Oberweis, A. & Stucky, W. (1993). "Extending ER Model Clustering by Relationship Clustering," Proceedings of the 12th International Conference on the Entity-Relationship Approach, Berlin, Germany, pp. 451-462.
- Jackson, M. (1983). System Development, Prentice-Hall, Englewood Cliffs, New Jersey.
- Jacobs, S. (1999). "Introducing Measurable Quality Requirements: A Case Study," IEEE International Symposium on Requirements Engineering.
- Jacobson, I., Christerson, M., Johnsson, P., & Overgaard, G. (1992). Object-Oriented Software Engineering: A Use Case Driven Approach, Addison-Wesley, Reading, Massachusetts.
- Jarvenpaa, S., & Machessky, J. (1989). "Data Analysis and Learning: An Experimental Study of Data Modeling Tools," International Journal of Man-Machine Studies, Vol. 31, pp. 367-391.
- Jarvenpaa, S., & Dickson, G.W. (1988). "Graphics and Managerial Decision Making: Research Based Guidelines," Communications of the ACM, Vol. 31, No. 6, pp. 764-774.
- Jarvenpaa, S., Dickson, G.W., & DeSanctis, G. (1985). "Methodological Issues in Experimental IS Research: Experiences and Recommendation," MIS Quarterly, Vol. 9, No. 2, pp. 141-156.
- Jasper, R., & Uschold, M. (1999). "A Framework for Understanding and Classifying Ontology Applications," Proceeding of the IJCAI-99 Ontology Workshop.
- Kang, K. (1990). "Feature-Oriented Domain Analysis Feasibility Study," CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie-Mellon University.
- Kay, P., Berlin, P., & Merrifield, W. (1991). "Biocultural Implications of Systems of Color Naming," Journal of Linguistic Anthropology 1(1), pp. 12-25.
- Kentworthy, E. (1997). "Use-Case Modeling: Capturing User Requirements," pp. 1-11.
- King, R., & McLeod, D. (1984). "A Unified Model and Methodology for Conceptual Database Design," In On Conceptual Modeling, Perspectives from Artificial Intelligence, Databases, and Programming Languages, M.L. Brodie, J. Mylopoulos, and J.W. Schmidt, Eds. Springer-Verlag, New York, pp. 313-327.
- Kobryn, C. (2000). "Modeling Components and Frameworks with UML," Communications of the ACM, Vol. 43, No. 1, pp. 31-38.
- Kolb, D.A. (1984). Experiential Learning, Prentice Hall, Englewood Cliffs, NJ.

- Kotonya, G., & Sommerville, I. (1996). "Requirements Engineering with Viewpoints," *Software Engineering Journal* Vol. 11, No. 1, pp. 5-18.
- Krishnamurthy, V., Su, S., Lam, H., Mitchell, Z., & Bancmeyer, E. (1987). "A Distributed Database Architecture for an Integrated Manufacturing Facility," *Proceedings of International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, IEEE, New York, pp. 4-13.
- Larman, C. (2001). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process (2nd Edition), Prentice Hall, Englewood Cliffs, New Jersey.
- Lederer, A., Maupin, D., Sena, M., & Zhuang, Y. (1998). "The Role of Ease of Use, Usefulness and Attitude in the Prediction of World Wide Web Usage," *ACM*, pp. 195-204.
- Leffingwell, D. (2000). "Features, Use-Cases, Requirements, Oh My!," *The Rational Edge*, December, pp. 1-13.
- Leffingwell, D., & Widrig, D. (1999). Managing Software Requirements: A Unified Approach, Addison-Wesley, Reading, Massachusetts.
- Loucopoulos, P., & Karakostas, V. (1995). Systems Requirements Engineering, McGraw-Hill, Boston, Massachusetts.
- Lucas, H.C. (1981). "An Experimental Investigation of the Use of Computer Based Graphics in Decision Making," *Management Science*, Vol. 27, No. 7., pp. 757-768.
- Lung, C., & Urban, J. (1995). "An Approach To The Classification Of Domain Models In Support for Analogical Reuse," *SSR*, Seattle, Washington, pp. 169-178.
- Macaulay, L. (1996). Requirements Engineering, Springer-Verlag, New York, NY.
- Martin, R.C. (2002). Agile Software Development, Principles, Patterns, and Practices, Prentice Hall, Englewood Cliffs, New Jersey.
- Martin, J., & Odell, J. (1995). Object-Oriented Methods: A Foundation, Prentice Hall, Englewood Cliffs, New Jersey.
- Meeker, M. (1969). The Structure of Intellect: Its Interpretations and Uses, Merrill Publishing, Columbus, Ohio.

- Moody, D.L., Sindre, G., Brasethvik, T., & Sølvsberg, A. (2003). "Evaluating the Quality of Information Models: Empirical Testing of a Conceptual Model Quality Framework," Proceedings of the 25th International Conference on Software Engineering (ICSE'03), pp. 295-305.
- Moody, D.L. (1996). "Graphical Entity Relationship Models: Toward More User Understanding Representation of Data," Proceedings of the 15th International Conference on Conceptual Modeling, Cottbus, Germany, pp. 227-244.
- Moore, J. (2000). "One Road to Turnover: An Examination of Work Exhaustion in Technology Professionals," MIS Quarterly, Vol. 42, No. 1, pp. 87-92.
- Motschnig-Pitrik, R., & Storey, V. (1995). "Modeling of Set Membership: The Notion and the Issues," Data & Knowledge Engineering, Vol. 16, pp. 147-185.
- Murine, G. (1984). "Measuring Software Product Quality," Quality Progress, Vol. 17, No. 5, pp. 16-20.
- Musa, J., & Ackerman, A. (1989). "Quantifying Software Validation: When to Stop Testing?," IEEE Software, Vol. 6, No. 3, pp. 19-27.
- Myers, M. (1999). "Investigating Information Systems with Ethnographic Research," Communications of the Associations for Information Systems.
- Mylopoulos, J., Bernstein, P.A., & Wong, H.K.T. (1980). "A Language Facility for Designing Database Intensive Applications," ACM Transactions Database Systems, Vol. 5, No. 2, pp. 186-207.
- Mylopoulos, J. (1998). "Information Modeling in the Time of the Revolution," Information System, Vol. 23, No. 3/4, pp. 127-155.
- Mylopoulos, J., Chung, L., & Yu, E. (1999). "From Object-Oriented To Goal-Oriented Requirements Analysis," Communications of the ACM, Vol. 42, No. 1, pp. 31-37.
- Nagasundaram, M., & Bostrom, R.P. (1995). "The Structuring of Creative Processes Using GSS: A Framework for Research," Journal of Management Information Systems, Vol. 11, No. 3, pp. 89-116.
- Nallon, J. (1994). "Implementation of NSWC Requirements Traceability Models," IEEE Transaction on Software Engineering, pp. 15-22.
- Neelameghan, A., & Maitra, R. (2000). "Non-Hierarchical Associative Relationships Among Concepts: Identification and Typology," Part A of FID/CR report No. 18, Bangalore: FID/CR Secretariat Document Research and Training Center.

- Nerson, J. (1992). "Applying Object-Oriented Analysis and Design," *Communications of the ACM*, Vol. 35, No. 9, pp. 63-74.
- Ng, P. & Yeh, R. (1990). "Software Requirements: A Management Perspective," *System and Software Requirements Engineering*, IEEE, Los Alamitos, California, pp. 450-461.
- Nixon, B., Chung, L., Lauzen, I., Borgida, A., Mylopoulos, J., & Stanley, M. (1987). "Implementation of a Compiler for a Semantic Data Model: Experience with Taxis," *Proceedings of the ACM SIGMOD Conferences (San Francisco, California.)*, ACM, New York, pp. 118-131.
- Nosek, J. (1998). "The Case for Collaborative Programming," *Communications of the ACM*, Vol. 41, No. 3, pp. 105-108.
- Nuseibeh, B., Kramer, J., & Finkelstein, A. (1994). "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification," *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, pp. 760-773.
- O'Brien, P. (1983). "An Integrated Interactive Design Environment for Taxis," *Proceedings of SOFTFAIR: A Conference on Software Development Tools, Techniques, and Alternatives (Silver Spring, Maryland.)*, IEEE, New York, pp. 298-306.
- Ocker, R., Fjermestad, J., Hiltz, S.R., & Johnson, K. (1998). "Effects of Four Modes of Group Communication on the Outcomes of Software Requirements Determination," *Journal of Management Information Systems*.
- Odell, J. (1994). "Six Different Kinds of Composition," *Journal of Object-Oriented Programming*, pp. 10-15.
- Palmer, J. (2000). "Traceability," *Software Requirements Engineering*, Second Edition, IEEE, Los Alamitos, California, pp. 412-422.
- Parsons, J., & Wand, Y. (1997). "Choosing Classes in Conceptual Modeling," *Communications of the ACM* Vol. 40, No. 6, pp. 63-69.
- Partridge, D., & Rowe, J. (1994). *Computers and Creativity*, Intellect Books.
- Paré, G., & Dubé, L. (1999). "Virtual Teams: An Exploratory Study of Key Challenges and Strategies," *Proceedings of the 20th International Conference on Information Systems*, pp. 479-483.
- Peckham, J., & Maryanski, F. (1988). "Semantic Data Models," *ACM Computing Surveys*, Vol. 20, No. 3, pp. 153-189.

- Prieto-Diaz, R. (1991). "Domain Analysis and Software Systems Modeling," IEEE Computer Society Press.
- Prietula, M.J., & March, S.T. (1991). "Form and Substance in Physical Database Design: An Empirical Study," *Inf. Syst. Res.*, Vol. 2, No. 4, pp. 287-314.
- Ranganathan, S.R. (1965). The Colon Classification, Vol. IV, The Rutgers Series on Systems for the Intellectual Organization of Information, New Brunswick, New Jersey.
- Rauh, O., & Stickel, E. (1992). "Entity Tree Clustering- A Method for Simplifying ER Design," *Proceedings of the 11th International Conference on the Entity-Relationship Approach*, pp. 62-78.
- Rodriguez, M., Egenhofer, M., & Rugg, R. (1999). "Assessing Semantic Similarities Among Geospatial Feature Class Definitions," *Interop 1999, Zurich, Switzerland*, in: A. Vckovski (editor), *Lecture Notes in Computer Science*, New York.
- Rombach, H. (1990). "Software Specifications: A Framework Curriculum Model," SEI-CM-11-2-1, Software Engineering Institute, Pittsburgh, Pennsylvania.
- Rosch, E. (1978). Cognition and Categorization, Erlbaum, Hillsdale, New Jersey.
- Rosenthal, R. & Rosnow, R. (1991). Essentials of Behavior Research Methods and Data Analysis, Second Edition, McGraw-Hill, New York.
- Ross, D. (1986). "Classifying Ada Packages," *Ada Letters*, Vol. 6, No. 4.
- Royce, W. (1970). "Managing The Development Of Large Software Systems: Concepts and Techniques," Wescon.
- Rumbaugh, J. (1991). Object-Oriented Modeling and Design, Prentice-Hall, New Jersey.
- Rumbaugh, J. (1994). "Getting Started: Using Use Cases to Capture Requirements," *Object-Oriented Programming*, pp. 8-12.
- Rundley, N., & Miller, W. (1994). "DOORS to the Digitized Battlefield: Managing Requirements Discovery and Traceability," *CSESAW*, pp. 23-28.
- Sabin, R. E., & Sabin, E. (1994). "Collaborative Learning in an Introductory Computer Science Course," *SIGCSE Symposium on Computer Science Education*, pp. 304-308.
- Saleem, N. (1996). "An Empirical Test of the Contingency Approach to User Participation in Information Systems Development," *Journal of Management Information Systems*, Vol. 13, No. 1, pp. 145-166.

- Schenk, K.D., Vitalari, N., & Davis, K. (1998). "Differences Between Novice and Expert Systems Analysts: What Do We Know and What Do We Do?," *Journal of Management Information Systems*, Vol. 15, No. 1, pp. 9-50.
- Searle, J.R., and Vanderveken, D. (1985). Foundations of Illocutionary Logic, Cambridge University Press, London, 1985.
- Selic, B., Gullekson, G. & Ward, P. (1994). Real Time Object-Oriented Modeling, Wiley, New York.
- Selvin, A. (1999). "Supporting Collaborative Analysis and Design with Hypertext Functionality," *Journal of Digital Information*, Vol. 1, Issue 4, pp. 1-20.
- Shaft, T. M. & Vessey, I. (1998). "The Relevance of Application Domain Knowledge: Characterizing the Computer Program Comprehension Process," *Journal of Management Information Systems*, Vol. 15 No. 1, pp. 51-78.
- Shipman, D.W. (1981). "The Functional Data Model and the Data Language DAPLEX," *ACM Trans. Database Systems*, Vol. 6, No. 1, pp.140-173.
- Shlaer, S., & Mellor, S. (1992). Object Life-Cycles: Modeling the World in States, Prentice-Hall, Englewood Cliffs, New Jersey.
- Shoval, P., & Frumermann, I. (1994). "OO and EER Conceptual Schemas: A Comparison of User Comprehension," *Journal of Database Management*, Vol. 5, No. 4, pp. 28-38.
- Shoval, P., & Shiran, S. (1997). "Entity-Relationship and Object-Oriented Data Modeling – An Experimental Comparison of design Quality," *Data & Knowledge Engineering*, Vol. 21, pp. 297-315.
- Simon, H.A., and Ericson, K. (1984). "Protocol Analysis: Verbal Reports as Data," MIT.
- Simos, M. (1995). "Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle," SSR, Seattle, Washington, pp. 196-205.
- Simonson, S., & Kensing, F. (1997). "Using Ethnography in Contextual Design," *Communications of the ACM*, Vol. 40, No., 7, pp. 82-88.
- Smith, J. (2001). "A Comparison of RUP and XP," *Rational Software White Paper*, pp. 1-21.
- Smith, J., & Smith, D. (1977). "Database Abstractions: Aggregation and Generalization," *ACM Transactions on Database Systems*, Vol. 2, No. 2, pp. 105- 133.

- Sommerville, I. (2001). Software Engineering, Sixth Edition, Addison-Wesley Publishers, Massachusetts.
- Spence, M. T., & Brucks, M. (1997). "The Moderating Effects of Problem Characteristics on Experts' and Novices' Judgements," *Journal of Marketing Research*, Vol. XXXIV, pp. 233-247.
- Straub, D. W. (1989). "Validating Instruments in MIS Research," *MIS Quarterly*, pp. 147-169.
- Su, S.Y.W. (1983). "SAM: A Semantic Association Model for Corporate and Scientific-Statistical Databases," *Information Science*, Vol. 29, pp. 151-199.
- Suchman, L. (1983). "Office Procedures as Practical Action," *ACM Transaction on Office Information Systems*, Vol. 1, No. 3, pp. 320-328.
- Suchman, L., & Jordan, B. (1990). "Interactional Troubles in Face-To-Face Survey Interviews," *Journal of the American Statistical Association*, Vol. 89, No. 409, pp. 232-241.
- Sun, L. (2002). "An Experimental Comparison of the Maintainability of Structured Analysis and Object-Oriented Analysis," Department of Information and Software Engineering Archive, George Mason University, pp. 1-11.
- Svoboda, C. (1990). "Structured Analysis," System and Software Requirements Engineering, IEEE, Los Alamitos, California, pp. 218-237.
- Sycara, K., & Navinchandra, D. (1991). "Influences: A Thematic Abstraction for Creative Use of Multiple Cases," First European Workshop on Case-Based Reasoning.
- Takagaki, K., & Wand, Y. (1991). "An Object-Oriented Information System Model Bases on Ontology," Proceedings of the IFIP Working Group 8.1 Conference, Quebec.
- Taylor, F.W. (1967). The Principles of Scientific Management, W.W. Norton & Company.
- Teorey, T., Yang, D., & Fry, J.P. (1986). "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," *ACM Comput. Surv.*, Vol. 18, No. 2, pp. 197-222.
- Teorey, T., Wei, G., Bolton, D., & Koenig, J. (1989). "ER Model Clustering as an Aid for User Communication and Documentation in Database Design," *Communication of the ACM*, Vol. 32, No. 8, pp. 975-987.

- Thayer, R., & Dorfman, M. (2000). "Software System Engineering Process Models," Software Requirements Engineering, Second Edition, IEEE, Los Alamitos, California, pp. 453-455.
- Topi, H., & Ramesh, V. (2002). "Human Factors Research on Data Modeling: A Review of Prior Research, an Extended Framework and Future Research Directions," Journal of Database Management, Vol. 13, No. 2, pp. 3-19.
- Tosca, S.P. (2000). "A Pragmatics of Links," Hypertext, San Antonio, Texas, pp. 77-84.
- Tullis, T.S. (1981). "An Evaluation of Alphanumeric, Graphic, and Color Displays," Human Factors, Vol. 23, No. 5, pp. 541-550.
- Turoff, M., Rao, U., & Hiltz, S.R. (1991). "Collaborative Hypertext in Computer Mediated Communications," Proceedings of the 24th Annual Hawaii International Conference on System Sciences, Vol. IV.
- Uschold, M. (1998). "Knowledge Level Modeling: Concepts and Terminology," Knowledge Engineering Review, Vol. 13, No. 1.
- Vertal, M. (1994). "Extending IDEF: Improving Complex Systems with Executable Modeling," Proc. Ann. Conf. for Business Re-engineering.
- Vessey, I. (1991). "Cognitive Fit: A Theory-Based Analysis of the Graphs Versus Tables Literature," Decision Sciences, Vol. 22, pp. 219-240.
- Vessey, I. & Conger, S. (1994). "Requirements Specification: Learning Object, Process, and Data Methodologies," Communications of the ACM, Vol. 37, No. 5, pp. 102-113.
- Vessey, I. (1985). "Expertise in Debugging Computer Programs: A Process Analysis," Journal of Man-Machine Studies, Vol. 23, pp. 459-494.
- Wallace, D., & Ippolito, L. (2000). "Verifying and Validating Software Requirements Specifications," Software Requirements Engineering, Second Edition, IEEE, Los Alamitos, California pp. 437-452.
- Wand, Y., Storey, V., & Weber, R. (1999). "An Ontological Analysis of the Relationship Construct in Conceptual Modeling," ACM Transactions on Database Systems, Vol. 24, No. 4, pp. 494-528.
- Wand, Y., Monarchi, D., Parsons, J., & Woo, C.C. (1995). "Theoretical Foundations for Conceptual Modeling in Information Systems Development," Decision Support Systems, Vol. 15, pp. 285-304.

- Wand, Y., & Weber, R. (1995). "On the Deep Structure of Information Systems," *Eur. J. Inf. Syst.*, Vol. 5, pp. 203-223.
- Weber, R., & Zhang, Y. (1996). "An Ontological Evaluation of NIAM's Grammar for Conceptual Schema Diagrams," *Eur. J. Inf. Syst.*, Vol. 6, No. 2, pp. 147-170.
- Wieringa, R. (1998). "A Survey of Structured and Object-Oriented Software Specification Methods and Techniques," *ACM Computing Surveys*, Vol.30, No. 4, pp. 459-527.
- Wilson, J., Hoskin, N., & Nosek, J. (1993). "The Benefits of Collaboration for Student Programmers," 24th SIGCSE Technical Symposium on Computer Science Education, pp. 160-164.
- Wilson, W., Rosenberg, L. & Hyatt, L. (1997). "Automated Analysis of Requirement Specifications," *Proceedings of the International Conference on Software Engineering*.
- Wirfs-Brock, R. (1990). "Tutorial Notes for Responsibility-Driven Design," *OOPSLA*, pp. 1-8.
- Wu, C.C., Dale, N.B., & Bethel, L.J. (1998). "Conceptual Models and Cognitive Learning Styles in Teaching Recursion," *SIGSCE*, Atlanta, Georgia, pp. 292-296.
- Yoo, J. (2000). "Relationship Analysis," Ph.D. Dissertation, Rutgers University.
- Zahniseer, R. (1990). "Building Software in Groups," *American Programmer*, Vol. 3, pp. 1-8.